versione
8.x

Davide Copelli

Angular

100% Operativo

Da zero alla realizzazione di una Web APP, in 24 ore!

Quinta Edizione

Collana: CreareApp Vol.1

Davide Copelli

Collana: CreareApp Vol.1

Via Natisone 2 – 35010 Vigodarzere (PD)

Url: https://www.dcopelli.it/ – Email: web@dcopelli.it

Editor: Indipendently Published

Redazione: Francesca e Chiara

Produzione: Davide C., Luca F., Massimo P.

Prima Edizione: luglio 2017

Seconda Edizione: febbraio 2018

Terza Edizione: marzo 2018

Quarta Edizione: settembre 2018

Quinta Edizione: giugno 2019

ISBN: 9781794295926

A Cip per la pazienza,

a Mery per i succulenti pranzi,

a Giancarlo per la costanza!

La semplicità è l'estrema perfezione

(Leonardo da Vinci)

Prefazione

Ciao e benvenuto al primo Corso Operativo su Angular, che ti aiuterà a capire come

sfruttare questo framework per iniziare a sviluppare le tue prime semplici WebApp.

Mi chiamo Davide Copelli e sarò il tuo docente per le prossime

24 ore , o se

preferisci, per le prossime 200 pagine circa!

Per chi fosse abituato a sentire la mia voce, non preoccuparti. A breve saranno online

anche dei video riassuntivi per ogni capitolo, che potrai consultare nei portali indicati

qui sotto, oltre che rimanere aggiornato sulle novità di Angular.

Per ulteriori approfondimenti, consulta i due siti qui sotto, di cui sono **fondatore**:

 Video-Corsi.com: Il primo portale di e

-learning nato in Italia, con

MasterClass Operative sulle tecnologie web in Video

 CreareApp.com: Il sito de dicato allo sviluppo di APP e non solo , con corsi, tutorial, laboratorio codice, riservata solo ai miei clienti!

Avvertenza importante da leggere!

Il libro, essendo "operativo", presuppone che tu abbia delle

discrete basi di

programmazione JavaScript, TypeScript, RxJS soprattutto per ciò che concerne la

terminologia legata ai concetti di

Oggetto, Classe, Observable, oltre che buone basi

di HTML, CSS e linguaggi lato server. Nel libro faremo riferimento ad alcuni

esempi con PHP.

Se ritieni di avere già queste conoscenze, allora puoi proseguire, diversamente ti

consiglio di seguire qualche corso, o direttamente nel mio portale Video-Corsi.com o

su CreareApp.com, dove potrai trovare articoli specifici su Angular e TypeScript.

Il libro è una sorta di guida operativa per **"rompere il ghiaccio" con Angular**, ed è

stato volutamente concepito così.

Non vuole essere un manuale su Angular

e

TypeScript e sulla relativa storia (ce ne sono fino troppi!), n é tantomeno un libro

sulla programmazione, con tutta la teoria annessa (e se hai mai studiato qualche libro

universitario, sai a cosa mi riferisco).

In sostanza, non troverai una sola riga che parli della storia di Angular, di cos"è un

array, un framework MVC , una SPA , un Observable e tante parole che avrebbero

farcito il libro di numerose pagine superflue per l"obiettivo che mi sono prefissato.

Con semplici ricerche su Google, puoi ovviare a queste lacune facilmente.

Questo approccio è quello ho voluto adottare **circa 15 anni fa**, quando ho fondato il

portale Video-Corsi.com, dopo aver conseguito una laurea magistrale in Ingegneria Elettronica a Padova, e

dopo aver "consumato" decine di quaderni con appunti

teorici.

Dalle migliai a di commenti raccolti, posso affermare con una buona dose di

sicurezza, che questa è la strada che preferiscono i miei discenti e spero anche tu.

Feedback o commenti

Per scrivere questo libro, mi sono "sporcato le mani"

fin dalla prima versione di

AngularJS e poi Angular , con numerosi test e pratiche sul campo. La semplice APP

che realizzeremo è stata sviluppata durante

tre incontri in aula

con sette

professionisti del settore , tenuti a Padova, della durata di 8 ore

circa, ciascuno:

questo il motivo per cui nel titolo, ho inserito “da zero a 24 ore” (più o meno!).

Come dicevo, ho cercato di renderlo il più pratico e op erativo possibile, tralasciando

decine di pagine di approfondimenti tecnici, che per mia esperienza, spesso creano

un *blocco mentale* nell'apprendimento di una nuova tecnologia ed è utile

approfondire in un secondo momento. Nonostante questo, il libro super a le 200

pagine, questo per farti capire che “nulla è stato lasciato al caso”!

Sulla base di queste considerazioni, t "invito a lasciarmi un feedback su Amazon. Un

commento positivo in più, non mi permetterà certo di diventare “milionario”, ma è

importante per il morale, quindi un grazie fin da ora. Risparmiami però commenti su

presunti errori di ortografia (so di non essere Manzoni) o sul codice (vedi indicazioni

qui sotto)

Dove scaricare il codice ed errata corrige

Il codice lo puoi **scaricare gratuitamente**, seguendo le istruzioni che trovi in fondo

al libro, nella sezione “Conclusioni e Codice personale”

. Angular si aggiorna

rapidamente, quindi è probabile che

dall"acquisto del libro ai primi test, sia cambiato

già qualcosa. Fai sempre riferimento al cod ice che trovi nella tua area privata (vedi

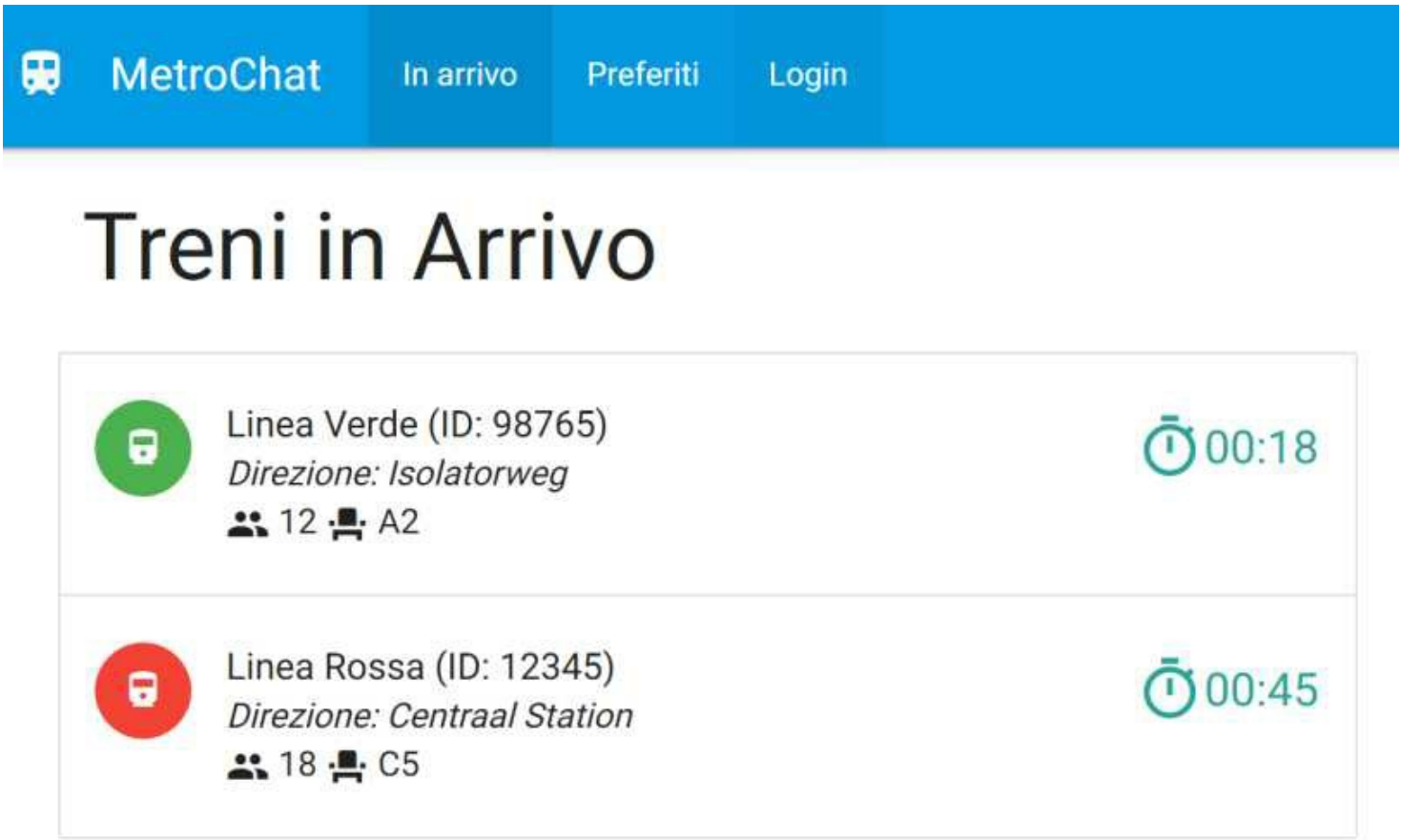
fine libro)

Buona lettura e buon lavoro!

Davide Copelli {ing}

Indice

Capitolo 5 - **Cambiare lo stile di elementi del DOM**



1

Che cosa realizzeremo?

L"applicazione di fantasia che progetteremo i n queste pagine,

l"ho chiamata

“**MetroChat**” e consiste di 4 “pagine” che puoi vedere completa di grafica a questo

link:

http://www.dcopelli.it/metrochat/

Nella prima schermata , potrai visualizzare la l ista dei treni in partenza presso una

determinata stazione metropolitana, con la possibilità di selezionar ne uno, ed entrare

così in chat con gli utenti - di quel treno - che stanno chattando.

Per ogni treno , sarà possibile visualizzare un conto alla roves cia, che permetterà

all'utente di conoscere il tempo mancante alla partenza.

Cliccando in corrispondenza ad un particolare treno, si entrerà nella schermata di

dettaglio, in cui saranno riepilogati i dati del treno e

visualizzati tutti i messaggi

scambiati tra gli utenti di quel treno.

Chat

Metro Chat

Linea Rossa (ID: 12345)
Direzione: Centraal Station
15 A2


Oggi è il mio primo giorno di lavoro e ho dimenticato la merenda!


Se vuoi ti porto un panino con il salame!


Ciao Martina, piacere di...incontrarti. Come stai?


Ciao Angela, che dici se ci vediamo sta sera, dopo lezione?

Messaggio

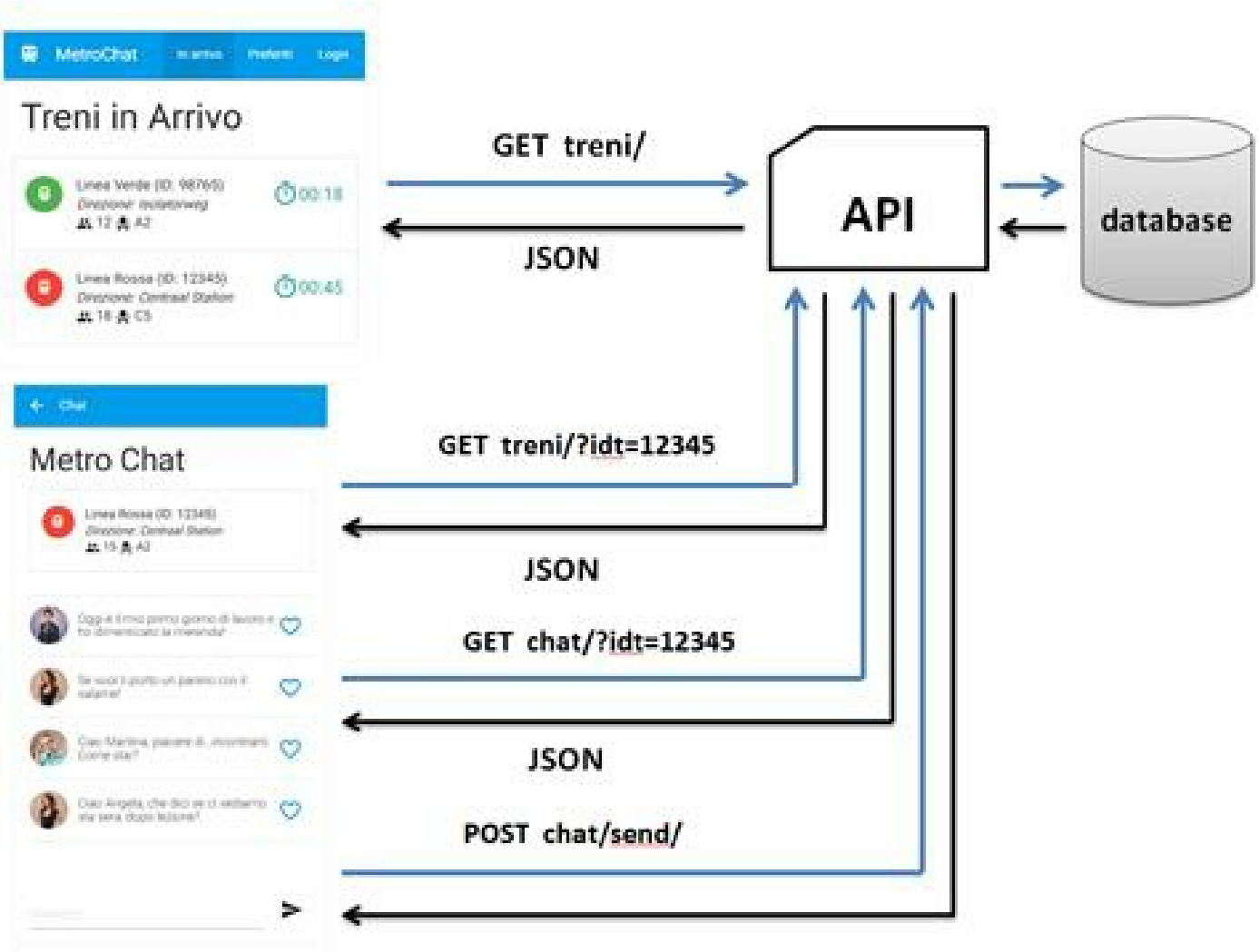
2

Per ogni messaggio di chat, sarà possibile visualizzare l'utente che l'ha spedito, e

inserirlo tra la lista dei preferiti, in modo da avviare una chat privata.

Quest"ultima funzionalità sarà proposta come esercizio finale per ricevere un

attestato di partecipazione.



3

Per evitare di appesantire la spiegazione, alcuni dettagli saranno omessi, come la

rappresentazione grafica dei diversi

elementi, che nella spiegazione sarà lasciata allo

stato grezzo, così come l'identifica

zione dell'utente e della posizione da cui

recuperare la lista dei treni o metro.

Il codice CSS e HTML completo dell'intera applicazione sarà comunque disponibile

per il download dalla tua area privata, previa registrazione (vedi fino libro).

Inoltre, per il recupero dei dati, utilizzeremo un'API d "appoggio creata ad hoc, che

non farà parte delle spiegazioni, ma c he potrai sfruttar e, per testare le funzionalità

dell"applicazione (vedi appendice).

Capitolo 1

Settings dell’ambiente di lavoro

1.1 Da dove partire: l'ambiente di test locale

Come per tutti i nuovi linguaggi di programmazione, che in 20 anni di attività sul

web ho dovuto imparare, anche per il **framework Angular**, il primo passo da fare è

dotarsi dell'ambiente su cui effettuare dei test.

Se per imparare il linguaggio HTML è sufficiente un

browser e un software per

scrivere del testo, per Angular, così come per altri framework basati su JavaScript è

necessario installare **Node.js e npm**.

Per ora non ci preoccuperemo di capire

a cosa servono esa

ttamente e che

caratteristiche hanno, perché li useremo solo per installare nel computer una serie di

strumenti utili alla creazione di nuovi progetti.

Agli inizi dello sviluppo di Angular, gran parte del lavoro di configurazione, doveva

essere fatto manua lmente. Ora, con una sola riga di comando, creeremo un progetto

completo da cui partire per iniziare

a inserire tutte le modifiche del caso, un

po"

come se avessimo il classico file html

di una pagina web , con già inseriti i tag

<html><head><body> etc.

Impostare l'ambiente di sviluppo sul proprio pc

Come dicevamo , il primo passo è installare le librerie Node.js e

npm. E'

un'operazione da 10 secondi . Tipicamente non sono presenti nei computer a livello

predefinito.

5

1) Installare Node.js e Npm

Collegati al link del sito ufficiale di Node.js , in particolare alla sezione di download

(https://nodejs.org/en/download/) e seleziona il sistema operativo su cui installerai i

due software.

Il file che salverai sul tuo pc, contiene sia Node.js che Npm.

NB: Nel caso tu abbia già installato queste librerie in precedenza, accertati di

avere delle versioni adeguate. Per Node.js deve essere superiore alla 12 e

per nmp superiore alla 3.

Il test lo puoi fare lanciando le seguenti righe all'interno della classica finestra del

prompt dei comandi, sia su Window, sia su Mac e Linux.

node -v // per verificare la versione di node.js installata

npm -v // per verificare la versione di npm installata

2) Installare Angular CLI

Il passo successivo è quello di installare un insieme di librerie necessarie a eseguire

dei comandi direttamente dalla finestra terminale.

Per dare un'occhiata all"attuale versione delle librerie e a eventuali operazioni da fare

per aggiornare progetti Angular datati, collegati al link ufficiale su

Github qui:

https://cli.angular.io/

Dovrai allora scrivere questa riga:

npm install -g @angular/cli

NB: Fai attenzione a rispettare gli spazi presenti nella stringa e armati di

pazienza, perché l'installazione potrebbe richiedere qualche minuto.

```
C:\>node -v
v10.16.0

C:\>npm -v
4.1.1

C:\>npm install -g @angular/cli
```

```
C:\Users\videocorsi>cd Desktop

C:\Users\videocorsi\Desktop>cd TestAngular

C:\Users\videocorsi\Desktop\testAngular>ng new miapp
```

6

3) Creare un nuovo progetto Angular

Per creare l'insieme dei file necessari ad entrare nel vivo dello sviluppo della tua

prima applicazione, puoi usufruire del comando qui sot to, che ti permetterà di creare

un nuovo progetto dal nome *miapp:*

ng new miapp

Ti consiglio vivamente di creare prima una cartella di lavoro (es. TestAngular), e

spostarti qui dentro per creare il progetto.

Se non lo ricordi più, per cambiare la

posizione usando la finestra terminale, dovrai usare i classici **comandi DOS** : *cd*

nomecartella per entrare in una sottocartella, e *cd..* per spostarti nella

cartella padre.

A seconda della versione di angular CLI, forse ti verrà chiesto se vuoi abilitare il

routing (rispondi no) e se usare i css (rispondi selezionando la voce) o oltro.

NB: Il nome da dare all'applicazione, deve essere scritto con lettere

minuscole, senza inserire spazi, caratteri alfanumerici, simbolo di underscore

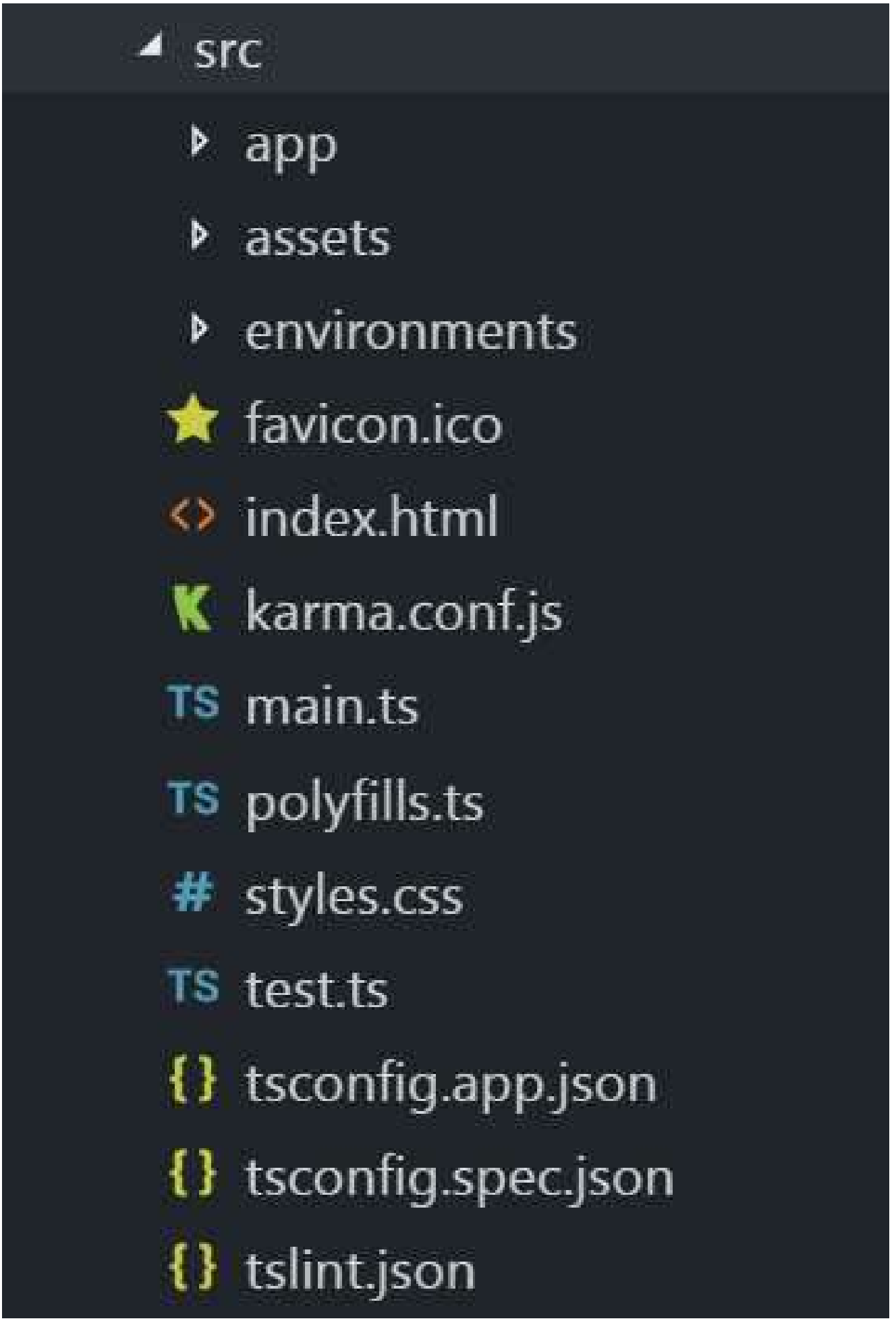
(trattino basso).

In alternativa, puoi usare l"opzione -*S*, per limitare il numero di file che

verranno

installati nel progetto.

ng new miapp -S


src
app
assets
environments
favicon.ico
index.html
karma.conf.js
main.ts
polyfills.ts
styles.css
test.ts
tsconfig.app.json
tsconfig.spec.json
tslint.json

Armati di pazienza, perch é questo passaggio potrebbe richiedere alcune decine di

secondi in quanto devono essere scaricati diversi pacchetti npm.

4) Avviare il server e caricare la pagina nel browser

Arrivati a questo punto, nella cartella in cui hai creato il progetto, troverai la

sottocartella ***src***, che conterrà tutta una serie di file, suddivisi tra file di

configurazione e file dell'app, c he

impareremo a capire in dettaglio nelle prossime

lezioni.

Purtroppo non è possibile lanciare l'applicazione cliccando su uno specifico file

index.html, come avviene per le classiche pagine web, ma dovrai prima di tutto

avviare un server locale integra to, che ti permetterà di aprire l'applicazione da un

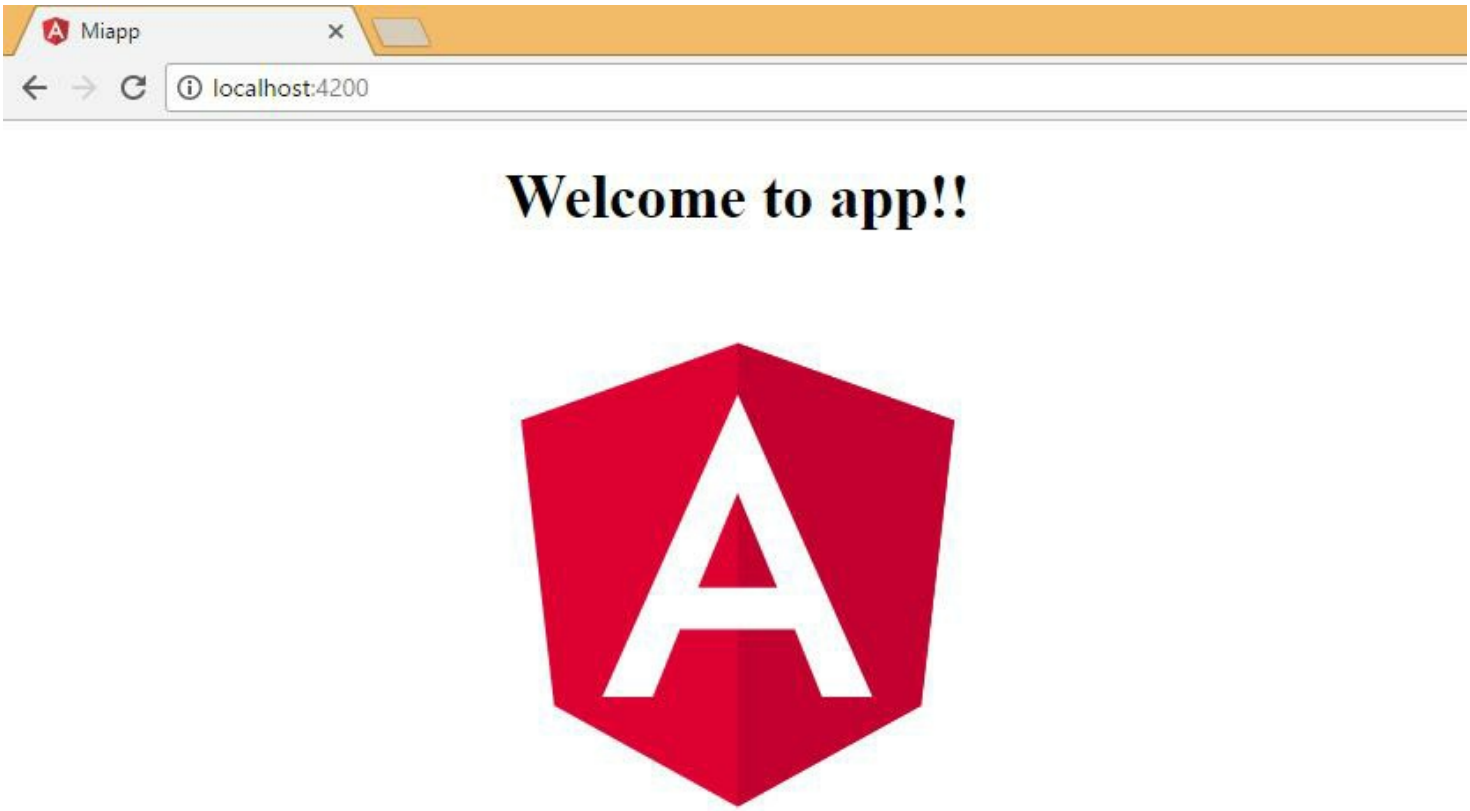
indirizzo web specifico, in particolare dall'indirizzo http://localhost:4200/

NB: Fai attenzione a inserire la porta ossia il numero :4200/

Sempre con l'ausilio della finestra terminale, dovrai spostarti all'interno della cartella

in cui è stato creato il progetto, nel nostro caso all'interno della cartella *miapp*:

cd miapp



8

e poi avviare il server di produzione

locale, che c i servirà per interpretare

correttamente le pagine:

ng serve --open

NB: Ricordati che devi sempre essere all'interno di una cartella in cui è

presente un progetto Angular. L’opzione --open (occhio ai due simboli di

“meno”), serve per aprire in automatico la finestra del browser predefinito). Ti

consiglio di usare Chrome per tutti i test.

Grazie a questo comando, ogni modifica che introdurremo ai diversi file dell"app,

automaticamente si rifletterà nella pagina visualizzata dal browser.

NB: Spesso può capitare che le ultime

versioni di angular-cli non funzionino

correttamente, quindi nel caso la pagina non dovesse aggiornarsi, dovrai

armarti di pazienza e attendere che sia corretto il bug e aspettare la nuova

versione o installare quelle precedenti.

Bene, se il browser non dovesse aprirsi, dovrai collegarti al link qui sotto:

http://localhost:4200/

per visualizzare il messaggio " Welcome to app!" (o simile) , che ti confermerà la

correttezza di tutti i passaggi fatti fino ad

ora per configurare l"ambiente di sviluppo

in locale.

9

Il file che viene

visualizzato dal browser , è proprio il file

index.html, presente

all'interno della cartella radice del tuo progetto

(cartella *src*). Se tu sbirciassi

all'interno del codice sorgente d i questa pagina , non troveresti gra nché: solo tag

html. Andando però a guardare con lo strumento "ispeziona documento" del

browser, troveresti anche una serie di file JavaScript, iniettati nel codice della pagina

in fase di compilazione.

In realtà, vedremo che lo "strano" tag <app-root> presente all'interno della pagina,

non è un classico tag HTML, ma un “potenziamento” del codice HTML.

Non appena il browser carica la pagina, Angular trasform a questo tag, in modo che

sia correttamente interpretato, sulla base d "informazioni che sarà nostra c ura inserire

all"interno dell"applicazione.

5) Terminare l'esecuzione del server

Non appena avrai concluso tutti i test sulla tua applicazione, conviene arrestare il

sever locale avviato in precedenza con *ng serve*, questo per non rallentare il proprio

pc con programmi che non servono più. Per farlo,

dovrai chiudere la finestra

terminale oppure digitare la combinazione di tasti CTRL+C.

E' chiaro che per poter nuovamente testare l'app, dovrai aprire la finestra terminale,

spostarti nella cartella del tuo pro getto e digitare nuovamente la riga vista nel punto

4.

Abbiamo imparato i pochi passaggi

necessari per preparare il proprio computer

all'esecuzione e test di un'applicazione creata con il framework Angular.

Questi

passaggi li dovrai ripetere per ogni nuova applicazione.

1.2 I migliori IDE per scrivere applicazioni

Dopo aver configurato il tuo computer al fine di poter testare l a prima applicazione

Angular, il passo successivo è quello di

scegliere

quale “editor” utilizzare per

scrivere il codice.

Qui l'offerta non manca ma , come sempre, la scelta deve cadere sul migliore, e con

migliore intendo quello strumento in grado di aiutarti a

scrivere meno codice

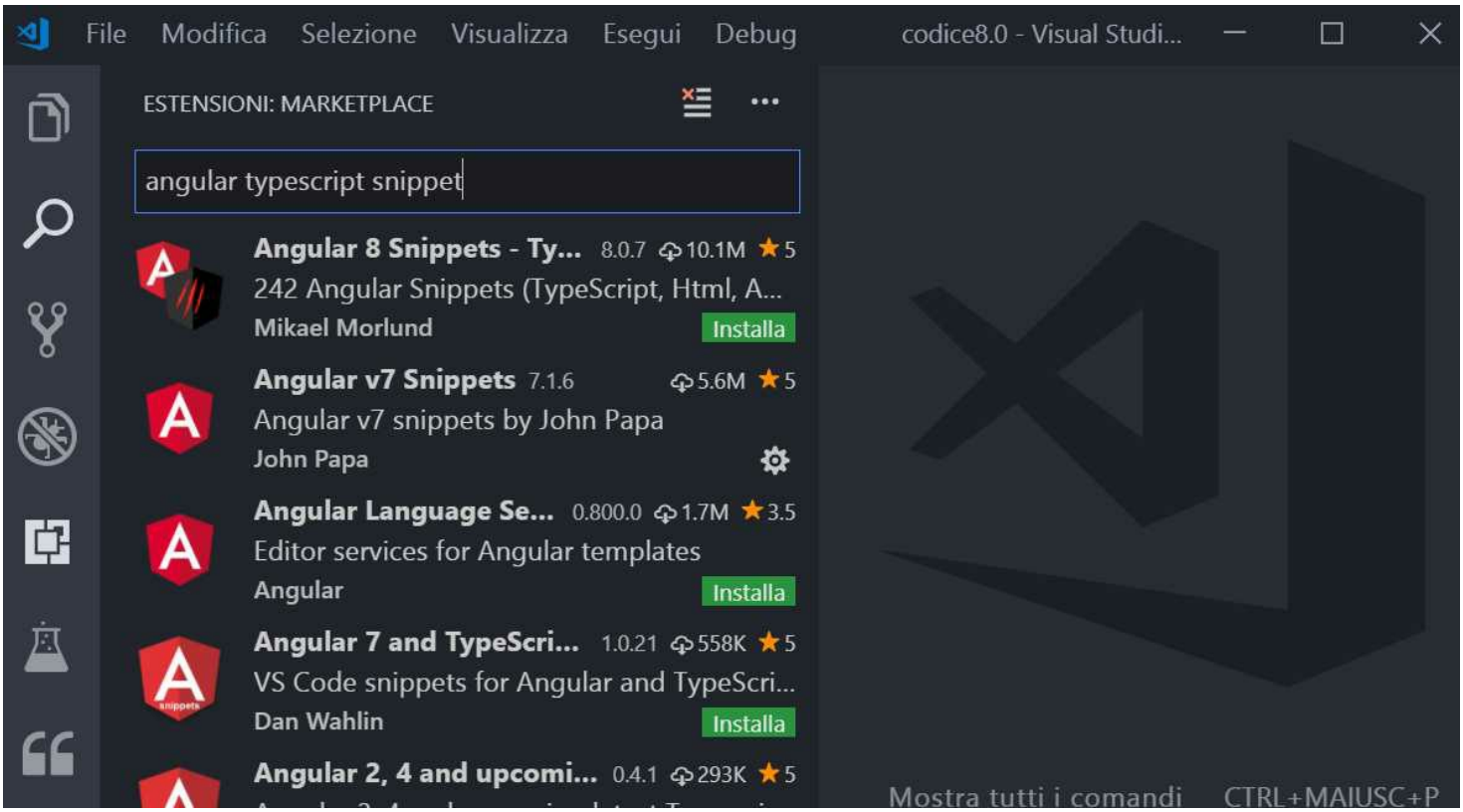
possibile, grazie all'auto - completamento di questo, a eventuali “snippets” e ad aiuti

visivi, come la co

lorazione dei diversi elementi del linguaggio.

Il linguaggio

preferenziale per scrivere applicazioni Angular è TypeScript.



10

Ecco allora la mia personale classifica dei migliori editor di testo , GRATUITI e non,

per scrivere applicazioni Angular con TypeScript.

Visual Studio Code - Gratuito

Per molti anni, la Microsoft, ha offerto un sof tware a pagamento, il famoso “ Visual

Studio”, rivolto principalmente a gli sviluppatori di applicazioni ASP.net

. P oi è

passata alla versione gratuita , con “Visual Web Developer ”, e infine ha creato

“Visual Studio Code”.

A mio avviso è il software più adatto per la scrittura di codice , grazie alla presenza

della funzionalità di "IntelliSe nse" ereditata da Visual Studio. Lo puoi scaricare da

questo link: https://code.visualstudio.com/

Altra funzionalità interessante, è la possibilità di aggiungere delle

estensioni in

grado di potenziarne il funzionamento. L"icona da cliccare è quella

Ti consiglio di installare, o quella sviluppata da John Papa o da Mikael Morlund. Ti

permetteranno di risparmiare molto tempo nella scrittura di pezzi di codice Angular,

grazie all'inserimento di snippet di codice. Le trovi cercando la voce "Angular

TypeScript Snippet". Fai attenzione che non tutte le este nsioni vengono aggiornate

frequentemente sulla base delle nuove versioni di Angular.

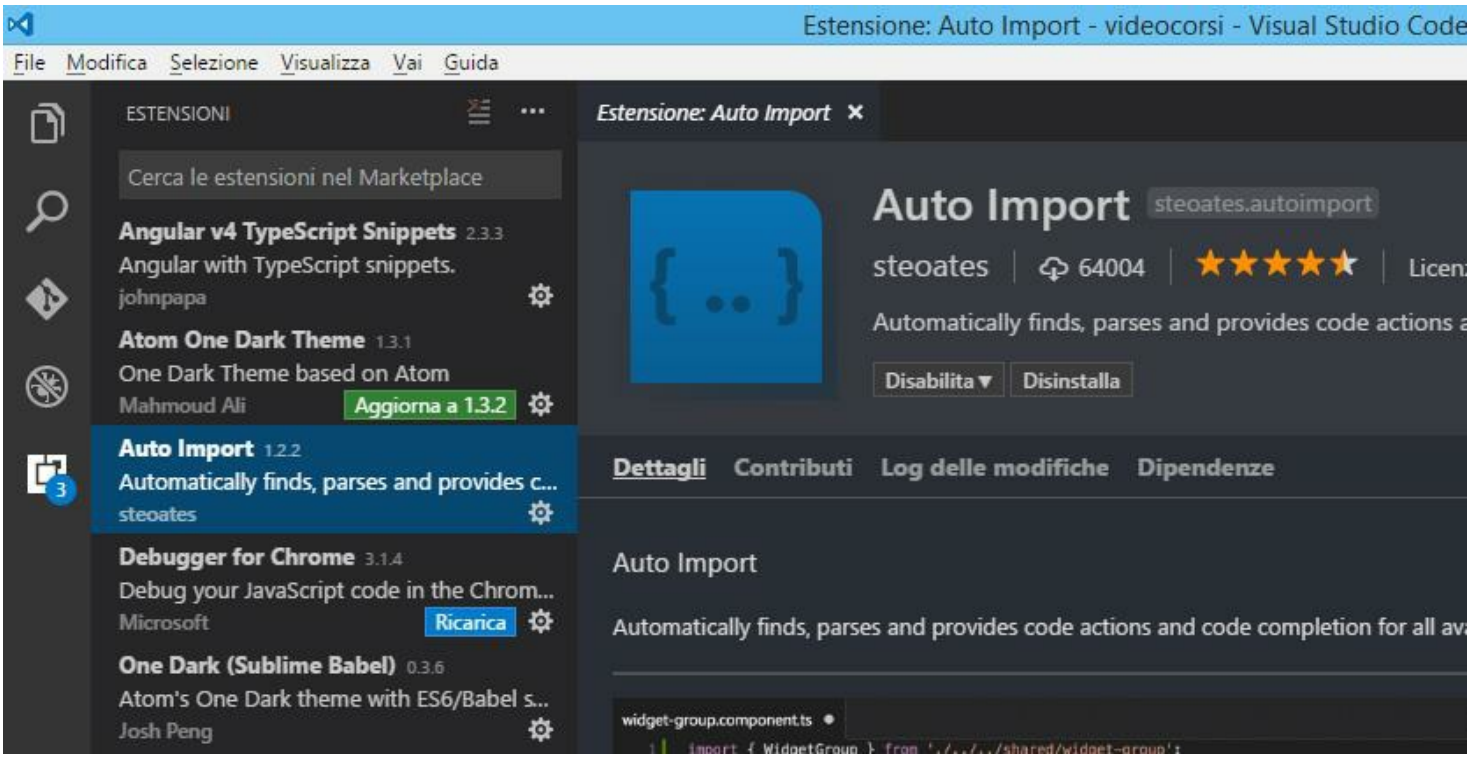
Chiaramente ne esistono decine e dovrai scegliere quella che più si adatta al tuo stile.

Per installarla, dovrai cliccare sull'ultima icona della barra laterale sinistra,

oppure

digitare i tasti CTRL+P e poi digitare nella barra di ricerca il tipo di estensione che

vorresti.



Una volta installata ad esempio quella di John Papa , l"uso è molto semplice, perché

sarà sufficiente iniziare a digitare la sigla a

-, per vedere car icare una finestra con

tutta una lista di nomi, legati a particolari pezzi di codice

(snippets), che

frequentemente dovrai inserire in un"applicazione Angular e che saranno oggetto del

corso. Ti invito a fare qualche prova per vedere cosa viene inserito nella pagina.

Altra estensione importante è quella che ti permette di auto completare il codice, con

l'importazione delle diverse librerie.

Questa estensione è particolarmente utile all"inizio, quando ancora non si ha

dimestichezza con i diversi package pres

enti in Angular. L"estensione si chiama

”AutoImport” e una volta installata, ti permetterà di aggiungere in automatico le

diverse librerie non appena aggiungerai un oggetto, un componente etc.

ATOM - Gratuito

L'editor “Atom”, da molti anni è considerato uno dei migliori software open source

per la scrittura di programmi, compatibile con diverse tipologie di linguaggio, grazie

alle decine di plugin.

Il software lo puoi scaricare da questo link: https://atom.io/. Dopo averl o scaricato, devi ricordardi di installare anche il plugin "TypeScript Plugin".

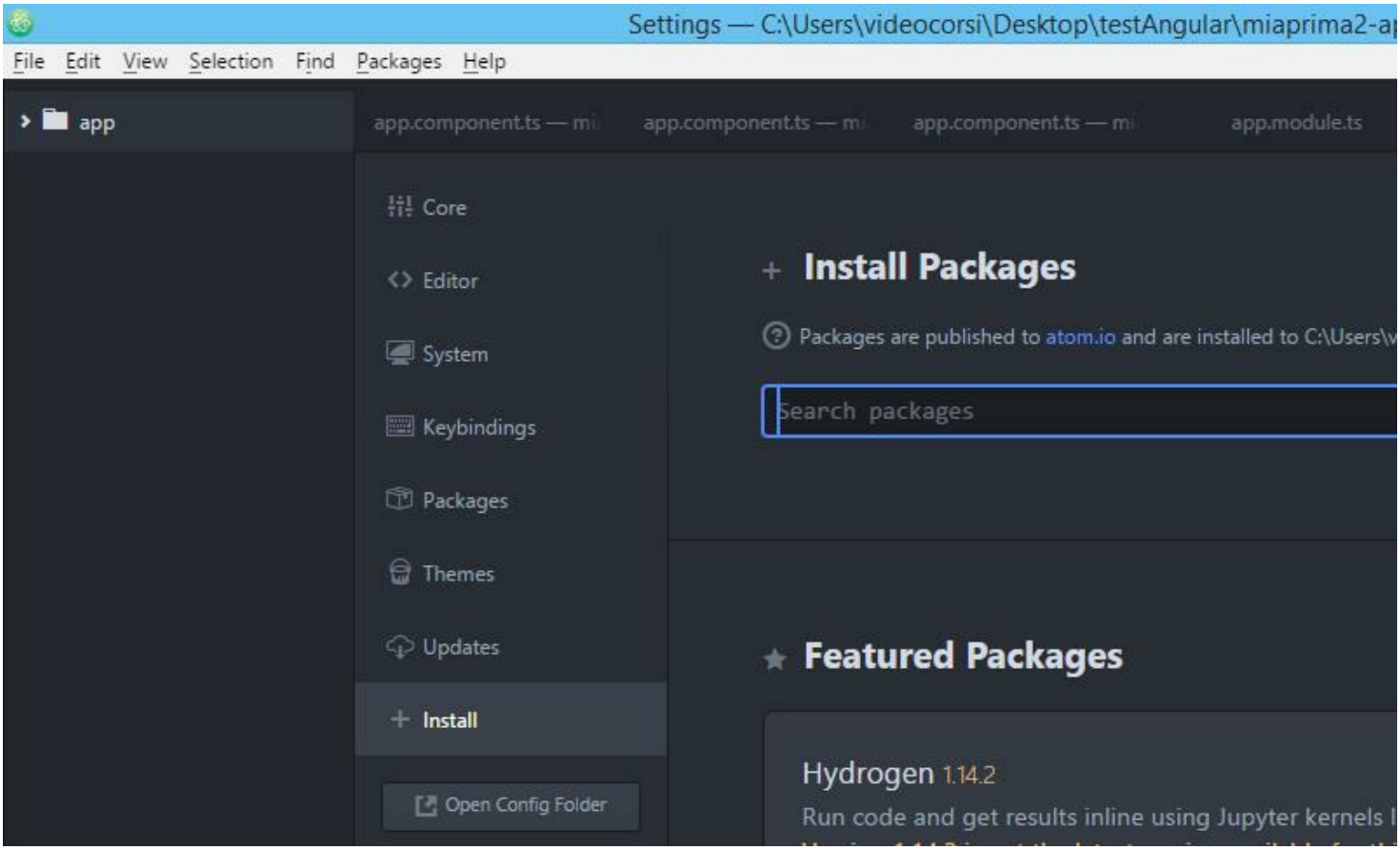
L'installazione di un plugin è un passaggio molto semplice e veloce, perch

é dovrai

solo accedere al menu "File>Settings" e poi selezionare , nella colonna centrale alla

pagina, la voce "Install" per accedere ad una finestra simile a quella qui sotto, da

dove potrai cercare i diversi plugin da installare.



Sublime Text - A pagamento

Quest'ultimo è un sof tware che tutti i possessori di Mac, conoscono molto bene.

Esiste tuttavia an che la versione

adattata per windows. Lo puoi scaricare da questo

link: http://www.sublimetext.com/3.
Oltre al software, dovrai ricordarti di installare anche il plugin "TypeScript-Sublime"

WebStorm - A pagamento

Altro software degno di nota, ma ahimè anch'esso a pagamento, è “WebStorm”. Puoi

scaricarne una versione demo di 30 gg da questo link:

https://www.jetbrains.com/webstorm/do

1.3 Comandi Utili CLI

Abbiamo visto che lo strumento per creare un nuovo progetto è Angular CLI

accessibile tramite il prompt dei comandi.

Qui sotto puoi trovare una lista di azioni utili per creare i diversi elementi che

compongono un'applicazione Angular e che ti perm etteranno di rispa rmiare tempo

nella scrittura del codice.

Le scritte in maiuscolo, sono quelle che

dovrai personalizzare.

13

a) Creare una nuova App Angular

ng new NOME // o in alternativa ng new NOME --minimal

b) Testare l'applicazione in locale

Il comando deve essere eseguito spostandosi nella cartella del progetto:

ng serve --open

c) Creare nuovi elementi dell'App

Per creare i diversi elementi, dovrai

sempre spostarti nella cartella in cui vuoi che

siano creati e lanciare il comando ng generate (o l"alias ng g), seguito dal tipo di

elemento e dal NOME da assegnargli:

Componente ng g component NOME

Direttiva

ng g directive NOME

Pipe

ng g pipe NOME

Service

ng g service NOME

Class

ng g class NOME

Interfaccia

ng g interface NOME

Modulo

ng g module NOME

d) Aggiornare un progetto alle ultime versioni di Angular

Una delle caratteristiche più “stressanti” per chi deve imparare una nuova tecnologia

è il continuo e inevitabile aggiornamento delle versioni. Angular amplifica questo

stress, in quanto viene aggiornato a cadenza mensile.

E" importante quindi sapere come aggiornare il proprio ambiente di test per rimanere

al passo con il rilascio delle nuove versioni.

Se ti iscriverai alla “ WEB StartUniversity (WEBSU)”, potrai rimanere aggiornato

con tutti i co dici che svilupperemo nel libro, oltre ai diversi tutorial e corsi che

svilupperemo nel tempo.

Il processo di aggiornamento varia a

seconda della versione in cui è stato sviluppato

il progetto di partenza.

Collegandoti a questo indirizzo: https://update.angular.io/, p otrai ricevere istruzioni dettagliate sui diversi passi da seguire per aggiornare il tuo progetto.

Se provieni da vecchie versioni <7.x, il punto di partenza è aggiornare Angular CLI,

sia in locale a l progetto che a livello globale, e cambiare la struttura del file

angular.json

Apri la finestra terminale, entra nella cartella del tuo progetto e scrivi, una alla volta,

le righe seguenti, schiacciando il tasto invio al termine dell"inserimento di ognuna:

ng update @angular/cli

ng update @angular/core

Nel caso dovessero comparire degli errori puoi provare la riga:

ng update --all --force // solo in casi estremi !!!

Ricordati che Angular si aggiorna ogni

5/6 mesi, quindi se vuoi mantenere il tu

o

progetto aggiornato, ricordati di rieseguire questi passaggi. Per verificare la versione

attualmente installata sul tuo pc, sia di CLI sia di Angular, digita:

ng version

e) Creare la versione di produzione da pubblicare nel proprio sito

Per progetti di bassa complessità come il nostro, sarà sufficiente scrivere la riga qui

sotto e poi trasferire tutti i file locali che si creeranno nella cartella “*dist*”, all"interno

della cartella radice del tuo sito web:

ng build --prod

15

Capitolo 2

Il motore di un’app Angular

2.1 La struttura di un’App Angular

Per creare un' app in Angular è necessario avere un p

o" di dimestichezza con

JavaScript e i concetti della programmazione ad oggetti , quindi è consigliabile aver

seguito un corso che tratta di JavaScript.

Questo perché? Perché il linguaggio da preferire per lo sviluppo, è TypeScript, che è

un"estensione di JavaScript creata da Microsoft.

Infatti, il cuore di ogni app Angular, è costituito da quelli che

vengono chiamati

"componenti", ossia un insieme di tag, estensione del codice HTML, con cui riesco a

realizzare ogni pagina della mia applicazione.

I componenti se vogliamo, sono

un' **evoluzione** del linguaggio HTML e stanno

diventando sempre più importanti in molti framework (vedi Polymer

, React, etc),

poiché ora la maggior parte dei browser è in grado di supportarli.

Per maggiori dettagli, visita la pagina dedicata: WebComponents.org

Se ad esempio, prendessi in esame l'interfaccia di fig.2.1, potrei dire che è costitui ta

da un certo numero di componenti: da i pulsanti del menu, dalla singola ricetta, dai

pulsanti “favoriti” e così via.

Ognuno di questi pu ò contenere all"interno altri componente, legati tra di loro con

una relazione “genitore/figlio” o “padre/figlio”.

GigRecipes

All | New | Search | Favorites

Sliders Four Ways

General Tso's Chicken

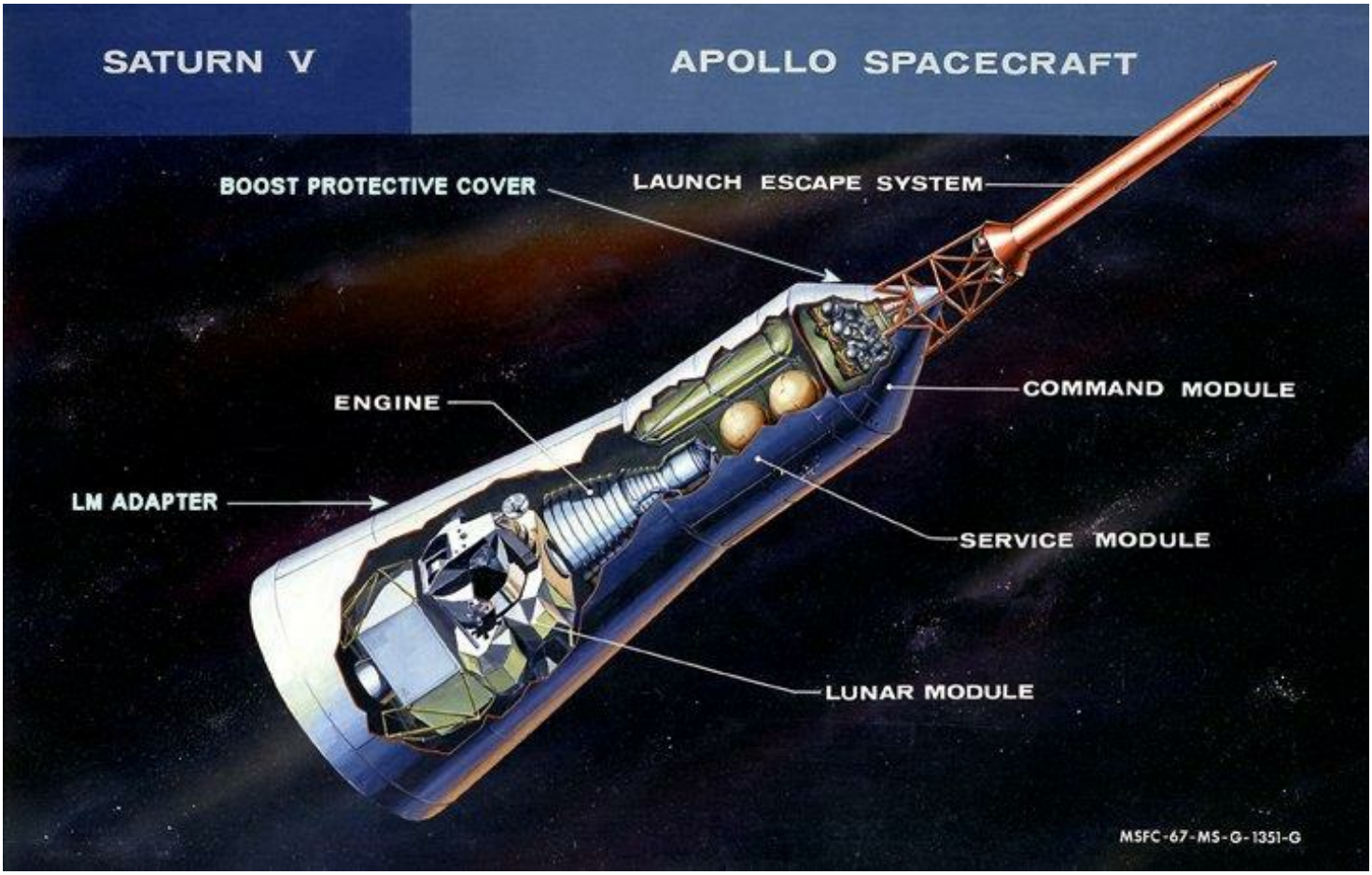


fig.2.1

In Angular, ogni componente sarà caratterizzato da tre **elementi fondamentali:**

1. il **nome (selettore)**

2. la sua rappresentazione **grafica (template o vista)**

3. le funzionalità che deve avere (**classe o modello**)

Un'app, si può paragonare all'astronave “Apollo 13”, che come si vede nell'omonimo

film, era costituita da più moduli, tra cui il lem, quello che gli astronauti usarono per

ricongiungersi con il modulo in orbita per poi ritornare a terra.

Apollo Spacecraft Wikipedia

La rappresentazione grafica è sicur amente la parte meno complessa, in quanto basta

che tu conosca la sintassi del codice

HTML, o tu sappia già usare

un framework

responsive, o Material Angular o Ionic, (vedi i miei corsi WebSU) e il gioco è fatto.

L'unica cosa da approfondire, è capire come

sia possibile iniettare dei valori, dal

corpo del componente o classe, al template e viceversa. Di tutto questo parleremo in

una sezione specifica del libro.

Chiarito che un'app Angular è costituita da tanti componenti e da alcuni moduli, la

domanda che p otresti farti è: ma come

si **crea una pagina** per un'applicazione

Angular? Ad esempio come posso creare l"ipotetica "homepage" della mia app?

Ebbene, come ogni razzo spaziale, che contiene tutte le attrezzature per la spedizione

nello spazio, anche ogni applicazione Angular, deve essere progettata partendo da un

file iniziale, che raggrupperà tutto il necessario per far partire l'applicazione, una

sorta di equivalente pagina di ingresso di un sito web.

Per convenzione, questa pagina dovrà avere nome, non Apollo 13 ma index.html,

e si progetterà come una semplice pagine web, con la differenza che al suo interno

dovrò inserire sia i classici tag del linguaggio HTML, che quelli legati ai componenti

da me progettati.

Il passo successivo sarà quello di far “partire il razzo ”, quindi avrai bisogno di un

“sistema di propulsione” (modulo radice) e di una “miccia” (file main.ts)

Riassumendo, per far partire un'applicazione, sono necessari tre

elementi:

1. Un file index.html, con interno almeno un componente.

2. Un Modulo (file app.module.ts), equipaggiato con tutti gli elementi che

servono all'applicazione per la compilazione e il lancio.

3. Una "miccia" (file main.ts), ossia un file che faccia partire l'applicazione.

Come vedi, rispetto a un semplice sito web - in cui basta c reare una pagina html e

aprirla con un qualsiasi browser - il

passo iniziale è più complesso e richiede diverse

conoscenze, che cercheremo di approfondire nelle prossime pagine.

2.2 Il file index.html

La struttura della pagina principal e, come dicevamo, si crea con le stesse tecniche

con cui si progetta una classica pagina html:

<!DOCTYPE html>

<head>

// importazione di una serie di librerie Angular

</head>

<body>

<!--componente Angular-->

<ca-miomenu></ca-miomenu>

<h1>Articoli Angular</h1>

<!--componente Angular-->

<ca-listaarticoli></ca-listaarticoli>

</body>

</html>

Index.html

dove i tag html

<ca-listaarticoli> e <ca-miomenu>, sono i nomi che

sceglierò per identificare dei futuri **componenti Angular** da progettare.

NB : Il nome del componente può essere scelto a piacere, anche se per

convenzione si preferisce anteporre una sigla per distinguere i propri

componenti da quelli sviluppati da altri. Nel nostro caso, abbiamo scelto la

sigla ca-, che è l'iniziale del nostro sito web creareapp.com

NB: Non usare caratteri alfanumerici

(&%$ etc.) e lo spazio. Inserisci sempre

tutte lettere minuscole.

Come dovranno essere progettati questi ultimi?

Prima di tutto è necessario conoscere la sintassi per creare una classe in TypeScript e

poi è necessario conoscere la sintassi con cui si creano i componenti in Angular.

Questo però non è sufficiente, in quanto devi anche importare una serie di librerie

necessarie al corretto funzionamento, altrimenti il browser non sarà mai in grado di

capire cosa sostituire al posto di <ca-miomenu> o di <ca-listaarticoli>.

Non avere fretta, impareremo a fare tutte queste cose molto presto. Per ora

concentriamoci sui passi per far partire la nostra prima applicazione.

Dove e come si crea questo file?

Non appena sfrutti i comandi CLI visti

nel primo capitolo "Settings dell"Ambiente di

Lavoro", in automatico si genera questo file, che sarà presente all'interno della

cartella *src* del progetto.

File indice con Librerie ANGULAR caricate da SystemJS

Al fine di poter elaborare i nuovi tag htm l inseriti e capire come eseguire eventuali

azioni "nascoste" nella logica di ciascun componente, è necessario aggiungere

all'interno della cartella principale del

progetto, una serie di file di configurazione,

che fortunatamente sono aggiunti in automatic

o non appena creiamo un nuovo

progetto, sfruttando i comandi in linea visti in precedenza.

Fino alla versione beta.10 di angular-cli, il sistema usato per caricare una serie di

"pezzi" necessari al funzionamento dell'applicazione, si basavano sulla libreri

a

“SystemJS”. Quest'ultima è una libreria che ha lo scopo di

caricare dei moduli ,

ossia dei file che conterranno all'interno altri pezzi dell'applicazione oltre che una

serie di altri file collegati.

Era tipico quindi avere dei file indice come questo:

<!DOCTYPE html>

<head>

```
<title>Angular App</title>

<meta charset="UTF-8">

<meta name="viewport"
content="width=device-width, initial-
scale=1">

<!-- importazione di una serie di librerie
angular -->

<!-- Polyfill(s) for older browsers -->

<script src="node_modules/core-
js/client/shim.min.js"></script>

<script
src="node_modules/zone.js/dist/zone.js"
</script>
```

```
<script
src="node_modules/systemjs/dist/system
</script>

<!--Librerie per caricare il primo
Modulo dell'app-->

<script src="systemjs.config.js">
</script>

<script>

System.import('app').catch(function(err)
{ console.error(err);});

</script>

</head>
```

<body>

<app-root></app-root>

</body>

</html>

Index.html

20

File indice con Librerie ANGULAR caricate da WebPack

Sebbene tu possa continuare a usare vecchie versioni di angular

-cli, ti consiglio

vivamente di optare per le nuove versioni, che sfruttano l

a libreria “WebPack”, un

sistema di caricamento dei moduli dell'applicazione Angular, decisamente più sne llo

e agevole da usare rispetto a SistemJS.

Il primo beneficio, lo osserviamo subito guardando il codice sottostante, in cui ci si

accorge al volo, che sono completamente sparite diverse righe

interne a

meta tag

<head>.

<!DOCTYPE html>

<head>

<title>Angular App</title>

<meta charset="UTF-8">

<meta name="viewport"
content="width=device-width,initial-
scale=1">

</head>

<body>

<app-root>Loading</app-root>

</body>

</html>

index.html

Un passo avanti sicuramente verso una semplificazione rispetto al vecchio sistema.

2.3 Convenzioni per la progettazione del file

index.html

I più attenti avranno notato che, il file *index.html* creato tramite la linea di comando,

presenta all"interno un tag <app-root>, che ha le sembianze di un cl assico tag

HTML, ma non corrisponde ad alcuno dei tag di tua conoscenza.

Si tratta d el componente chiamato “**componente radice**”, e vedremo sarà uno degli

elementi chiave di un"applicazione.

<app-root>Loading</app-root>

Una delle caratteristiche di Angular, è la capacità di tenere separa ti i diversi pezzi d i

un"app, per rendere la fase di sviluppo e di test, semplificata.

Pertanto, non avremo mai una situazione come quella rappresentata nel codice visto

inizialmente, in cui nel file

index.html erano presenti più

componenti, ma si

partirà sempre dal codice simile a quello indicato qui sotto , in cui si intrave de un

unico componente, identificato dal tag **<app-root>** , il quale a sua volta conterrà

altri elementi HTML o altri componenti, con

una struttura tipica ad albero

rovesciato.

<!DOCTYPE html>

```
<head>

<title>Angular App</title>

<meta charset="UTF-8">

<meta name="viewport"
content="width=device-width,initial-
scale=1">

</head>

<body>

<app-root>Loading</app-root>

</body>

</html>
```

index.html

NB: Tutte le applicazioni Angular che progetterai, saranno caratterizzata

dall'avere un solo componente nel file index.html.

Cosa serve ad Angular oltre al file index.html?

Come detto in precedenza, così come accade per un razzo che ha bisogno di una

serie di elementi per partire (propellente, base di lancio , equipaggio, materiale etc .),

anche per un'applicazione Angular, dobbiamo avere un meccanismo che ci consenta

di mostrare il componente principale inserito nel file index.html.

Abbiamo bisogno di quello che si chiama in gergo tecnico “Modulo”, che conterr à

all'interno tutto l'occorrente per far funzionare l'applicazione. Questo modulo

è

chiamato “**Modulo Radice”** per distinguerlo da even tuali altri moduli,

di cui sarà

costituita un'applicazione complessa.

Vediamo allora come possa essere strutturato un file

di questo tipo e dove debba

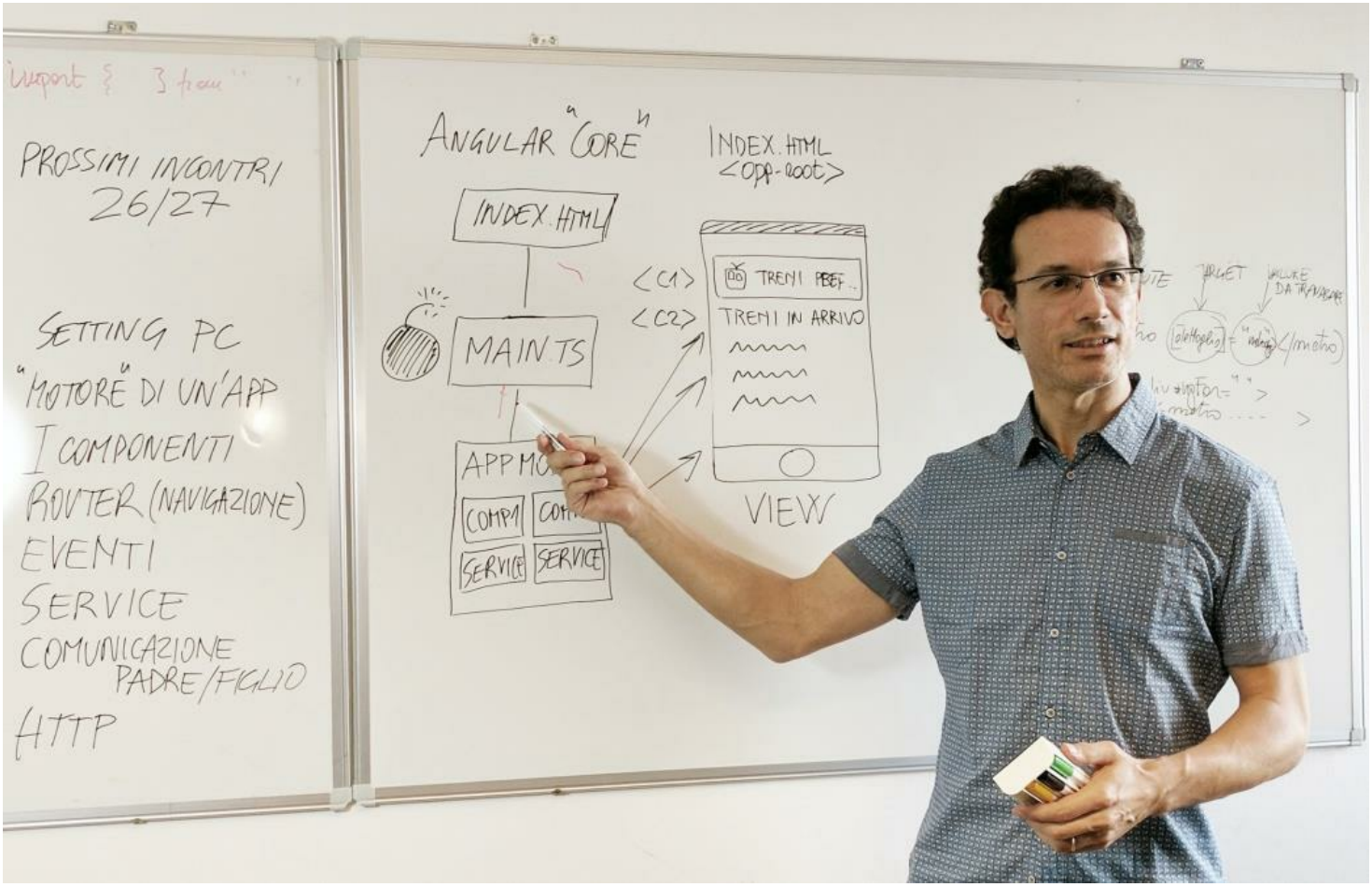
essere creato.

2.4 Il mio primo Modulo: AppModule

Se nel mondo dello sviluppo di siti web in HTML le pagine html sono il fulcro

dell"intero progetto, nel mondo dello sviluppo di applicativi Angular, la pagina

index.html è l'equivalente dell"homepage.



22

La grossa differenza rispetto allo sviluppo di siti web, è che dobbiamo includere una

serie di funzionalità che aiuteranno gli attuali browser a interpretare correttamente i

diversi elementi tipici di un'applicazione Angular, come i componenti, le direttive, i

servizi.

In sostanza d ovremo predisporre una sorta di “cabina di regia ”, che includa queste

informazioni.

Non prendere paura perché si tratta d "inserire solo due file. Il primo è una semplice

classe, “adornata” con il decorator e @NgModule() (vedremo cos'è un decoratore

prossimamente).

Questa tipologia di classe

, dicevamo in precedenza, è chiama

Modulo, e

un"applicazione Angular avrà sempre un modulo principale, chiamato **"Modulo**

Radice" o "AppModule".

Questo concetto potrebbe essere difficile da “digerire” e da ricordare, ma vedrai che ,

dopo aver visto i primi esempi, assimilerai questa terminologia

rapidamente.

In precedenza, avevamo detto che il modulo radice

“AppModule”, lo potevamo

paragonare al motore di un razzo, che inizia

lmente è spento ma che, grazie alla

“miccia” costituita dal file main.ts, sarà in grado di accendersi e recuperare quelle

risorse necessarie al funzionamento dell'app.

Qui sotto puoi vedere un mio schizzo sulla lavagna, durante una lezione.

23

Dove è salvato il primo modulo e con che estensione?

Il file è salvato all'interno di una sottocartella con un nome facile da ricordare: ***app***

(scritta in minuscolo). Qui dentro, vedremo, salveremo molti altri file, sempre con

una struttura a sottocartelle. L'estensione che dovrà avere sarà *.ts*, che è

l'estensione

di ogni file TypeScript. Il tipico nome usato per il modulo radice è app.module.ts

e questo è il motivo per cui spesso si chiama anche “AppModule”.

NB: Nota la presenza del punto di separazione tra la parola app e la parola

module. Questa convenzione, sarà usata anche per altri file (es. i

componenti).

La scelta di sfruttare questo meccanismo centralizzato per il caricamento delle

risorse necessarie all'applicazione, è uno dei punti di forza di Angular.

Come deve essere creato questo file?

Il metodo consigliato è quello di sfruttare la linea di comando, perch é non appena si

crea un nuovo progetto, sarà automaticamente creato anche questo file. Per ora non è

necessario ricordare a memoria la struttura delle informa zioni da inserire all'interno

ma solo cosa deve essere inserito al fine di far funzionare l'applicazione.

Sintassi del file *app.module.ts*

Essendo una classe, la tipica rappresentazione del modulo radice, avrà una struttura

di questo tipo:

```
import { NgModule } from
'@angular/core';

import { BrowserModule } from
'@angular/platform-browser';

import { MiaAppComponent } from
'./app.component';

import { MenuComponent } from
'./menu/menu.component';
```

@NgModule({

imports: [BrowserModule],

declarations: [MiaAppComponent, MenuComponent],

bootstrap: [MiaAppComponent]

})

export class **AppModule** {}

app.module.ts

Assimilerai l"uso e il significato di

ognuna, durante i vari esempi che vedremo

durante il percorso di creazione dell"applicazione MetroChat. Ecco una rapida

spiegazione:

24

 Il modulo , è chiamato per convenzione “AppModule”, ed è salvato con il

nome app.module.ts all'interno della cartella *app*.

 Ci deve essere sempre un solo “modulo radice”, per applicazione.

 Il modulo radice, deve sempre

importare “BrowserModule”, che conterrà una

serie di elementi importanti (direttive) come *ngIf, *ngFor. Vedremo che

esistono altri moduli importanti come “FormsModule” e “HttpModule”

 All'interno della proprietà declarations, inserirai un array con i nomi di

tutti i **componenti** usati nell'applicazione.

 All'interno della proprietà bootstrap, inserirai il nome del componente che

dovrà essere **caricato per primo.**

Sintassi del file *main.ts*

Nell'analogia con il razzo vista in precedenza, per far partire l'applicazione Angular,

abbiamo bisogno di una “miccia”, rappresentata dal file main.ts, anch"esso inserito

durante il processo di creazione di un progetto con la linea di comando. Si trova

all"interno della cartella *src*, insieme al file index.html.

Questo è il secondo file che ci serve. La tipica struttura sarà di questo tipo:

```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';

platformBrowserDynamic().bootstrapM
```

Non mi soffermo più di tanto sul la struttura, perché nella quasi totalità delle app che

progetterai, non dovrai mai modificarla. L'unica cosa da osservare è la presenza della

riga con i l metodo bootstrapModule() che, come avrai capito, ha il compito di

richiamare proprio il modulo creato in precedenza.

Il motivo per cui si è scelto di creare un file adibito al caricamento del Modulo

principale dell"app, è legato al fatto che Angular è un framework che permette lo

sviluppo di applicazioni in diversi ambienti, quindi non solo sul browser. Cambiando

la riga di lancio dell"applicazione, facilmente la potrai eseguire anche su ambienti

diversi.

25

Capitolo 3

I componenti in Angular

3.1 Il “cuore” di un’applicazione Angular

Come ricordato più volte, i l **“cuore”** di ogni applicazione Angular, è sicuramente

costituito dai ***componenti***. In questo capitolo cercheremo di capire come si creano ,

che caratteristiche hanno

e come si inseriscono all"interno di una

“vista”. E"

sufficiente infatti paragonare un"applicazione Angular alla squadra di calcio della tua

città o alla classe di una scuola.

Ogni giocatore, ogni alunno della classe, non è altro che un

componente della

squadra/classe, dotato di un proprio nome, della propria maglia e delle sue

peculiarità.

Se provi a guardare l'immagine

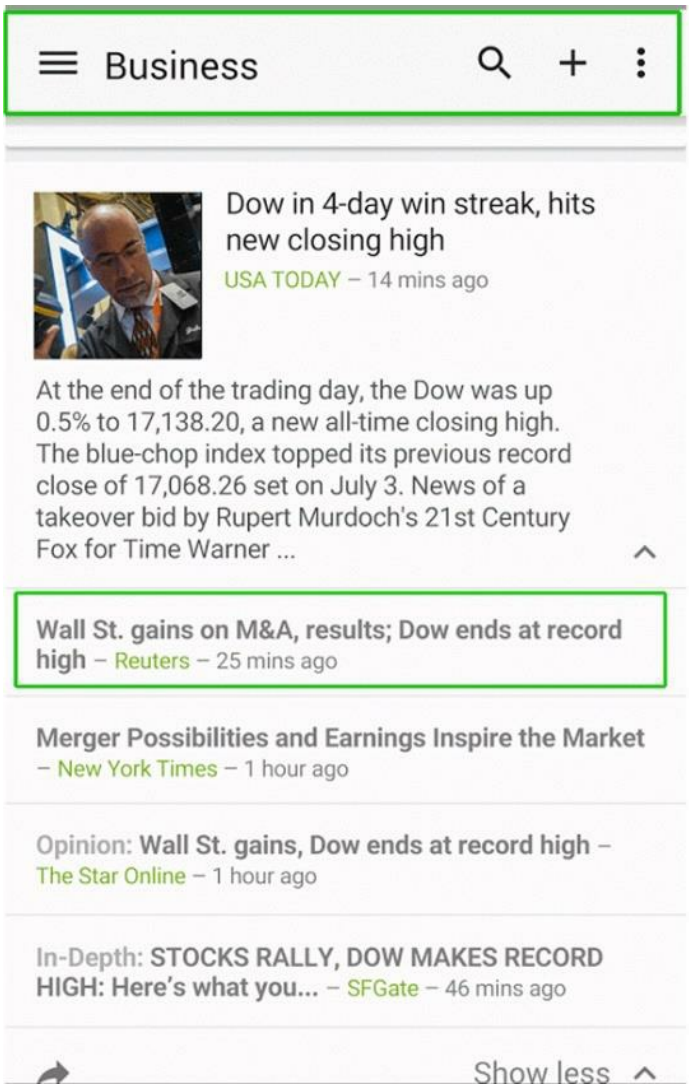
dell'interfa

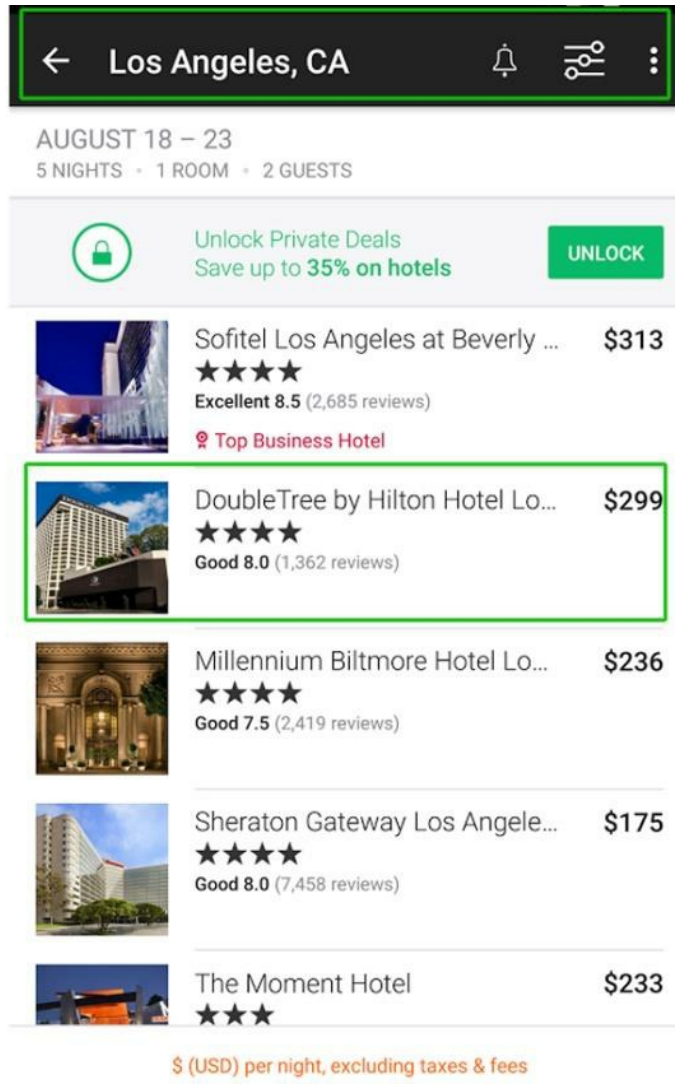
ccia di semplici APP come quelle

di

fig.3.1, ti convincerai che anch'ess a può essere pensata costituita da tanti elementi,

ognuno dei quali ha uno specifico comportamento.





26

fig. 3.1

C'è il menu dell'intera applicazione e ci sono i singoli articoli. Ebbene in Angular, è

possibile isolare ciascun elemento di un'APP, e trasformarlo

appunto in un

Componente. Quest"ultimo quindi non è altro che un singolo elemento

dell'interfaccia utente (UI) , dotato di caratteristiche univoche e che è possibile

collegare ad altri elementi dell'interfaccia.

Un componente inoltre, può avere all'interno altri componenti, detti componenti

"figlio" (Child Component) e con cui il componente “genitore” o “padre” è in grado

di comunicare.

Come già visto nel capitolo precedente,

un com ponente deve avere allora **tre**

caratteristiche fondamentali:

1. Deve avere un **nome identificativo (nome o selettore)**

2. Deve avere una **rappresentazione grafica (template o vista)**

3. Deve avere delle **funzionalità (classe**

o modello)

Applicando questi concetti alla prima applicazione di fig.3.1, potremmo dire che il

menu é:

1. identificato con il nome "ca-menu"

2. ha una rappresentazione grafica con del codice html che ne definisce la

struttura e il testo da visualizzare

3. ha delle funzionalità tali da poter cambiare pagina no n appena il

navigatore

tocca un link.

A livello tecnico, vedremo che un componente sarà rappresentato da un file con

estensione .ts, quindi sarà a tutti gli effetti una **classe**.

3.2 Il componente <app-root> creato in

automatico

Chi di voi ha già provato a creare una prima applicazione Angular con la linea di

comando, avrà notato che, all'interno della cartella *app*, vengono aggiunti una serie

di file, compreso quello con nome app.component.ts.

Ebbene questo file è proprio il primo componente visualizzato da Angular, perché è

quello indicato all"interno del file

app.module.ts, in corrispondenza alla

proprietà bootstrap.

Il file con nome

app.component.ts, definisce il componente con selettore

<app-root> e impareremo a "decifrare" le righe presenti all'intern

o, non appena

avremo capito come creare un nuovo componente.

3.3 Come creare il componente Menu dell'app

MetroChat

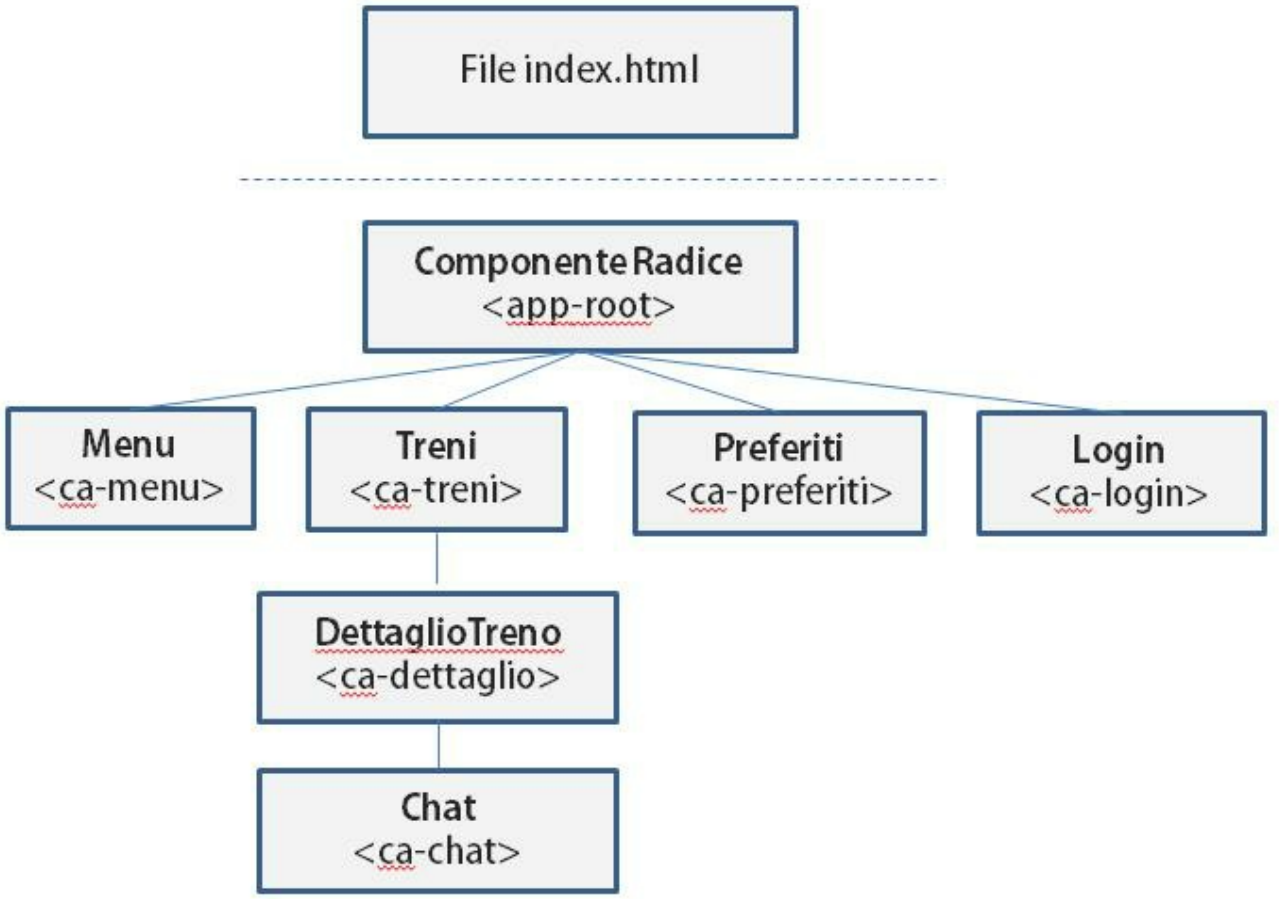
Come ogni sito web che si rispetti, costituito da più pagine, anche nel caso di un'app,

non dovremo fermarci al componente predefinito <app-root> creato in automatico,

ma dovremo imparare a crearne di altri, proprio perch

é un'app complessa, sar à

costituita da diversi di questi, con una tipica struttura ad albero rovesciato.



In questo modo, con un percorso graduale, impareremo tutti i concett

i legati ai

componenti, dal nome da dare al

selettore, dove inserirlo, come aggiungere un

template, come definire la classe e che nome darle.

Il metodo più rapido per creare un nuovo componente, è sfruttare la linea di

comando. E' chiaro che dovrai prima d

i tutto creare un nuovo progetto, e poi

spostarti all'interno della cartella

app. Qui dentro andrai a creare i diversi

componenti di cui sarà costituita la tua applicazione, oltre che al file

app.module.ts già creato in precedenza.

Per creare un nuovo componente, sfruttando la linea di comando, dovrai scrivere:

ng g component NOME

dove al posto di NOME dovrai sostituire l"effettivo nome che vorrai assegnare al

componente.

NB: Ricordati di creare i tuoi componenti, all'interno della cartella *app* della tua

applicazione

Per il nome del tuo componente, lo stile consigliato, è di usare la notazione "Upper

Camel Case" ossia la prima lettera maiuscola di ogni porzione di parola. Nel caso

allora volessi creare il componente "menu" dell'ipotetica app

licazione di fig. 3.1,

potresti scrivere:

ng g component Menu

Dopo l"esecuzione del comando, apparirà una nuova cartella di nome *menu*, con

all"interno una serie di file, compreso quello con nome

menu.component.ts, che

rappresenta proprio il nostro componente.

Nota co me il nome sia scritto in

minuscolo ed è stata aggiunta la parola "component".

Questa è una convenzione cui dovrai abituarti, e che si applicherà anche ad

altri

elementi di un'app Angular.

Se provassi ad aprire il file, con sorpresa trovere sti all'interno una serie di righe già

create per te e che inizieremo ad analizzare nelle prossime pagine. All'apparenza

potrebbero sembrare complesse, ma fra meno di 20 minuti saprai “decifrarle” senza

alcuna difficoltà.

```
import { Component, OnInit } from
```

```
'@angular/core';

@Component({

selector: 'app-menu',

templateUrl: './menu.component.html',

styleUrls: ['./menu.component.css']

})

export class MenuComponent
implements OnInit {

constructor() { }

ngOnInit() {
```

```
}

}
```

menu/menu.component.ts

Se il progetto è stato creato sfruttando l"opzione –*minimal*, troverai un codice

leggermente diverso, ma capiremo le differenze nei prossimi capitoli.

3.4 Tecniche alternative per creare un

componente

Una possibile alternativa per la creazione di un componente è copiare un

componente già rea

lizzato, ad esempio proprio quello associato al selettore

<app-root> e incollarlo nella cartella *app* o in una sottocartella.

Chiaramente dovrai poi andare a modificare i nomi interni, ma soprattutto dovrai

andare a modificare il file

app.module.ts, perché è necessario aggiungere all'array

abbinato alla proprietà declarations del decoratore @NgModule(), il nome della

classe associata a questo nuovo componente, oltre che importare la classe stessa, con

l'istruzione import.

Questa operazione è chiamata "**registrazione**" del componente nel Modulo.

Nel 99% dei casi sono sicuro che ti dimenticherai di farla, e questo è il motivo per

cui è preferibile usare la tecnica precedente, che al contrario, e

segue queste

operazioni in automatico.

3.5 Definire il nome del componente grazie al

"selettore"

Abbiamo detto che per **identificare** il componente, dobbiamo assegnargli un **nome**

univoco che potrà essere usato all'interno del codice html della pagina principale o di

altri componenti, per richiamare il componente stesso, sotto

forma di tag html

personalizzato (es. <ca-menu></ca-menu>)

Abbiamo detto che un componente non è altro che una classe TypeScript, quindi

come sempre avremo un costruttore e del codice all'interno. La grossa differenza

rispetto alla classica classe TypeScr ipt, è che dovremo dare ulteriori indicazioni ad

Angular, per informarlo che si tratta di un componente.

Questo si fa aggiungendo alla classe una particolare riga, che precede il costruttore, e

che viene chiamata "Decoratore", un nome che richiama alla mente un'arte antica che

ormai sta scomparendo.

Analizzando il codice qui sotto, di cui ho eliminato alcuni pezzi:

```
import { Component } from
'@angular/core';

@Component({

selector: 'ca-menu', // Qui definisco il
nome!

...

})

export class MenuComponent {

constructor() { }

}
```

menu/menu.component.ts

notiamo - subito prima del costruttore - la presenza del termine @Component().

Ebbene questo termine è specifico del linguaggio TypeScript e

assume il nome

inglese di " **decorator**" o decoratore, e come detto, è un element

o che pe rmette di

assegnare alla classe, specifiche **caratteristiche o metadati** , un p o" come avviene

per il tag <head>, interno alle pagine html, in cui la presenza dei vari metatag,

permette di specificare proprietà della pagina web, come il titolo, la descrizione etc.

Queste caratteristiche sono espresse sotto forma di un oggetto JavaScript, quindi

sono racchiuse all'interno delle classiche parentesi graffe.

NB: Affinché Angular sia in grado di capire cosa sia @Component() è

necessario importare la definizione dalla libreria @angular/core. Non

preoccuparti più di tanto per ora, perché queste righe sono aggiunte in

automatico all'atto della creazione del componente con la linea di comando.

La sintassi con cui p uoi definire il nome del **selettore del component e**, da usare

all'interno di codice html, è la seguente:

<NOMECOMPONENTE>
</NOMECOMPONENTE>

è necessario inserire la prima proprietà selector:

@Component({

// identifico il nome del selettore

selector: 'ca-menu'

})

In questo caso, il nome scelto per il selettore del componente è

ca-menu, ma sono

libero di chiamarlo come voglio, rispettando però le regole di stile indicate in

precedenza.

NB: Il nome del selettore non deve avere all'interno i simboli < e > dei tag

HTML

Fai attenzione alla sintassi: nomeproprietà: 'valore', perché sarà usata anche

per altre proprietà.

3.6 Analisi del componente <app-root>

Alla luce di quanto appreso, siamo in grado ora di capire a cosa fa riferimento

l'elemento html <app-root> che abbiamo trovato nel file *index.html* non appena è

stata creata la prima applicazione Angular.

```
<!DOCTYPE html>

<head>

<title>Angular App</title>

<meta charset="UTF-8">

<meta name="viewport"
content="width=device-width, initial-
scale=1">

</head>

<body>

<!-- il componente radice, padre di tutti
i componenti -->
```

<app-root></app-root>

</body>

</html>

index.html

Abbiamo imparato che questo tag identifica un componente, quindi dovremo

verificare tra la lista di tutti i file che hanno

un no me con questa sintassi

nome.component.ts, se ne esiste uno che fa riferimento a questo tag.

E' facile individuarlo, in quanto per ora la nostra app è costituita da un solo

componente, quindi aprendo il file app.component.ts scopriremo dove è stato

definito questo nome:

```
import { Component } from '@angular/core';

@Component({

// 1) qui è stato definisco il nome del componente

selector: 'app-root',
```

templateUrl: './app.component.html',

(continua)

---

styleUrls: ['./app.component.css']

})

export class AppComponent {

title = 'app!';

}

app.component.ts

NB: Il nome impostato per il selettore del componente, tipicamente non deve

coincidere con il nome della classe, anche se conviene sceglierlo in modo che

sia facile da ricordare, in funzione dell'uso che se ne farà all'interno

dell'applicazione.

NB: Il nome del selettore, devi includerlo all'interno di singole virgolette.

La domanda che potresti farti adesso è: come mai, all'apertura dell'app nel browser, è

visualizzata la scritta “Welcome to app!” se nel file index.html non è presente?

Questo lo capiremo con l'introduzione del concetto di “Template” o “Vista”.

3.7 Definire la rappresentazione grafica:

Template

Vediamo di rispondere alla domanda preceden

te, partendo dal componente

<ca-menu> creato in precedenza, che ci permetterà di entrare nel vivo dello

sviluppo della nostra applicazione.

La domanda che devi farti è: come posso rappresenta re graficamente il componente

rappresentativo del menu della mia applicazione? Beh, se stessi creando una

classica

pagina HTML, partirei con definire un foglio di stile associato e del codice html.

Preoccupandoci per ora solo del codice html, potresti scrivere:

```
<!-- menu applicazione -->

<nav>

<ul>

<li><a href="">Link Menu 1</a></li>

<li><a href="">Link Menu 2</a></li>
```

Link Menu 3

</nav>

34

Ora sapendo che con il decoratore @Component() posso aggiungere una serie di

proprietà alla classe (

metadati), i progettisti di Angul

ar hanno predispost

o

un'ulteriore proprietà dal nome template, o in alternativa templateUrl,

finalizzata all"inclusione di codice html.

La differenza tra le due è che la prima, ci permette di aggiungere il codice html

,

direttamente nel file della classe d el componente , la seconda invece, sfruttando il

riferimento a un file con estensione *.html* che andrai ad inserire nel progetto.

Nel primo caso quindi dovr

ai scrivere - sempre nell'ipotesi del componente

<ca-menu> :

import { Component } from '@angular/core';

@Component({

// 1) Definisco il nome del selettore

selector: 'ca-menu',

// 2) Definisco il template grafico "inline"

```
template: `

<!-- menu applicazione -->

<nav>

<ul>

<li><a href="">Link Menu 1</a></li>

<li><a href="">Link Menu 1</a></li>

<li><a href="">Link Menu 1</a></li>

</ul>

</nav>

`
```

```
})

export class MenuComponent {

constructor() { }

}
```

menu/menu.component.ts

NB: Osserva come la stringa rappresentativa del codice html della pagina, sia

stata inserita tra i due simboli detti "backtick" che sono diversi dal classico

simbolo di singola virgoletta. Per poterli inserire usando la tastiera italiana,

dovrai digitare la combinazione di tasti: ALT e contemporaneamente digitare il

numero 96 (ALT+96)

Sfruttando la proprie

tà template, sicuramente si ha il vantaggio di poter

visualizzare il codice html del componente e di poter apportare modifiche alla logica

che inserirai nel corpo della classe, senza dovere aprire un altro file.

Per applicazioni più complesse, che hanno mo

lto codice html per singolo

componente, di norma si lavora con un file esterno, dove si andrà ad inserire il

codice html rappresentativo del componente, nel nostro caso del menu.

Dovrai però sfruttare la proprietà templateUrl, come visualizzato qui sotto:

```
import { Component } from '@angular/core';

@Component({
```

```
// 1) Definisco il nome del selettore
selector: 'ca-menu',
// 2) Definisco il template grafico
templateUrl: './menu.component.html'
})
export class MenuComponent {
constructor() { }
}
```

menu/menu.component.ts

dove *menu.component.html*, conterrà

proprio il codice per rappresentare

graficamente nel browser il menu, come indicato qui sotto:

```
<!-- menu applicazione -->

<nav>

<ul>

<li><a href="">Link Menu 1</a></li>

<li><a href="">Link Menu 2</a></li>

<li><a href="">Link Menu 3</a></li>

</ul>
```

</nav>

NB: Il percorso a cui fa riferimento il file menu.component.html, è relativo alla

posizione dello stessa classe. E' prassi infatti tenere i file rappresentativi di un

componente, raggruppati all'interno della stessa cartella.

Pertanto quello che farà Angular non appena incontrerà un selettore collegat o ad un

componente, sarà di sostituire al tag, il relativo codice html definito nella proprietà

template o templateUrl.

36

3.8 Come aggiungere un foglio di stile

Altra proprietà che puoi sfruttare per aggiungere delle regole CSS da applicare solo

alla sezione rappresentata dal componente, è costituita da

styleUrls. Anche in

questo caso dovrai definire un file esterno, che ospiterà tutte le regole CSS da usa re

per il componente.

Questa proprietà in realtà puoi anche ometterla per i componenti interni, perch

é

spesso si userà un foglio di stile centralizzato, per evitare di dover spezzettare le

regole in decine di file difficili da tenere aggiornati. Questo nel

l'ipotesi il foglio di

stile non sia particolarmente "pesante" dal punto di vista dei kB da scaricare.

3.9 Il template del componente <app-root>

Ora che sappiamo come si definisce il template del componente, ossia la propria

rappresentazione a livello di codice HTML, vediamo come sia stato progettata la

grafica del componente principale dell'applicazione.

Se proviamo ad aprire il file

app.component.ts, troveremo la proprietà

templateUrl valorizzata con
'./app.component.html'.

```
import { Component } from
'@angular/core';

@Component({

selector: 'app-root',

templateUrl: './app.component.html' ,

styleUrls: ['./app.component.css']

})

export class AppComponent {

title = 'app!';
```

}

app.component.ts

All"interno di quest'ultimo file, troveremo, tra le diverse righe di codice HTML, il

seguente pezzo:

<h1> Welcome to {{title}} </h1>

37

E' chiaro allora il motivo per cui, non appena hai aperto nel browser la tua prima app

Angular, compariva una scritta con tag H1 come quella qui sotto:

Rimane da capire da dove viene prelev

ata la stringa " Welcome to app!" e capire la

strana notazione presente all'interno del codice html, costituita dai simboli {{}}, ma

penso tu abbia già intuito qualcosa.

Ora che l'applicazione ha ben due componenti, potrei inserire il

componente con

selettore <ca-menu> internamente al template del componente radice, in modo da

sperimentare la costruzione di un

"albero di componenti" , caratteristico di ogni app

Angular.

Le modifiche a livello di codice html, potrebbero essere di questo tipo:

<ca-menu><ca-menu>

<h1> MetroChat {{title}} </h1>

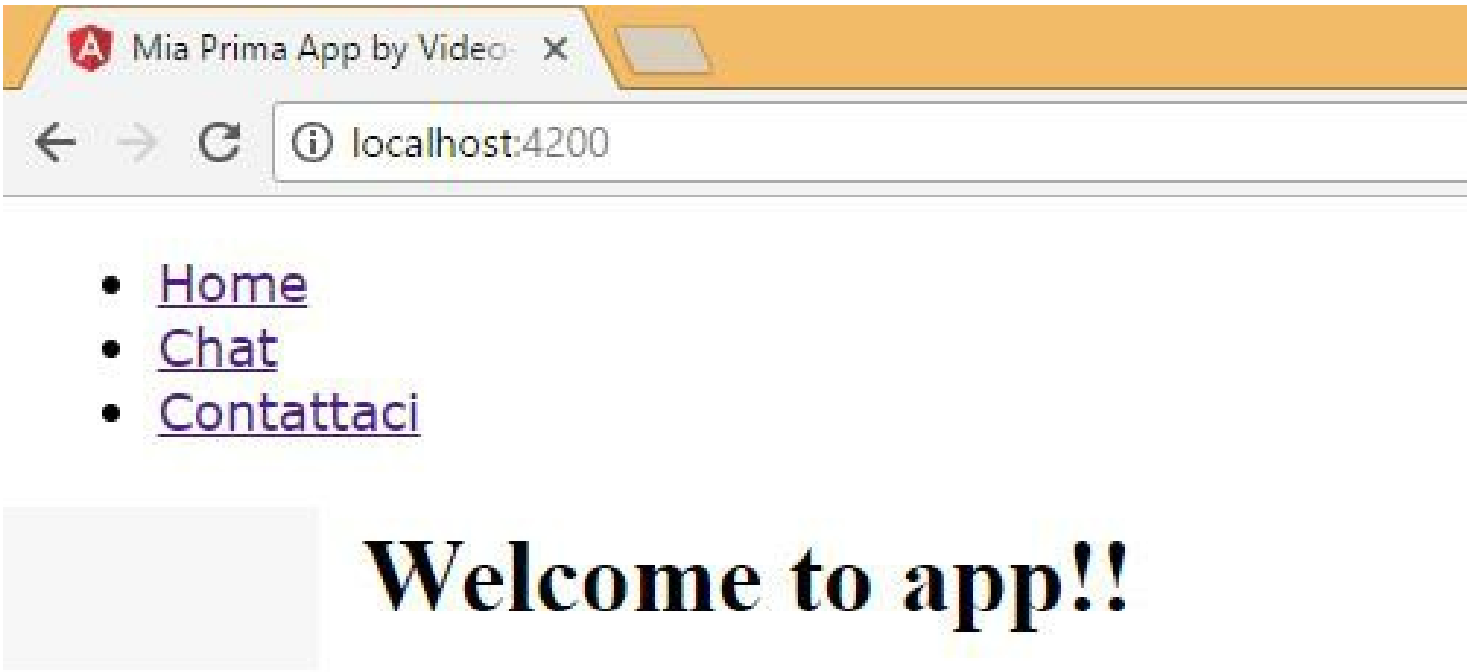
mentre a livello di classe del componente non cambierà nulla:

import { Component } from '@angular/core';

@Component({

```
selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})
```

(continua)



```
export class AppComponent {

title = 'app!!';

}
```

app.component.ts

Quello che ottengo aprendo l"applicazione nel browser, sarà proprio la lista dei menu

appena creata, con sotto la scritta “Welcome to app!!”.

Siamo riusciti a incastrare il nostro primo componente MenuComponent,

all"interno

di un altro componente, in particolare all"interno del componente padre con selettore

<app-root>

A questo punto l a domanda che potresti farti è: come fa Angul ar a rappresentare il

componente *menu*, se nella classe del componente principale, non vi è alcuna

indicazione di questo tipo?

Ebbene, se ti ricordi, ogni qualvolta

si crea un nuovo componente, questo viene

aggiunto all'interno del file app.module.ts, propr io all'interno dell'array della

proprietà declarations del decoratore @NgModule().

Se tu provassi ad eliminare questo elemento commentandolo

, come indicato qui

sotto:

```
import { NgModule } from
'@angular/core';

import { BrowserModule } from
'@angular/platform-browser';

import { MiaAppComponent } from
'./app.component';

//import { MenuComponent } from
'./menu/menu.component';

@NgModule({

imports: [ BrowserModule ],

// declarations: [ MiaAppComponent,
MenuComponent ],
```

declarations: [MiaAppComponent],

bootstrap: [MiaAppComponent]

})

export class AppModule {}

app.module.ts

verrebbe generato un errore, proprio perch é Angular non sarebbe in grado di capire

come rappresentare il componente <ca-menu>.

3.10 Definire la logica del componente

Arrivati a questo punto, l'ultima cosa che ci rimane da vedere è come definire il

cuore del componente, ossia le sue **funzionalità**.

Un componente, nella sua forma più banale, potrebbe solo mostrare del codice html,

corrispondente a quello inserito nel template, ma nelle magg

ior parte delle

applicazioni, dovrà anche interagire con le azioni fatte dall'utente o mostrare dei dati

provenienti da un database remoto.

Ecco allora che devo poter capire come creare una relazione tra gli elementi definiti

nel template ed eventuali azioni da far fare.

Queste funzionalità dovranno essere progettate sfruttando proprio TypeScript come

linguaggio, quindi per grossa parte del codice il classico JavaScript.

Sapendo che ogni componente è una classe, la logica dovrà essere inserita all'interno

del corpo della classe, che chiaramente dovrà avere un nome, corrispondente proprio

a quello scelto in fase di creazione del componente con la linea di comando, (es.

MenuComponent)

Il risultato finale di tutto il nostro sforzo per definire i tre elementi pr

incipali del

componente

con selettore

<ca-menu>, sarà rappresentato qui sotto:

```
import { Component } from
'@angular/core';

@Component({

selector: 'ca-menu',

templateUrl: './menu.component.html',

styleUrls: ['./menu.component.css']

})

export class MenuComponent {
```

// qui dentro definisco la logica del componente

// ed eventuali operazioni di inizializzazione dati

constructor()

}

menu/menu.component.ts

Nota la presenza della parola chiave export che precede la parola class. Questa

serve nel caso abbia la necessit à di importare la classe anche in altre

sezioni della

mia applicazione.

Il passo successivo sarà quello di iniziare a sperimentare come si possa creare una

"comunicazione" tra le righe presenti all'interno della classe e il template, perch

é

spesso avrò la

necessità di mostrare dei dati provenienti da sorgenti esterne,

operazioni che dovrò chiaramente fare all'interno della classe.

3.11 Popolare il template con proprietà della

classe

Ora che siamo arrivati al cuore del componente

, iniziamo a divertirci un p

o",

vedendo cosa sia possibile fare.

Partiamo con un esempio banale, per cercare di capire come avviene i l meccanismo

di comunicazione tra template e corpo della classe detto anche " **Data Binding**".

Se ad esempio, volessi creare un menu,

con il testo dei lin

k programmato

direttamente nel codice della classe, come potresti fare?

Beh, noi sappiamo che in JavaScript, si possono definire delle variabili, quindi il

primo passo potrebbe essere quello di definire tre **variabili interne alla classe**.

Nella terminologia della programmazione ad oggetti, queste ultime vengono

chiamate anche proprietà o membri.

41

```
import { Component } from
'@angular/core';

@Component({

selector: 'ca-menu',

templateUrl: './menu.component.html',

styleUrls: ['./menu.component.css']

})
```

```
export class MenuComponent {

// definisco 3 proprietà del componente

link_menu_1;

link_menu_2;

link_menu_3;

constructor(){

this.link_menu_1 = 'Treni';

this.link_menu_2 = 'Preferiti';

this.link_menu_3 = 'Login';

}
```

}

menu/menu.component.ts

TypeScript ci permette di associare ad ogni variabile anche la tipologia di dato che

andrà a "contenere", nel nostro caso una stringa. Pertanto abituiamoci fin da ora a

specificarlo con la notazione

nomevariabile: TIPO, per le tre proprietà

link_menu_N scelte:

import { Component } from

'@angular/core';

@Component({

selector: 'ca-menu',

templateUrl: './menu.component.html',

styleUrls: ['./menu.component.css']

})

export class MenuComponent {

// definisco 3 proprietà del componente

link_menu_1:string;

link_menu_2:string;

link_menu_3:string;

constructor(){

this.link_menu_1 = 'Treni';

this.link_menu_2 = 'Preferiti';

this.link_menu_3 = 'Login';

}

}

menu/menu.component.ts

NB: E' buona norma abituarsi a **inizializzare** le proprietà direttamente nel

costruttore, perché è il primo pezzo di codice eseguito all’atto del caricamento

del componente (vedi ciclo di vita di un’app)

3.12 Definire i "segnaposto" nel template con

l'interpolazione

Come fare ora per visualizzare il contenuto di queste tre variabili all'interno d

el

template? Se provassi a inserire il solo nome di queste, verrebbe visualizzata una

stringa corrispondente al nome della variabile e non al suo contenuto.

E' necessario allora usare la tecnica

chiamata **“interpolazione”** , una delle tante

tecniche che vedremo per manipolare il contenuto del template.

In sostanza si tratta d "inserire nel template, il nome della variabile definita nella

classe, "contornata" da due parentesi graffe in apertura e chiusura: una sorta di

“segnaposto”:

{{NOMEVARIABILE}}

Ogni qua lvolta Angular trova questa notazione nel template , si ricorda che dovrà

andare a sostituire il corrispondente valore prelevato dalla rispettiva variabile

definita nel corpo della classe del componente.

Solo le variabil i di quel componente possono essere v

isualizzate nel relativo

template. Non possono visualizzare variabili definite in altri componenti figli o

padri, a meno che non applichi ulteriori tecniche che vedremo quando parleremo di

progettazione di componenti intelligenti.

Il template del componente menu (menu.component.html), diventerebbe così:

```
<!-- menu applicazione -->

<nav>

<ul>

<li><a href="">{{link_menu_1}}</a>
</li>

<li><a href="">{{link_menu_2}}</a>
</li>

<li><a href="">{{link_menu_3}}</a>
</li>

</ul>

</nav>
```

Le due parent esi di apertura e chiusura, devono essere inserite SENZA spazi tra le

due parentesi, mentre il nome della variabile interna può essere inserito con spazi

Qui sotto alcuni esempi di cosa è possibile e non è possibile fare:

<!-- menu applicazione -->

<nav>


```
<li><a href="">{ {link_menu_1}}</a>
</li> <!-- NO -->

<li><a href="">{{ link_menu_2}}</a>
</li> <!-- Sì -->

<li><a href=""> {{ link_menu_3 }}
</a></li> <!-- Sì -->

</ul>

</nav>
```

Il risultato finale, non appena il componente verrà visualizzato nel browser, sarà che

al posto dei tre "segnaposto", saranno so stituiti i valori inseriti nelle tre

variabili,

ossia rispettivamente le stringhe “Treni”, “Preferiti” e “Login”.

3.13 Inserire espressioni all'interno del

"segnaposto"

All'interno del segnaposto evidenziato dalle doppie parentesi graffe, non ci si limita a

inserire variabili di tipo *stringa*, ma è possibile inserire un elemento di un *Array*, oppure un'espressione, come nell'esempio qui sotto.

<button>Pulsante LUCE : {{ isActive ? "ON" : "OFF" }}</button>

All'interno delle parentesi graffe, ho inserito la classica istruzione if then else

(operatore ternario), per aggiungere alla stringa di testo "Pulsante LUCE:" il valore

"ON" o "OFF" sulla base del valore della variabile isActive , membro della classe .

Angular pertanto è in grado di valutare un'espressione semplice inserita interna alle

doppie parentesi graffe. Per capire come

collegare un evento alla pressione sul

buttone, dovremo aspettare il tutorial sugli gestione degli eventi.

Avrai notato che, nel codice del template precedente, sono ripetuti tre tag li. Per un

menu, di cui sappiamo il numero massimo di voci da visualizzare, è una situazione

abbastanza normale, ma quando a

bbiamo la necessità di visualizzare dei dati

provenienti da fonti esterne, quali

database o file JSON, la situazione cambia.

Infatti, non conosceremo a pr iori il numero di dati restituiti, e come conseguenza,

non riusci remo a impostare un layout con N tag ripetuti , se il valore di N appunto

non è noto.

Per questo motivo dobbiamo conoscere altre tecniche in grado di aiutarci nella

manipolazione del DOM.

3.14 Ciclo di vita di un'app Angular

Il caricamento di un'app Angular, presuppone il caricamento di N componenti,

ognuno dei quali ha precisi "stati" durante il **ciclo di vita dell'intera app**.

Per capire il concetto di “stato”, possiamo ricorrere all"analogia delle fermate di un

autobus, da un capolinea all'altro. Durante il percorso, è possibile salire sull'autobus,

in corrispondenza alle diverse fermate.

Nel caso di un'app Angular,

queste “fermate” sono

gestite in automatico e ci

permettono di entrare o agganciarci a diversi stati d"esecuzione di un'app.

Sappiamo infatti, dai capitoli precedenti, che il cuore di ogni componente è costituito

dal corpo della classe. E all"interno di ogni classe TypeScript, ci sarà sempre il

costruttore, dove per convenzione ed efficienza, si inizializzano le eventuali

proprietà del componente.

Il costruttore , quindi, è proprio il primo elemento che viene richiamato all'atto de

l

caricamento di un componente.

Questo è il motivo per cui, nel caso i n cui il componente dipenda da altri elementi, si

sfrutta il costruttore per "iniettare" le dipendenze.

Ad esempio, quando un componente o service ha la necessità di comunicare con la

rete, dovremo iniettare nel costruttore un service di nome http, grazie al quale

potremo accedere ad una serie di metodi della libreria

HttpClient, utili per

dialogare con la rete.

Ecco allora la lista degli eventi principali, che nel corso dello sviluppo di

un'app,

potrai sfruttare per effettuare degli interventi sui dati o

per agire sulla logica di

funzionamento dell"applicazione.

E' chiaro che per poterli utilizzare, dovrai implementarli nel costruttore del

componente, e includerli richiamandoli dalla libreria @angular/core.

ngOnInit

Il metodo ngOnInit è chiamato sempre **una sola volta dopo l'esecuzione del**

costruttore e subito dopo la prima chiamata al metodo ngOnChanges. Può essere

sfruttato per inizializzare alcune proprietà intern

e al componente o

richiamare

metodi di un *service*, per operazioni di scambio dati con la rete. All'interno del corpo

della classe del componente, dovrai usare la notazione:

class MioComponente **implement OnInit** {

constructor() {}

ngOnInit() {

// Definisco qui le azioni da fare

}

}

NB: Tale metodo viene sempre chiamato una sola volta.

ngOnChanges

Quando un componente riceve dei dati di ingresso che cambiano nel tempo e sulla

base dei quali devi eseguire determinate azioni, ci viene in aiuto il metodo

ngOnChanges, per segnalare tutte le volte in cui il valore subisce una modifica.

Il metodo riceve un parametro in ingresso, che è un oggetto di tipo SimpleChange,

il quale descrive nel dettaglio le

variazioni di ogni proprietà di input, grazie a tre

proprietà (isFirstChange, previousValue, currentValue).

Un possibile oggetto collegato alla proprietà di ingresso "nome", potrebbe essere:

{"nome":
{"previousValue":"","currentValue":"Da

La proprietà d "ingresso cambia dalla stringa vuota "", al valore "

Davide". Per

intercettare tale cambiamento, potrei

scrivere:

```
class MioComponente implements
OnChanges {

constructor() {}

ngOnChanges(changes: SimpleChange})
{

if(changes['nome'].isFirstChange()) {

console.log('Primo cambio di valore');

}

// Mostro l'attuale valore e quello
precedente di nome
```

```
console.log('Valore attuale: ' +
changes['nome'].currentValue);

console.log('Valore prec. ' +
changes['nome'].previousValue);

}

}
```

ngDoCheck

ngDoCheck viene attivato ogni volta che le proprietà d"ingresso di un componente o

una direttiva vengono controllati. È possibile utilizzare questo aggancio

al ciclo di

vita per estendere il controllo con la tua logica personalizzata.

ngAfterContentInit

Chiamato dopo ngOnInit quando il contenuto di un componente o una direttiva è

stato inizializzato.

ngAfterContentChecke

Chiamato dopo ogni verifica del contenuto del componente o di una direttiva.

ngAfterViewInit

Chiamato dopo ngAfterContentInit quando la vista del componente è stata

inizializzata. Si applica solo a componenti.

ngAfterViewChecked

Chiamato dopo ogni verifica della vista di un componente. Si applica solo

ai

componenti.

ngOnDestroy

L'ultimo metodo che viene chiamato nel ciclo di vita di un componente, appena

prima che l'istanza del componente sia finalmente distrutta.

Capitolo 4

Manipolare il DOM con le direttive

4.1 Ripetere elementi del DOM: la direttiva

*ngFor

Angular offre una serie di strumenti per manipolare il DOM e

potenziare i tag

dell'HTML, di funzionalità non present i nel linguaggio nativo. A questo scopo, i n

Angular, sono state introdotte le

“**Direttive**”. Non ti devi spaventare dal nome,

perché si tratta semplicemente di un insieme di notazioni, da incastrare all'interno di

un normale tag HTML e che permettono

di arricchirlo di nuove funzionalità.

Le direttive si distinguono tra quelle in grado di cambiare la struttura del DOM della

pagina (direttive “strutturali”), e quelle che in grado di cambiare solo l'aspetto di tag

già presenti (direttive “attributo”), come ad esempio il colore di un testo.

Non appena si utilizzano le direttive, si inizia a creare quel collegamento tra il

template e il **corpo della classe** , che si manifesterà in tutta la sua potenza non

appena parleremo dei form.

Si parla in gergo tecnico di “

Template binding ” intendendo il collegamento di

proprietà di una classe, con proprietà presenti nel template del componente.

Non devi confondere la proprietà di un elemento del DOM con gli attributi di un tag

HTML, che sono quelli con cui generalmente siamo abit

uati a lavorare quando si

sviluppano le classiche pagine web . E" una sottigliezza che andrebbe capita a fondo,

ma per non deviare troppo dal nostro obiettivo, memorizza questo concetto:

“Il Template binding lavora con le proprietà di un elemento del DOM, d i un

componente, o direttiva e NON con gli attributi.”

Tornando a noi, una delle direttive più usate in Angular, è quella che ti permetterà di

modificare il DOM ripetendo un certo

numero di volte uno specifico tag HTML o

componente. Sto parlando della direttiva *ngFor.

E' frequente infatti creare delle applicazioni che visualizzino un insieme ripetuto di

articoli, dotati di immagine, titolo, descrizione. L"insieme di questi articoli, potrebbe

essere inserito all'interno di un tag <article>, oppure all'inter no di un semplice

, come nella rappresentazione qui

sotto:

```
<div class="news">

<ul>

<li><h2>Titolo Notizia 1 </h2>
<p>Desc. notizia 1</p></li>

<li><h2>Titolo Notizia 2 </h2>
<p>Desc. notizia 2</p></li>

<li><h2>Titolo Notizia 3 </h2>
<p>Desc. notizia 3</p></li>

<li><h2>Titolo Notizia 4 </h2>
<p>Desc. notizia 4</p></li>

</ul>
```

che sarà visualizzato nel browser in questo modo:

In un'applicazione reale, non saprai a priori quanti e quali articoli dovrai aggiungere,

perché questi saranno tipicamente prelevati da una sorgente esterna all'applicazione

(es. database o servizio esterno).

Pertanto non potrai inserire i vari segnaposto, come visto nel tutorial precedente.

Entra in azione allora la direttiva *ngFor.

49

Nell'ipotesi di aver progettato il componente

con nome <ca-listanews>, nel

seguente modo:

```
import { Component } from '@angular/core';

@Component({

selector: 'ca-listanews',
```

```
template: `

<div class="news">

<ul>

<li><h2>Titolo Notizia 1 </h2></li>

<li><h2>Titolo Notizia 2 </h2></li>

<li><h2>Titolo Notizia 3 </h2></li>

</ul>

</div> `

})

export class NotizieComponent {
```

}

notizie/notizie.component.ts

la domanda che potresti farti è: come posso iniettare delle notizie nel template, senza

conoscere a priori il numero?

4.2 Sintassi della direttiva *ngFor

Grazie a questa direttiva, io riesco a ripetere determinati elementi del DOM un certo

numero di volte. La sintassi base è la seguente:

<nometag *ngFor="let variabile of listadati">

...

</nometag>

Il simbolo * deve essere inserito attaccato a ngFor. Fai poi attenzione a rispettare le

minuscole e maiuscole. Questa è la sintassi base, ma altre caratteristiche le puoi

consultare collegandoti alla pagina: https://angular.io/api/common/NgForOf

A destra dell'uguale, all'interno delle doppie virgolette, si inseri

sce un"espressione

che stabilirà il numero totale di ripetizioni:

50

X è il nome di una variabile locale

da usare nel template, mentre Y è un array,

definito nel corpo della classe, che conterrà l"insieme dei valori con cui valorizzare

la variabile X ad ogni ripetizione

. Se hai già esperienza

con altri linguaggi di

programmazione, la notazione ricorda quella che si usa per i classici

“cicli for ”, da

cui è stato copiato il nome.

Spesso avrai a che fare con a rray di stringhe o numeri ma avremo modo di vedere

anche come manipolare array di oggetti.

Quello che viene ripetuto nel DOM, è il ta

g a cui è abbinata la direttiva

(comprensivo di eventuali figli) , per un

numero di volte pari al numero di elementi

presenti nell'array.

4.3 Uso della direttiva *ngFor con un array

Ipotizzando di aver popolato un array , di nome listanews, con notizie prelevate in

qualche modo da una sorgente esterna e ipotizzando per ora di inserirle manualmente

nell'array in questo modo:

listanews= ['Titolo Notizia 1','Titolo

Notizia 2', 'Titolo

Notizia 3', 'Titolo Notizia 4'];

allora per visualizzare tutte le notizie , sempre nell'ipotesi che non conosca il numero

totale presente in listanews, potrei sfruttare la direttiva *ngFor applicata a <li>:

import { Component } from '@angular/core';

@Component({

selector: 'ca-listanews',

```
template: `

<div class="news">

<ul>

// elemento che si ripete N volte

<li *ngFor="let titolo of listanews">

<h2>{{titolo}}</h2>

</li>

</ul>

</div> `

})
```

```
export class NotizieComponent {

listanews = ['Titolo Notizia 1','Titolo
Notizia 2',

'Titolo Notizia 3', 'Titolo Notizia 4'];

}
```

notizie/notizie.component.ts

51

A seconda di quante notizie saranno memorizzate all'interno dell'array listanews,

la direttiva andrà a modificare il

DOM crea ndo tanti tag <li> con all'intero un

segnaposto {{titolo}}, valorizzato proprio con il valore del relativo elemento

dell'array che si sta percorrendo.

Non appena aggiung i un nuovo elemento all'array listanews, in automatico si

aggiornerà anche il template, senza che tu debba intervenire manualmente sul codice

html.

Se questo componente lo aggiungessi al

template del componente

radice

dell"applicazione, app.component.ts:

<ca-menu></ca-menu>

<h1>{{title}}</h1>

<ca-listanews></ca-listanews>

otterresti la visualizzazione di tutte le news sotto il menu

, come evidenziato qui

sotto:

4.4 Uso della direttiva *ngFor con un oggetto

JSON

Se al posto dell'array avessi un oggetto JSON

, cosa cambierebbe? Non ci sono

particolari avvertenze. Si tratt a solo di fare attenzione a come è strutturato l'oggetto

JSON che contiene i dati da scorrere.

Spesso i nfatti i dati recuperati da sorgenti esterne , sono proprio in formato JSON,

quindi avere già un'infarinatura di come

si dovranno gestire, ci può essere d'aiuto p er

capire al meglio come sfruttare la potenza di Angular.

Partiamo come sempre dalla rappresentazione dei dati, e poi vediamo come creare il

componente con il relativo template.

listanews = [{

titolo: 'Titolo Notizia 1',

descrizione: 'Descrizione Notizia 1'

},

```
{
titolo: 'Titolo Notizia 2',
descrizione: 'Descrizione Notizia 2'
},
{
titolo: 'Titolo Notizia 3',
descrizione: 'Descrizione Notizia 3'
},
{
titolo: 'Titolo Notizia 4',
```

descrizione: 'Descrizione Notizia 4'

}];

Come puoi osservare si tratta della rappresentazione di un array di oggetti

JavaScript, con chiavi titolo e descrizione. La direttiva *ngFor in questo caso,

scorrerà se mpre tra i diversi elementi dell'array, solo che ora la variabile locale al

template, sarà valorizzata con un oggetto JavaScript.

Trattandosi di un oggetto,

dovrò utilizzare la tipica **notazione del punto,** per

accedere alle diverse chiavi o proprietà dell'oggetto.

```
import { Component } from
'@angular/core';

@Component({
```

(continua)

```
selector: 'ca-listanews',

template: `
```

```
<div class="news">

<ul>

<li *ngFor="let notizia of listanews">

// notizia è un oggetto JS

<h2>{{notizia.titolo}}</h2>

<p>{{notizia.descrizione}}</p>

</li>

</ul>

</div> `

})
```

```
export class NotizieComponent {

listanews = [{

titolo: 'Titolo Notizia 1',

descrizione: 'Descrizione Notizia 1'

},

{

titolo: 'Titolo Notizia 2',

descrizione: 'Descrizione Notizia 2'

},

{
```

```
titolo: 'Titolo Notizia 3',
descrizione: 'Descrizione Notizia 3'
},
{
titolo: 'Titolo Notizia 4',
descrizione: 'Descrizione Notizia 4'
}];
}
```

notizie/notizie.component.ts

Abbiamo pertanto raggiunto lo stesso

risultato

visto in precedenza, con l"unica

differenza che nel templat

e è stato aggiunto anche un nuovo paragrafo con

all"interno la visualizzazione della chiave descrizione.

Insomma i primi passi per poter visualizzare dei dati recuperati in modo asincrono da

una sorgente esterna via HTTP.

Negli esempi visti, non abbiamo s

fruttato la peculiarità di TypeScript di definire la

tipologia di dati per ogni variabile.

Potremmo definire listanews con tipo <any>, anche se avrebbe più senso andare

a definire una tipologia di dati specifica che potremmo chiamare News . Cosa che

impareremo a fare in uno dei prossimi capitoli.

4.5 Creiamo il componente Treni

dell'app

MetroChat

Vediamo di applicare questi concetti anche all"applicazione che stiamo progettando.

La schermata iniziale che l'utente

visualizza non appena acced e all'applicazione,

nell'ipotesi abbia già eseguito l'accesso con i propri dati, è una lista di treni in arrivo

nell'ipotetica stazione geolocalizzata in funzione della sua attuale posizione.

Ipotizzando per ora d "inserire manualmente questi dati, così come fatto per il

componente Notizie, potre sti creare un array di nome listametro e sfruttare le

nozioni appena apprese, per mostrarli in sequenza.

```
listametro = [

{idt:'ASD', linea:'Rossa', numchatting:32, tempo:125000},

{idt:'AKE', linea:'Verde', numchatting:29, tempo:145000},

{idt:'BFD', linea:'Gialla',
```

numchatting:47, tempo:155000}

];

Il componente lo devi creare

all"interno della cartella principale del progetto

sfruttando la linea di comando:

ng g component treni

che aggiungerà una cartella ***treni*** e il file *treni.component.ts*.

Ti ricordo che in automatico Angular modifica anche il file

app.module,

aggiungendo il nuovo componente all'array associato alla proprietà declarations

di @ngModule(), e in più la relativa riga di import.

Modificando il nome del selettore in <ca-treni> ottengo:

```
import { Component } from '@angular/core';

@Component({

selector: 'ca-treni',
```

```
template: `
<div class="listatreni">
<ul>
<li *ngFor="let metro of listametro">
<h2>{{metro.linea}}</h2>
<p>ID: {{metro.idt}}</p>
```

(continua)

```
<p>Partenza tra: {{metro.tempo}}</p>
```

```
</li>

</ul>

</div> `

})

export class TreniComponent {

listametro = [

{idt:'ASD', linea:'Rossa',
numchatting:32, tempo:125000},

{idt:'AKE', linea:'Verde',
numchatting:29, tempo:145000},

{idt:'BFD', linea:'Gialla',
```

numchatting:47, tempo:155000}

];

}

treni/treni.component.ts

Come puoi vedere, il componente mostra i

dati come il nome d

ella linea,

l'identificativo associato al treno e il tempo rimanente

alla partenza, che vedremo

sarà espresso in millisecondi.

Per testare il funzionamento del componente, dev i aggiungerlo a l template del

componente principale dell'app, ossia app.component.ts, in quanto ancora non

abbiamo visto come si naviga tra le diverse sezioni di un'app Angular.

Potresti inserirlo come se fosse un normale tag html dal nome <ca-treni>, subito

sotto il componente Menu inserito in precedenza:

```
<ca-menu></ca-menu>

<h1>MetroChat {{title}}</h1>

<ca-treni></ca-treni>
```

4.6 Visualizzare o nascondere elementi del

DOM: direttiva *ngIf

Altra direttiva importante è quella che ci servirà per

indicare se una determinat o

messaggio chat è stato selezionato come preferito o meno

, quindi in generale per

mostrare o meno un elemento del DOM. Vediamo un esempio legato alla domotica

per rendere le cose più “luminose”.

Ipotizziamo tu voglia progettare un"interfaccia in grado di visualizzare all"utente

l"elenco di tutte le stanze della casa con abbinata un"icona a forma di lampadina

nello stato acceso o spento.

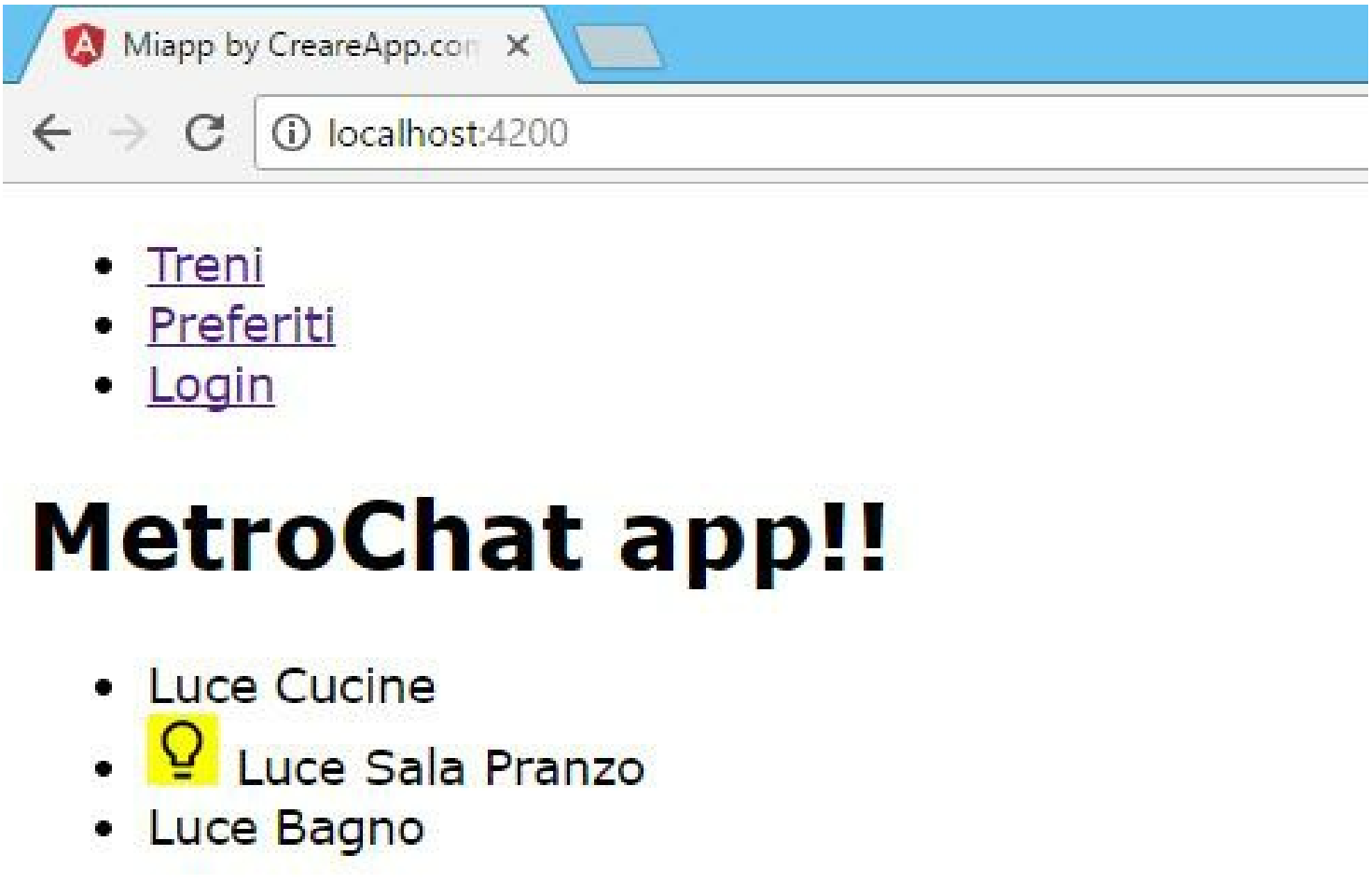
Una prima bozza di pagina, potrebbe essere simile a quella qui sotto, in cui elenco

all"interno di un tag <ul>, i nomi delle diverse stanze, abbinando

l"icona di una

lampadina, solo per le stanze attualmente illuminate (fig. 4.4):

```
<div class="lucistanze">

<ul>

<li>Luce Cucine</li>

<li><i class="material-icons
on">lightbulb_outline</i> Luce Pranzo

</li>

<li>Luce Bagno</li>

</ul>

</div>
```



dove, grazie all'ausilio dell'attributo di classe "on", sono in grado colorare lo sfondo

della lampadina e indicare così all"utente che la particolare stanza è

illuminata.

L'icona della lampadina è stata inserita con un tag <i>, sfruttando i l "font material"

di Google, quindi inserendo la seguente riga, nei meta tag del file *index.html*:

<link href="https://fonts.googleapis.com/icon?family=Material+Icons"

rel= "stylesheet">

Il risultato è simile a questo:

fig.4.4

L'unica differenza tra una riga e l'altra, è data dalla presenza o meno d el tag <i> a

cui è associata la visualizzazione della lampadina.

Ebbene in Angular è possibile usare una direttiva che ci permette di aggiungere

alcuni elementi nel DOM, sulla base di una condizione. Si tratta della

direttiva

***ngIf,** che ha una sintassi simile a questa:

<taghtml ***ngIf="condizione o espressione"** >...</taghtml>

Osserva la presenza del simbolo * e come la lettera I sia stata scritta in maiuscolo , il

tutto senza inserire spazi tra l"asterisco e la prima lettera.

Nell'ipotesi la condizione inserita

all'interno delle doppie virgolette sia vera,

il tag

html a cui è applicata la direttiva, sarà visualizzato nella pagina.

Questo è uno dei possibili comportamenti ma a seconda della condizione, io posso

visualizzare un blocco con associata una specifica "etichetta" o in alternativa, un

altro blocco, seguendo il classico stile dell'istruzione if, else di molti linguaggi

di programmazione.

4.7 Esempio d'uso della direttiva *ngIf

Abbiamo già imparato a ripetere ciclicamente un p articolare elemento del DOM ,

sulla base di valori memorizzati all"interno di un array.

Visto che nell"esempio analizzato in precedenza il tag <li> si ripete più volte, potrei

sfruttare la direttiva *ngFor per visualizzare i dati d i ogni stanza, e progettar e un

template simile a questo:

```
<div class="lucistanze">

<ul>

<li *ngFor="let lucisingole of lucidb">

<i *ngIf="lucisingole.stato == 'ON'"
class="material-icons on">

lightbulb_outline </i>

{{lucisingole.stanza}}

</li>

</ul>

</div>
```

Come puoi osservare, all'interno del tag

, ho inserito la notazione ***ngIf**, e

all'interno delle doppie virgolette, un "espressione che rappresenta la condizione da

verificare:

lucisingole.stato == 'ON'

Il simbolo di doppio uguale ==, fa p

arte degli operatori di confronto di

TypeScript/JavaScript, come ben saprai.

A seconda del risultato d
ell"uguaglianza, “vero” o “falso”, il
corrispondente tag <i>

con associata la direttiva *ngIf, verrà o
meno visualizzato.

Il codice completo dell'ipotetic

o componente lucicasa.component.ts,
con

selettore <ca-lucicasa>, potrebbe essere
scritto in questo modo:

```
import { Component } from
'@angular/core';
```

```
@Component({

selector: 'ca-lucicasa',

template: `

<div class="lucistanze">

<ul>

<li *ngFor="let lucisingole of lucidb">

<i *ngIf="lucisingole.stato == 'ON'"
class="material-icons on">

lightbulb_outline

</i> {{lucisingole.stanza}}
```

```
</li>

</ul>

</div> `,

styleUrls: ['./lucicasa.component.css']

})

export class LucicasaComponent {

lucidb = [{ stanza: 'Luce Cucine',

stato: 'OFF',

luminosita: 5

},
```

```
{ stanza: 'Luce Sala Pranzo',
stato: 'ON',
luminosita: 2
},
{ stanza: 'Luce Bagno',
stato: 'OFF',
luminosita: 8
}];
}
```

lucicasa/lucicasa.component.ts

dove ho inserito dei dati di test all'interno di un oggetto JSON , mentre le regole CSS

per modificare lo sfondo dell'icona,

le ho

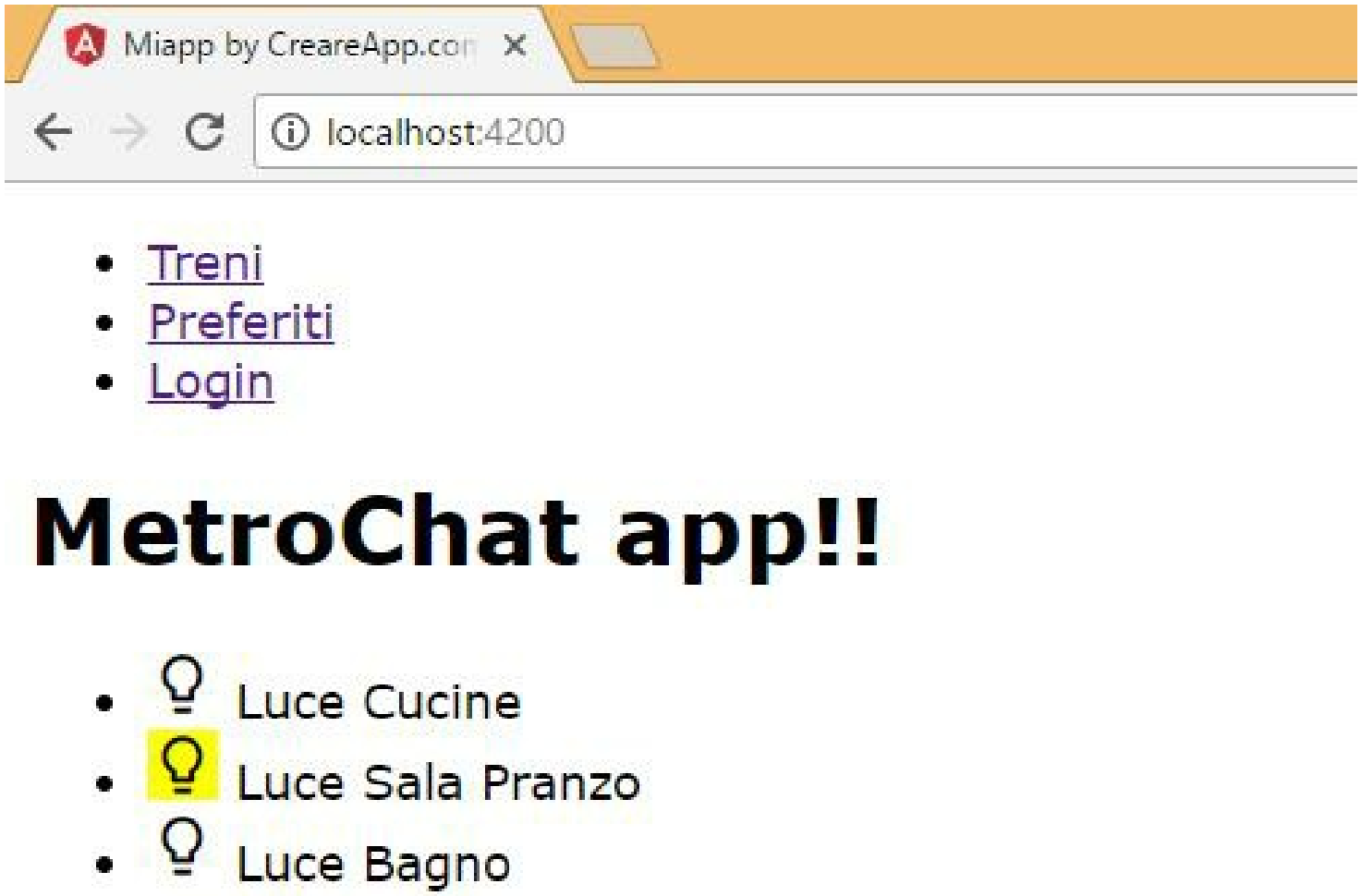
inserite all'interno di un file

lucicasa.component.css, specifico del componente.

Il codice CSS, potrebbe contenere all''interno una regola di nome .on, che imposta il

colore di sfondo dell'elemento:

```
.on {

background-color: yellow;

}
```



In questo caso abbiamo sfruttat o la proprietà stato definita all"interno dell"oggetto

JSON, per impostare una condizione del tipo:

lucisingole.stato == 'ON'

che risulterà essere vera, solo se la relativa proprietà memorizzata nei diversi oggetti,

risulta pari alla stringa “ON”.

4.8 Esempio d'uso del blocco else

A partire da Angular 4, la direttiva *ngIf è stata potenziata, ed è possibile

utilizzare

un costrutto molto simile alla cla ssica condizione if, else di molti linguaggi di

programmazione.

Questo ci può essere d'aiuto per visualizzare il simbolo della lampa dina, anche nella

condizione in cui sia spenta.

Per fare questo si può sfruttare questa notazione:

<taghtml *ngIf="condizione; **else secondoblocco"** >...</taghtml>

<ng-template #secondoblocco>...</ng-template>

61

Se la condizione è “vera”, verrà visualizzato il tag <taghtml>, mentre se “falsa”,

verrà visualizzato un secondo blocco di codice che d ovrà essere racchiuso all'interno

della direttiva <ng-template> a cui è stata associata la variabile del template

locale #showluce, affinché sia identificata.

```
<div class="lucistanza">

<ul>

<li *ngFor="let lucisingole of lucidb">

<span *ngIf="lucisingole.stato == 'OFF'; else showluce">

<i class="material-icon">lightbulb_outline</i>

{{lucisingole.stanza}}

</span>

<ng-template #showluce>

<span>
```

```
<i class="material-icon
on">lightbulb_outline</i>

{{lucisingole.stanza}}

</span>

</ng-template>

</li>

</ul>

</div>
```

Non appena impareremo a gestire gli eventi, vedremo come implementare anche

delle funzionalità che ci permett eranno di accendere le lampadine di una stanza, con

un click sopra alla relativa icona.

Come dicevo, tutte queste nozioni che ci serviranno per lo sviluppo dell'applicazione

MetroChat, quando dovremo inserire tra i preferiti un

determinato messaggio

chat, cliccando sulla relativa icona.

Altri esempi li puoi vedere consultando la pagina ufficiale:

https://angular.io/api/common/NgIf

4.9 Condizioni multiple: direttiva *ngSwitch

Grazie alla direttiva strutturale

***ngIf**, riusciamo a visualizzare un determinato

elemento del DOM, sulla base del risultato - “vero” o “falso” - di un'espressione.

In altri casi, potresti avere la necessità di effettuare una scelta sulla base di più valori

possibili, non solo “vero” o “falso”.

Per questo, ci viene in aiuto la direttiva **ngSwitch**, che dovrà essere usata con la

seguente sintassi:


```
<tagContenitore
[ngSwitch]="espressione">

<tagFiglio
*ngSwitchCase="valoreA">AAA</tagfig

<tagFiglio
*ngSwitchCase="valoreB">BBB</tagfig

<tagFiglio
*ngSwitchCase="valoreC">CCC</tagfig
```

</tagContenitore>

NB: Nota come ngSwitch, sia stata inserita all'interno delle **parentesi**

quadre. Questa notazione ti diventerà familiare ed è stata adottata dal team

di Angular, per collegare una proprietà di un componente o di un elemento

HTML del DOM a dei dati o a un’espressione presente alla destra

dell’uguale. Questa tecnica viene usata in Angular per inizializzare un

elemento o impostare uno stato per una direttiva, come nel nostro caso.

Questa direttiva si applica a strutture HTML che presentano un tag padre e tanti figli.

L'espressione inserita alla destra dell"uguale verrà valutata e sulla base del valore

restituito, sarà mostrato uno dei tag figlio.

E' chiaro che verrà visualizzato solo il tag figlio che avr

à il corrispondente valore

inserito all'interno della direttiva *ngSwitchCase.

Quest"ultima può essere inserita anc he sfruttando la notazione con le parentesi

quadre, ossia scrivendo:
[ngSwitchCase]

Riprendendo l'esempio dell'ipotetico applicativo per la gestione da remoto della

nostra abitazione, se all'interno della proprietà luminosità, avessi un valore che

rappresenta l'intensità della luce attualmente imposta ta in una stanza, da 1 a 5, allora

potresti mostrare all'utente una

lampadina di colore diverso, dal nero (buio), al giallo

paglierino (tanta luce).

Provando per ora a inserire questa informazione sotto forma di stringa:

<h1>Luce proveniente dall'esterno</h1>

<div [ngSwitch]="luminosità">

<span *ngSwitchCase="1"
class="buio">Buio

**<span *ngSwitchCase="2"
class="penombra">Molto
Nuvoloso**

<span *ngSwitchCase="3"
class="nuvoloso">Nuvoloso

<span *ngSwitchCase="4"
class="solenuvole">Sole e
Nuvole

<span *ngSwitchCase="5"
class="sole">Sole

Nel caso in cui il valore impostato per
la variabile

luminosità fosse 2, allora

verrebbe visualizzato il tag c on
l'attributo di classe impostato a "

penombra" e la

scritta “Molto Nuvoloso”.

63

Nel caso in cui il valore della variabile non corrisponda ai valori da 1 a 5, non verrà

visualizzato alcun tag figlio.

In alternativa è possibile usare una terza direttiva

, *ngSwitchDefault, che ti

permetterà di visualizzare sempre almeno un tag figlio, nell'ipotesi le

condizioni non

siano soddisfatte dai precedenti ngSwitchCase.

Potrebbe succedere infatti che l'applicazione non sia in grado di leggere il dato

remoto, fornito da l sensore. In questo caso potresti visualizzare comunque un

messaggio informativo all"utente:

```
<h1>Luce proveniente dall'esterno</h1>

<div [ngSwitch]="luminosità">

<span *ngSwitchCase="1"
```

```
class="buio">Buio</span>

<span *ngSwitchCase="2"
class="penombra">Molto
Nuvoloso</span>

<span *ngSwitchCase="3"
class="nuvoloso">Nuvoloso</span>

<span *ngSwitchCase="4"
class="solenuvole">Sole e
Nuvole</span>

<span *ngSwitchCase="5"
class="sole">Sole</span>

<span *ngSwitchDefault
class="errore">Errore
Lettura</span>
```

64

Capitolo 5

Cambiare lo stile di elementi del DOM

5.1 Aggiungere o togliere proprietà CSS con

ngStyle

Entriamo nel vivo delle **“direttive attributo”** , in grado di cambiare l"aspetto o il

comportamento di un elemento del DOM.

Abbiamo già sottolineato che in Angular , il collegamento tra una proprietà di un

elemento del DOM e i dati, avviene grazie alla notazione delle doppie parentesi

quadre - [ ] -, che ci permette di aggiungere delle direttive *attributo*,

come attributi di

un elemento, con la stessa notazione usata nei CSS per i selettori di attributo.

Nel mondo delle pagine web, i fogli di stile o CSS

, come ben sai, sono l'elemento

base con cui andare a modificare l'aspetto di ogni elemento di una pagina, dal testo,

alla disposizione di immagini, ai colori etc. Anche in Angular posso sfruttare i fogli

di stile per modificare l'aspetto di ogni

elemento definito all'interno del template.

Oltre a questo, posso avva lermi anche di alcune direttive che vanno a modifi care

l' **aspetto di un componente** , senza che manualmente debba andare a inserire le

diverse regole CSS nel template. La prima di queste

, agisce direttamente

sull'attributo *style* di un tag html.

Vediamo subito un esempio. Se volessi cambiare **dinamicamente** la dimensione

del

un testo definito qui sotto, come potesti fare?

Questa è una news con il testo molto

piccolo

65

Devi avere la possibilità di iniettare un valore diverso da 1 1px all'interno del tag p

della pagina. Si ottiene questo con la direttiva ngStyle.

```
import { Component } from
'@angular/core';

@Component({

selector: 'ca-news',

template: ` <p [ngStyle]="miostile">

News con il testo molto piccolo

</p> `

)}

export class NewsComponent {

miostile: object;
```

constructor() {

this.miostile = {'font-size': '14px'};

}

}

notizie/news.component.ts

L'effetto che otterrai, sarà un ingrandimento del carattere a 14 px.

Come vedi , all"interno del tag p ho inserito un attributo con la classica notazi one

delle parentesi quadre, che mi permetterà di valorizzare la proprietà

style del tag

HTML con un oggetto JavaScript, la cui prima chiave è proprio la proprietà CSS che

si vuole impostare.

NB: Nota come miostile sia un oggetto "Literal JavaScript", dove la chiave

corrisponde alla proprietà CSS che si vuole impostare, il tutto racchiuso

all'interno di singole virgolette. In realtà per alcune proprietà potresti farne a

meno, ma per non sbagliare è meglio inserirle sempre.

Se oltre a cambiare la dimensione, volessi cambiare anche la colorazione del testo,

allora dovrei valorizzare la variabile miostile con il seguente oggetto JS:

// ho aggiunto due prorietà CSS

this.miostile = {'font-size': '14px', 'color': 'red'};

Una caratteristica della direttiva ngStyle, è che eventuali attributi di stile applicati

direttamente nel template, non vengono persi ma si aggiungono a quelli applicati con

la direttiva. Se , ad esempio , il testo avesse una tipologia di carattere

“Arial”

applicata tramite una regola CSS “inline”:

```
import { Component } from '@angular/core';

@Component({
```

(continua)

```
selector: 'ca-news',

template: `<p style="font-family:
Arial" [ngStyle]="miostile">

News con il testo molto piccolo

</p>`

)}

export class NewsComponent {

miostile: object;

constructor() {

this.miostile = {'font-size': '14px',
'color': 'red'};
```

}

}

notizie/news.component.ts

il risultato finale che visualizzerò nel

DOM, sarà un testo con caratt ere “Arial”,

dimensione 14 px, colorato di rosso.

Chiaramente potre sti inserire direttamente nel template alcune proprietà che rit ieni

non debbano cambiare, e lasciare quelle variabili nella direttiva; oppure

aggiungere

tutto nella direttiva, esplicitando le sezioni modificabili, con il nome di una variabile

da valorizzare nel corpo della classe.

import { Component } from '@angular/core';

@Component({

selector: 'ca-news',

template: ` <p [ngStyle]="{'font-family': 'Arial', 'color':'red',

'font-style': **dimcarattere**}">News con

```
il testo molto piccolo</p> `

)}

export class NewsComponent {

dimcarattere: string;

constructor() {

// dimensione carattere con unità di
misura

this.dimcarattere = '14px' ;

}

}
```

notizie/news.component.ts

L'oggetto che riceve la direttiva ngStyle, accetta la scrittura di proprietà CSS, con

due tipologie di notazione: "kebab - case" (quella usata fino ad ora) o "camel case"

ossia eliminando il trattino (-) da quei nomi che li presentano e iniziando la seconda

parte della parola in maiuscolo.

Meglio a farsi che a dirsi. Ecco alcuni esempi di trasformazione di proprietà CSS con

questa notazione:

67

- padding-top > paddingTop
- border-bottom > borderBottom

e così via.

5.2 Aggiungere o togliere regole CSS

Sempre nell'ambito d elle modifiche all"aspetto di un elemento o di un tag html

presente nel DOM, sappiamo che con gli attributi CSS di classe, è possibile applicare

una regola o insieme di regole a più tag html . Ad esempio, il paragrafo qui sotto ha

impostato un attributo di classe "news", che sarà defin ito come regol a nel foglio di

stile.

Testo di una notizia, formattata con i CSS

Ora se il mio obiettivo è quello di andare a modificare dinamicamente l'aspetto di

alcuni elementi della pagina bypassando la direttiva ngStyle, l'alternativa è usare

la

direttiva ngClass.

Una delle peculiarità di quest'ultima direttiva è che può essere utilizzata con stringhe,

array, oppure oggetti.

5.3 Uso di ngClass con una proprietà stringa

L'esempio è molto simile a quello visto per l'altra direttiv

a ngStyle. Si dovrà

“incastrare” la direttiva all'interno delle

parentesi quadre, e il valore dovrà essere

impostato come se fosse una stringa rappresentativa delle diverse classi da

aggiungere a quelle già eventualmente presenti.

@Component({

selector: 'ca-news',

template: `<p **[ngClass]="mieclassi" class="big-text"** >News con

testo molto piccolo</p>`,

styleUrls: ['./miostile.css']

```
})
export class NewsComponent {
mieclassi:string;
constructor() {
```

(continua)

```
this.mieclassi = 'text-center light-
blue';
}
```

}

notizie/news.component.ts

In questo caso, ho usato un file esterno *miostile.css*, con all"interno definire le

regole light-blue, text-center, e big-text.

```
.big-text{

font-size:18px;

}

.text-center{

text-align:center;
```

}

.light-blue{

color:#ADD8E6;

}

Le classi definite all"interno della proprietà mieclassi, verranno aggiunte a quella

già presente e definita nel template (big-text).

In questo modo posso modificare dinamicamente lo stile di specifiche sezioni del

template, usando nomi di regole definite nel file CSS.

5.4 Uso di ngClass con un array

Spesso capita che i valori da

aggiungere dinamicamente, siano recuperati da una

sorgente esterna che li confeziona all'interno di un array. In Angular è possibile usare

anche la seguente notazione:

@Component({

selector: 'ca-news',

```
template: `<p [ngClass]="mieclassi"
class="big-text"></p>`,

styleUrls: ['./miostile.css']

})

export class NewsComponent {

mieclassi:Array<string>;

constructor(){

// applico due attributi di classe

this.mieclassi = ['text-center', 'light-
blue'];

}
```

}

notizie/news.component.ts

5.5 Uso di ngClass con un oggetto

Una caratteristica interessante della direttiva ngClass, è che posso sfruttarla non

solo per aggiungere attributi di classe come visto fino ad ora ma anche per togliere

attributi di classe, sulla base di un valore booleano VERO o FALSO.

Questo è possibile solo sfruttando la notazione qui sotto, che prevede l'inserimento di

un oggetto JavaScript le cui chiavi sono rappresentate dagli attributi di classe, mentre

il valore è rappresentato da un dato booleano, impostato a "true" o "false" a seconda che tu voglia aggiungere o eliminare quel particolare attributo.

this.isCentered = true;

this.isBlue= false;

this.mieclassi = {'text-center' : this.isCentered,

'light-blue': this.isBlue};

Questo, ad esempio, può essere usato per mostrare o d oscurare un determinato tag

html della pagina.

@Component({

selector: 'ca-news',

template: `<p [ngClass]="mieclassi" class="big-text"></p>`,

styleUrls: ['./miostile.css']

})

```
export class NewsComponent {

mieclassi: object;

isCentered: boolean;

isBlue: boolean;

constructor() {

this.isCentered = true;

this.isBlue= false;

this.mieclassi = {'text-center' :
this.isCentered, 'light-blue':

this.isBlue};
```

}

}

notizie/news.component.ts

In questo caso, il risultato finale non mostrerà l'attributo di classe light-blue, in

quanto è stato oscurato dal valore impostato per la variabile isBlue:

<p class="text-center big-text">

70

Formattare i dati con i PIPE

6.1 Formattare le date e l'ora

La visualizzazione dei dati nel template, può essere arricchita di uno strumento

molto potente che viene chiamato "Pipe".

Per la nostra applicazione

MetroChat, sfrutteremo un Pipe personalizzato che

svilupperemo ad hoc.

Chiaramente in Angular vi sono già diversi Pipe sviluppati, che posso

sfruttare sia

per formattare delle date, sia per formattare del testo.

Il nome deriva dal fatto che si fa seguire il sim bolo di pipe |, al nome della proprietà

che si vuole visualizzare nel template con la tecnica dell'interpolazione:

{{ nomeproprietà | NOME_PIPE: PARAMETRO }}

Ad esempio, la data di un giorno può essere rappresentata in diversi modi, pertanto

potrebbe es sere utile avere uno strumento che al volo, mi permetta di cambiare il

formato senza dover scrivere, di volta in volta, del codice nella classe.

Nel caso di una data, sarà necessario usare il "DatePipe" che ha questa notazione:

{{datadaformattare | **date: PARAMETRO**}}

dove datadaformattare è una variabile di tipo Date.

Ad esempio, nella seguente notazione:

// visualizza la data nel formato: MM/GG/AA

{{datadaformattare | date:'shortDate'}}

visualizzo la data nel formato Mese, Giorno e Anno a 2 cifre (es. 12/01/20, ossia il

primo dicembre 2020).

Altre notazioni di uso comune sono:

'shortDate': equivalente a MM/GG/AA (Es. 12/25/21)

'fullDate': equivalente a

GiornoSettimana, Mese GG, AAAA

'longDate': equivalente a Mese GG, AAAA

Ognuna di queste può e

ssere creata anche sfruttando la notazione con lettere

rappresentative del giorno, mese e anno, come indicato qui sotto:

d: numero indicativo del giorno (1 cifra senza zero. Es. 1, 9, 15)

M: numero indicativo del mese (deve essere maiuscolo. Es. 9,12)

y: numero minimo cifre indicative dell'anno (es. 4 cifre)

MMMM: nome del mese per esteso (es. June)

MMM: nome del mese abbreviato a tre lettere (es. Jun)

EEEE: nome del giorno della settimana (es. Monday)

EEE: nome del giorno della settimana abbreviato (es. Mon)

Quindi, per ottenere la stessa data indicata sopra con mese, giorno e anno:

// NB: le ho invertite per avere

MM/GG/AAAA

{{datadaformattare | date:'Mdy'}}

In alternativa, avrei potuto mantenere lo stesso ordine, inserendo manualmente il

simbolo di /:

{{datadaformattare | date:'M/d/y'}}

Nel caso di una misura del tempo, posso sfruttare le lettere:

h: ora

m: minuti

s: secondi

Riassumendo, nell'ipotesi volessi formattare una data creata con la classica

istruzione:

```
data = new Date();
```

potrei scrivere:

```
import {Component} from
'@angular/core';

@Component({

selector: 'ca-pipe',
```

```
template: `

<h3>Esempio di PIPE con le Date</h3>

<p>{{dataoggi | date:'shortTime'}}</p>

<p>{{dataoggi | date:'mediumDate'}}
</p>

// mostro GG Mese AAAA

<p>{{dataoggi | date:'d MMMM y'}}
</p>

// mostro l'ora HH:MM:SS

<p>{{dataoggi | date:'hms'}}</p> `

})
```

```
export class TestPipe {

dataoggi: Date;

constructor() {

this.dataoggi = new Date();

}

}
```

pipe/test.pipe.ts

6.2 Formattare stringhe e numeri

Proseguendo la panoramica sulle tipologie di pipe predefinite, vediamo quelle che ci

permettono di trasformare un testo in maiuscolo. Si tratta del pipe "UppercasePipe".

La sintassi è:

{{ stringa | **uppercase** }}

Analogo discorso per il minuscolo:

{{ stringa | **lowercase** }}

Nel caso di numeri frazionari, spesso si ha la necessità di rappresentarli con un certo

numero di decimali. La sintassi sarà:

{{ numero | **number: 'PARAMETRO'** }}

dove parametro deve essere costruito con questa sintassi:

PARAMETRO = Minimo-num-cifre-intere . Minimo numero cifre decimali -

Massimo numero cifre decimali

Ad esempio, usando la notazione:

2.1-2 // Es. 12.43, 12.4, 08.3

intendo un numero rappresentato da due cifre intere, e da 1 a 2, cifre decimali. Se il

numero fosse costituito da più di due cifre decimali, avver

rebbe l'arrotondamento

automatico.

Applicando il pipe alla variabile numero valorizzata a 1.348 otterrei:

{{ numero | number: '2.1-2' }} // ottengo 01.35

Analogo discorso per un n

umero percentuale, che necessiterà dell'uso del pipe

"PercentPipe" con la seguente sintassi:

{{ numero | **percent: 'PARAMETRO'** }}

dove la notazione *: 'PARAMETRO'* in questo caso è opzionale ma se inserito, segue

la stessa logica vista per un numero.

Ad esempio, se il numero da trasformare in percentuale fosse 0.3267 e scrivessi:

74

{{ numero | percent }} // ottengo: 32.67%

Se invece aggiungessi il parametro:

{{ numero | percent: '2.1-1' }} // ottengo: 32.7%

Altro pipe che spesso viene usato in fase d i test per visualizzare un oggetto JSON, è

dato dal pipe “JsonPipe”, che equivale ad utilizzare il metodo JSON.stringify(),

come impareremo a fare quando parleremo di invio e ricezione dati via HTTP.

Se avessi un oggetto JavaScript del tipo:

this.metro = {'idt':'12345', 'linea': 'Rossa'};

e provassi a visualizzarlo nel template, otterrei solo una stringa [object Object],

mentre se lo scrivessi nella forma:

{{ metro | **json** }}

otterrei la stessa visualizzazione di come è rappresentato nella classe.

6.3 Creare Pipe personalizzati

Dopo aver visto alcuni dei pipe più usati in Angular, vediamo come si possa creare

un nostro pipe personalizzato che ci servirà all'interno dell'applicazione MetroChat ,

per rappresentare i minuti e secondi mancanti alla partenza

di un treno , con una

notazione del tipo:

{{ attesa | mmss }}

L'ipotesi che avevamo fatto infatti, quando abbiamo visto come rappresentare la

sequenza di treni in arrivo grazie al componente *TreniComponent*, era che l'orario di

partenza fosse espresso in millisecondi.

Per conoscere il tempo mancante alla partenza, dovremo fare la differenza tra l'ora

attuale espressa in millisecondi e quella

di partenza, per poi trasformare il tutto in

minuti e secondi e rappresentarli in un formato comprensibile al volo dall"utente.

In JavaScript, la funzione getTime() ci permette di ottenere l'ora attuale espressa in

millisecondi a partire dalla famosa data di Unix (01/01/1970). Ipotizzando pertanto

che la data di partenza sia anch'essa espressa nello stesso formato, dov rei inserire le

semplici operazioni qui sotto per avere i minuti e secondi:

attesa = orapartenza-oraattuale;

minuti = Math.floor(attesa / 60000);

secondi = Math.floor((attesa - this.minuti*60000)/1000);

dove ho diviso per 60*1000 per avere il numero d i minuti, arrotondati con il metodo

floor(), e sottratto questi al tempo di attesa, per ottenere la parte mancante in secondi.

A questo punto dobbiamo mostrare i due

dati nella forma MM:SS dove con MM

intendo i minuti e con SS i secondi.
Chiaramente devo a nteporre degli zeri
per valori

inferiori a 10, quindi potrei avvalermi di
una semplice funzione di appoggio che

sfrutta un array di 4 elementi,
valorizzato con il numero a una o due
cifre , per poi

traslare le prime due posizione
dell"array verso sinistra.

duecifre(numero:number,zero:string,leng
{

```
return (new
Array(length+1).join(zero)+numero).slic
length);

}
```

ottenendo così:

```
attesa = orapartenza-oraattuale;

minuti = Math.floor(attesa / 60000);

secondi = Math.floor((attesa -
this.minuti*60000)/1000);

contorovescia =
duecifre(minuti,'0',2)+':'+
duecifre(secondi,'0',2);
```

```
duecifre(numero:number,zero:string,leng
{

return (new
Array(length+1).join(zero)+numero).slic
length);

}
```

Queste righe, possono diventare la ba

se per creare un pipe personalizzato, che

potremmo chiamare **mmss**, da usare nel template del componente *TreniComponent*.

Per creare un pipe, possiamo avvalerci della linea di comando, che offre il vantaggio

di aggiungere già gran parte delle righe che costituiscono un pipe.

Per mantenere ordine tra le diverse cartelle, posso creare una cartella apposita di

nome *pipe* all'interno della quale ci sposteremo per eseguire la linea di comando:

ng g pipe mmss

Verrà generato un nuovo file dal nome

mmss.pipe.ts, che sarà aggiunto in

automatico all"interno del file app.module.ts, al fine di renderlo disponibile per

tutti i componenti, senza necessità di importare alcuna riga aggiuntiva.

```
import { NgModule } from '@angular/core';

import { BrowserModule } from '@angular/platform-browser';

import { MiaAppComponent } from './app.component';
```

```
import { MenuComponent } from
'./menu/menu.component';

import { MmssPipe } from
'./pipe/mmss.pipe';

@NgModule({

imports: [ BrowserModule ],

declarations: [ MiaAppComponent,
MenuComponent, MmssPipe],

declarations: [ MiaAppComponent ],

bootstrap: [ MiaAppComponent ]

})
```

export class AppModule {}

app.module.ts

Il file *mmss.pipe.ts* così creato, avrà questa struttura:

import { Pipe, PipeTransform } from '@angular/core';

@Pipe({

name: 'mmss'

})

export class MmssPipe implements PipeTransform {

```
transform(value: any, args?: any): any {

return null;

}

}
```

pipe/mmss.pipe.ts

77

Come puoi osservare si tratta di una classe, a cui è stato aggiunto un decoratore

@Pipe() con una proprietà name, pari al nome scelto per il pipe.

La classe implementa PipeTransform, con il metodo transform che dovrà essere

sviluppato. I parametri ricevuti in ingresso sono il dato aggiunto nel template, e dei

dati opzionali, così come visto per altri pipe predefiniti, con la n

otazione

nomepipe:'PARAMETRI'

Nel nostro caso, non passeremo dei parametri aggiuntiv i, ma solo il dato legato alla

differenza tra il tempo di partenza e il tempo attuale espresso in millisecondi.

Sfruttando pertanto le righe già sviluppate:

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({

name: 'mmss'

})

export class MmssPipe implements PipeTransform {

minuti: number;
```

```
secondi: number;

contorovescia: string;

transform(attesa: number, args?: any):
string {

this.minuti = Math.floor(attesa/60000);

this.secondi = Math.floor((attesa -
this.minuti*60000)/1000);

this.contorovescia =
this.duecifre(this.minuti,'0',2)+':'

+ this.duecifre(this.secondi,'0',2);

return this.contorovescia;
```

```
}

duecifre(numero:number,zero:string,leng
{

return (new
Array(length+1).join(zero)+numero).slic
length);

}

}
```

pipe/mmss.pipe.ts

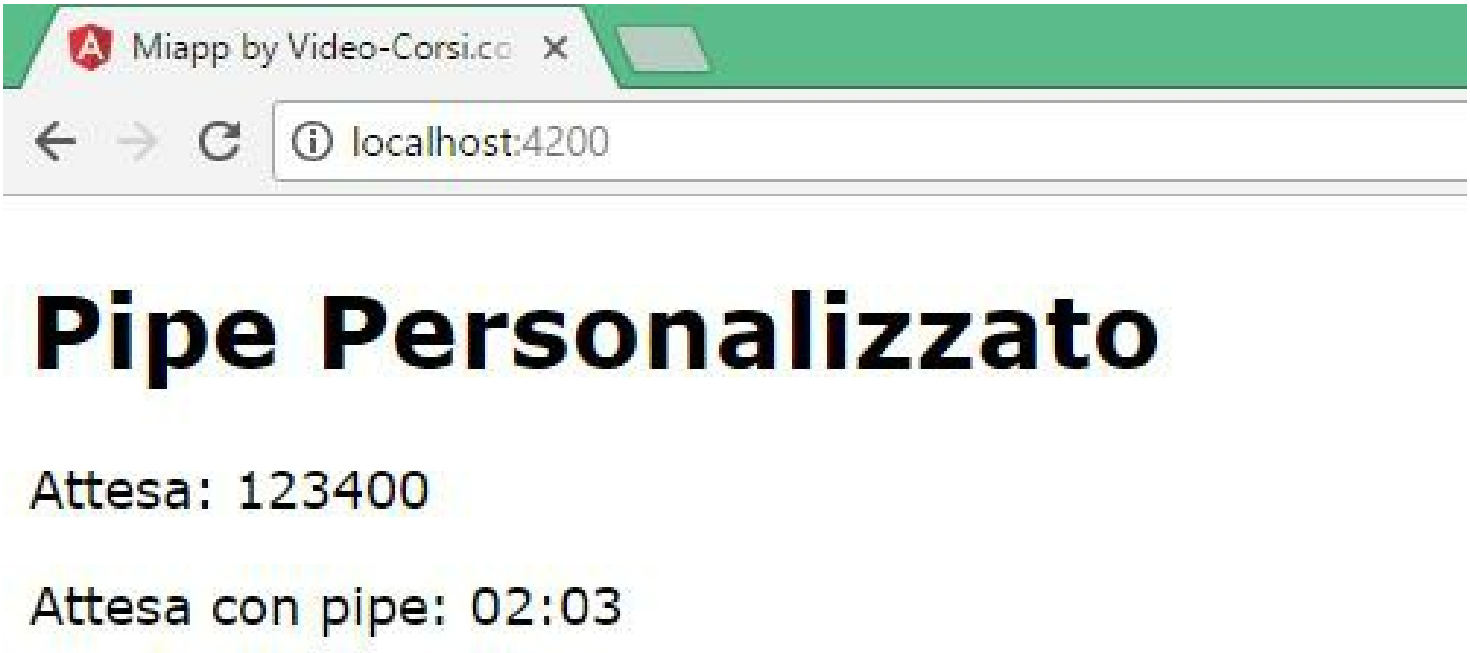
Il valore restituito da transform è una stringa. Questo pipe vedremo ci servirà per

rappresentare l'orario di partenza sotto forma di

conto alla roves cia non appena

avremo sviluppato i primi componenti "intelligenti".

Pertanto se la proprietà attesa fosse valorizzata con „123400" otterrei:



78

79

Capitolo 7

Modellare i dati

7.1 Una classe per definire il tipo di dati

Fino ad ora abbiamo visto

che i dati visualizzati all'interno del template di un

componente, venivano inseriti direttamente all'interno del corpo della classe,

valorizzando una variabile.

Questo può andare bene per fare dei piccoli test ma nelle applicazioni reali,

avrai la

necessità di costruire delle r
appresentazioni dei dati, che siano facili
da gestire e da

mantenere aggiornati , oltre che essere
coerenti con il tipo di informazione da

memorizzare in ognuno.

Se poi pensiamo che la maggior parte
dei dati che un'applicazione visual

izzerà,

arriveranno da una sorgente esterna,

tipicamente sotto forma di oggetti JSON,

dobbiamo trovare un modo per maneggiarli senza introdurre potenziali errori.

In previsione di dover gestire dei dati di un certo

“TIPO” con la possibilità di

manipolarli, il pensiero va subito alle classi.

L'idea è quella di definire una classe modello, che conterrà all'interno la definizione

dei diversi campi in grado di rappresentare una particolare tipologia d''informazione.

In sostanza si crea un nuovo tipo di dati, che potremmo usare esattamente come si fa

per un qualsiasi tipo di dati primitivo (numero, stringa, etc.).

Riprendendo il classico esempio delle notizie, costituite da un titolo, una descrizione

e da un identificativo numerico per accedere all'eventuale detta

glio, queste

potrebbero essere visualizzare all'interno del template del componente

<ca-listanews>, con la classica direttiva *ngFor:

- **Titolo Notizia 1**

  Descrizione Notizia 1

- **Titolo Notizia 2**

  Descrizione Notizia 2

- **Titolo Notizia 3**

  Descrizione Notizia 3

- **Titolo Notizia 4**

  Descrizione Notizia 4

Ogni riga rappresenta un'informazione che potre sti modellare definendo la classe di

nome News, da memorizzare nel file *news.model.ts* in questo modo:

```
export class News {

constructor(public id: number, public titolo: string,public desc: string){

}

}
```

Ogni qualvolta definiamo una classe in TypeScript comprensiva di costruttore, nella

realtà avvengono multiple dichiarazioni allo stesso istante.

I tre parametri di nome

id, titolo e descrizione, con relativo **tipo di dati** ,

grazie alle peculiarità di TypeScript, sono a tutti gli effetti delle proprietà della classe

definite con lo stesso nome.

E" proprio il compilatore TypeScr ipt che crea una proprietà pubblica per ogni

parametro presente nel costruttore, assegnando gli il valore non appena

viene creato

un nuovo oggetto con l'operatore *new*.

E' sempre buona norma separare i diversi pezzi che costituiscono l"applicazione,

quindi t i consiglio di creare una cartella di nome *model*, in cui andare a salvare le

diverse rappresentazioni dei dati.

I file tipicamente vengono salvati aggiungendo al nome identificativo della classe,

l'estensione *.model.ts*

Nota la presenza della parola export, necessaria al fine di esportare la classe come

modulo e poterla poi importare nei diversi componenti che avranno la necessità di

gestire questo tipo di dati.

Per ogni parametro del costruttore ho aggiunto la parola public, anche se avrei

potuto ometterla, in quanto TypeScript considera le proprietà senza indicazione sulla

visibilità, sempre di tipo public

La notazione TypeScript abbreviata qui sopra, sostanzialmente equivale a scrivere:

```
export class News {

// variabili di classe

id: number;

titolo: string;

desc: string;

constructor(id:number, titolo:string, desc:string) {
```

```
this.id = id;

this.titolo = titolo;

this.desc = desc;

}

}
```

I puristi della programmazione ad oggetti, potrebbero storcere il naso, in q

uanto

definire pubbliche proprietà di una classe è sempre rischioso, ma al limite possono

sempre creare dei **getter** e **setter** e rendere le proprietà private.

Come ogni classe, potresti non limitarti solo alla definizione di proprietà ma definire

anche dei me todi, che andranno ad effettuare delle operazioni specifiche sui dati

ricevuti.

Limitandoci per ora alle proprietà della classe, nel caso volessi creare e valorizzare

un nuovo dato di tipo News, potre sti sfruttare la classica notazione con

l'operatore

new, e inserire nel compone nte che necessita di usare questo tipo di dati, le seguenti

righe:

82

// Array di oggetti News

listanew: News[] = [];

constructor() {

this.listanew.push(new **News(1, 'Titolo Notizia 1', 'Descrizione**

notizia 1'));

}

Come vedi, ho passato al costruttore della classe News, tre parametri, che sono i dati

della prima notizia da visualizzare. Questo oggetto è stato poi inserito all'interno

dell'array listanew, che conterrà proprio una lista di oggetti di tipo News.

E' chiaro che Angular n on sa come rappresentare l'oggetto News, quindi dobbiamo

indicargli dove si trova la relativa definizione, sfruttando l'importazione della classe.

Pertanto il componente

<ca-listanews>, progettato nelle precedenti lezioni,

diventerebbe:

```
import { Component } from '@angular/core';

import { News } from './../model/news.model';

@Component({
```

```
selector: 'ca-listanews',

template: `

<div class="news">

<ul>

<!--elemento che si ripete N volte-->

<li *ngFor="let notizia of listanews">

<h2>{{notizia.titolo}}</h2>

<p>{{notizia.desc}}</p>

</li>

</ul>
```

```
</div> `

})

export class NotizieComponent {

// Array di oggetti News

listanews: News[] = [];

constructor () {

this.listanews.push(new News(1, 'Titolo
Notizia 1', 'Descrizione

notizia 1'));

this.listanews.push(new News(2, 'Titolo
Notizia 2', 'Descrizione
```

```
notizia 2'));

this.listanews.push(new News(3, 'Titolo
Notizia 3', 'Descrizione

notizia 3'));

}

}
```

notizie/notizie.component.ts

Abbiamo sicuramente fatto un passo in avanti a livello di "pulizia" del codice, in

quanto ora News è un tipo di dati definito esternamente alla classe, e tutti gli usi

interni ai diversi componenti, rispetteranno

la stessa struttura di informazioni da

memorizzare, oltre che il tipo.

Nonostante questo, i valori delle notizie sono ancora presenti inseriti all'interno del

corpo della classe , e se volessi usare gli stessi valori per mostrarli in un altro

componente, dovrei manualmente

ricopiare le stesse righe.

Non preoccupiamoci per ora, perch é vedremo in seguito che la separazione dei dati

dal componente, si farà grazie alla creazione di quello che prende il nome di

"Service", che sarà iniettato all'occorrenza all'int

erno del componente che lo

richiederà.

7.2 Il modello dati per l'applicazione MetroChat

Sulla base delle nozioni viste, possiamo progettare anche il modello dati per

rappresentare gli oggetti che costituiranno il cuore dell'applicazione che intendiamo

progettare ossia l'oggetto

Metro, identificativo del treno in arrivo, e l'oggetto

Messaggio, identificativo del messaggio chat scambiato tra i diversi utenti presenti

all'interno di un treno.

Entrambi potrebbero essere salvati

all"interno della cartella *model* creata in

precedenza, sempre interna alla cartella *app*.

Una possibile struttura per l'oggetto

Metro, da memorizzare all"interno del file

metro.model.ts, potrebbe essere:

```
export class Metro {

constructor (

public idt: string,

public linea: string, // nome della linea
```

public numchatting: number, // numero passeggeri in chat

public tempo: number // orario partenza

) {}

}

model/metro.model.ts

Altri dati li aggiungeremo più avanti, quando vedremo come recuperare le

informazioni da sorgenti esterne via HTTP.

Una possibile struttura invece per l'oggetto Messaggio

- *messaggio.model.ts* -

potrebbe essere:

```
export class Messaggio {

constructor (

public idm: number, // identificativo del messaggio

public idt: string, // identificativo del treno
```

```
public idu: string, // identificativo
dell'utente

public testo: string, // testo del
messaggio

public idd?: string, // identif. del
destinat. (opzionale)

) {}

}
```

model/messaggio.model.ts

La notazione vista per il campo idd, la spiegheremo nelle prossime pagine.

7.3 Una classe senza costruttore

La notazione vista in precedenza, ci ha obbligati ad usare l'operatore new per creare

l'istanza di un nuovo oggetto da inserire all'interno di un array di tipo News.

Questo potrebbe essere oneroso a livello di programmazione, quindi quello che

spesso si fa, è definire una classe senza costruttore e sfruttare la possibilità offerta da

TypeScript, di inizializzare un oggetto assegnando un valore alle diverse proprietà.

Sulla base di queste considerazioni, il modello per rappresentare la notizia può essere

riscritto in questo modo:

```
export class News {

id: number;

titolo: string;

descrizione: string

}
```

model/news.model.ts

e per valorizzare l'array listanew, dovrai usare:

```
this.listanew.push({id:1,

titolo:'Titolo Notizia1',

descrizione:'Descrizione notizia 1'

});
```

Come puoi osservare, abbiamo aggiunto all"array direttamente un

oggetto

JavaScript, costruito sulla base della struttura del modello.

Questo perché la maggior parte de i servizi remoti che int errogherò, forniranno i dati

sotto forma di array di oggetti JSON.

La classe non ha definito alcun costruttore o metodo interno e non ha la necessità di

essere istanziata per essere usata. La si usa in sostanza , solo per definire un nuovo

tipo di dati, che sarà il tipo News.

La stessa cosa si potrebbe fare creando in tipo di dati News, nel seguente modo:

```
export type News = {

id: number;

titolo: string;

descrizione: string

}
```

model/news.model.ts

con la diffe renza che in questo caso non posso creare dei metodi interni per

manipolare le proprietà dell'oggetto.

Esercitazione: prova a trasformare la struttura dati *metro.model.ts*, in modo che non abbia il costruttore. Trasforma il componente *treni.component.ts*, in modo che la

proprietà listatreni, sia di tipo Metro.

7.4 Simulare dati remoti per semplici test

Per poter effettuare dei test locali al fine di apprendere le caratteristiche di Angular,

può essere utile spostare la valorizzazione degli array di dati, in

una classe esterna,

sotto forma di costante, che avrò cura di importare come se fosse un classico

componente.

Pertanto, sfruttando il modello dati creato nell'esercitazione richiesta, potre sti creare

il file *metroremoti.ts*, intern amente alla cartella di n ome *dati*, in cui inserire una costante da esportare di nome LISTAMETRO:

```
import { Metro } from
'./../model/metro.model';
```

```
export const LISTAMETRO: Metro[] =
[

{idt:'12345',linea:'Rossa', numpass:23,
tempo:12500},
```

(continua)

```
{idt:'67890',linea:'Verde', numpass:23,
tempo:145000},

{idt:'09876',linea:'Gialla', numpass:23,
tempo:165000}
```

];

dati/metroremoti.ts

Analogo discorso potrei fare per la lista dei messaggi associati ad una chat,

definendo una costante LISTAMSG, all'interno di un file *messaggiremoti.ts*.

```
import { Messaggio } from
'./../model/messaggio.model';

export const LISTAMSG: Messaggio[] =
[

{idm:1,idt:'12345',idu:'AAA',testo: '1°
msg Linea Rossa',idd: ''},
```

{idm:2,idt:'12345',idu:'BBB',testo: '2° msg Linea Rossa',idd: ''},

{idm:3,idt:'67890',idu:'CCC',testo: '3° msg Linea Verde',idd: ''},

];

dati/messaggiremoti.ts

Per poter usare queste costanti

- costituite da array di oggetti

Metro e

Messaggio- all'interno di un componente, dovrò scrivere la classica istruzione di

import, indicando il percorso relativo da cui prelevare il file:

```
import { LISTAMETRO } from
'./dati/metroremoti';

import { LISTAMSG } from
'./dati/messaggiremoti';
```

7.5 Classe con dati opzionali

Negli esempi precedenti abbiamo sempre ipotizzato che tutti i dati definiti nella

classe modello, fossero da inserire.

Nell"esempio qui sotto, la classe rappresentativa dell'oggetto News, è costituito da tre

dati obbligatori.

export class News {

```
id: number;

titolo: string;

desc: string

}
```

model/news.model.ts

Non sempre è così, nel senso che alcuni di questi potrebbero essere anche opzionali.

In TypeScript questi ultimi vengono

indicati usando l'operatore “punto di domanda ”

“?”, come nell'esempio qui sotto:

```
export class News {

id: number;

titolo: string;

desc: string;

datapubb?: string // proprietà della classe opzionale

}
```

model/news.model.ts

Il membro della classe datapubb, risulta così essere un parametro opzionale, e il

compilatore accetta che venga creata l'istanza della classe, anche in assenza di tale

parametro, come nell'esempio qui sotto:

```
import { Component } from '@angular/core';

import { News } from './model/news.model.ts';

@Component({

selector: 'ca-listanews',
```

```
template: `

<div class="news">

<ul>

<li *ngFor="let notizia of listanews">

<h2>{{notizia.titolo}}</h2>

<p>{{notizia.desc}}</p>

</li>

</ul>

</div> `

})
```

```
export class NotizieComponent {

// Array di oggetti News

listanews: News[] = [];

constructor() {

this.listanews = [{id:1,

titolo: 'Titolo Notizia 1',

descrizione: 'Descrizione notizia 1'}

];

}

}
```

notizie/notizie.component.ts

88

Capitolo 8

Interagire con il template e l’app: gli Eventi

8.1 Gestire il click o il tocco su un elemento del

template

Tra le operazioni che un utente comunemente fa all"interno di un'applicazione, c i

sono l'inserimento di dati tramite tastiera, i l click per cambiare pagina, accedere a

informazioni di dettaglio, e così via.

In Angular, ogni evento in grado di essere intercettato dal browser, può essere gestito

in modo molto simile a quello che avviene in JavaScript.

Se in JavaScript , per gestire i l click su di un elemento del DOM si utilizza la

seguente sintassi:

<taghtml onclick="azione()">...
</taghtml>

in Angular, si incastra il tipo di evento da monitorare e gestire, all'interno di

parentesi tonde ():

<taghtml **(click)**="azione()">...
</taghtml>

Come vedi la struttura è abbastanza simile a JavaScript. Cambia solo il nome

dell'evento del browser, che perde il prefisso "on".

La sintassi da usare in Angular, quindi sarà sempre del tipo:

89

(nomeEVENTO)="gestoreEvento()"

Il gestore d ell"evento, è una funzione che verrà richiamata non appena si verifica

quel particolare evento sull'elemento del DOM, evento in grado di essere intercettato

dal browser.

Nell'ipotesi in cui la vista o template del componente, sia rappresentata dal classico

campo di input e dal bottone per l'invio, per intercettare la pressione su quest'ultimo

potrei scrivere:

<button **(click)="inviaDato()"** >Invia</button>

Come vedi è stato inserito il nome dell'evento “click” da monitorare, all'interno di

parentesi tonde, e poi a destr a dell"uguale si è inserito il nome del

gestore/metodo

della classe che si prenderà cura di recuperare l'informazione.

Nel corpo della classe chiaramente dovrò definire il gestore d "evento, chiamato nel

nostro esempio inviaDato():

```
class MioComponente {

constructor() {

// azioni da eseguire all'inizializzazione del componente

}
```

```
inviaDato() {

// metodo per l'invio del dato

}

}
```

Il gestore d "evento può ricevere in ingresso un oggetto del DOM di tipo

event,

indicato con la notaz ione $event, che posso usare per accedere alle proprietà

specifiche che varieranno a seconda del tipo di evento generato in uscita dal

componente.

<button
(click)="inviaDato($event)">Invia</butt

All"interno del metodo inviaDato(),
posso sfruttare que sto evento per
accedere a

diverse proprietà:

90

export class MioComponente {

constructor() {

}

```
inviaDato(oggettoevento) {

// qui posso usare l'oggetto event e le
sue proprietà

alert(oggettoevento.currentTarget.name);

}

}
```

Qui sotto puoi vedere quelle più comuni legate all'evento click del mouse:

 currentTarget - Restituisce l'elemento su cui si è collegato il gestore di

evento

 target - Restituisce l'elemento effettivo su cui si è verificato l'evento

 target.name - Restituisce il nome dell'elemento del DOM si cui si è

verificato l'evento

 target.getAttribute("class") - Restituisce l'attributo class dell'elemento sui

cui si è verificato l'evento

 target.getAttribute("id") - Restituisce l'attributo id dell'elemento su cui si è

verifica l'evento

8.2 Gestire il click su un elemento del componente <ca-listnews>

Vediamo come sia possibile sfruttare questi concetti, per recuperare delle informazioni dal click fatto su un elemento del com ponente <ca-listnews> progettato in precedenza.

Lo scopo è quello di poter recuperare l'identificativo associato a ciascun articolo, al fine poi di visualizzare il dettaglio di

questo, prelev

ando ulteriori informazioni da

un" ipotetica sorgente esterna.

Il template del componente, potresti modificarlo nel seguente modo:

```
<div class="news">

<ul>

<li *ngFor="let notizia of listanews" (click)="dettaglio(notizia.id)">

<h2>{{notizia.titolo}}</h2>

<p>{{notizia.descrizione}}</p>
```

91

Come puoi osservare, ho aggiunto

ad ogni ta g <li>, un evento “click” da

monitorare, e un gestore d "evento, di nome dettaglio(), che riceve in ingresso

l'identificativo id recuperato dall'oggetto notizia.

Non ho usat o le doppie parentesi graffe per passare l' id della notizia al gestore di

evento, perché ci troviamo a destra del simbolo di uguale.

Il componente, completo di gestore d"evento, potrebbe quindi diventare:

```
import { Component } from '@angular/core';

import { News } from './../model/news.model';

@Component({

selector: 'ca-listanews',
```

```
template: `

<div class="news">

<ul>

<li *ngFor="let notizia of listanews"
click)="dettaglio(notizia.id)">

<h2>{{notizia.titolo}}</h2>

<p>{{notizia.descrizione}}</p>

</li>

</ul>

</div> `
```

```
})

export class NotizieComponent {

listanews: News[] = [];

constructor() {

this.listanews = [

{id:1, titolo: 'Titolo Notizia 1', desc:
'Descrizione notizia 1'}

];

}

// definisco il metodo dettaglio che
riceve un id notizia
```

```
dettaglio(id){

alert("Id Notizia: " + id);

}

}
```

notizie/notizie.component.ts

Non ho specificato il tipo di dato in ingresso alla funzione dettaglio, anc

he se è

buona norma aggiungerlo, per rendere il codice maggiormente leggibile ed essere

più "protetto" da eventuali errori.

Facendo il clic k su ogni notizia sia sopra al titolo che sopra alla descrizione,

visualizzerò un messaggio di alert con la scritta: "Id Notizia: X" dove al posto di X

sarà visualizzato il relativo id associato alla notizia cliccata.

8.3 Gestire gli eventi del mouse

Oltre al classico click, è possibile intercettare anche una serie di eventi legati al

mouse. Qui di seguito i più usati, chiaramente in ambito di applicazioni realizzate

per il web:

 mouseup - Emesso non appena si rilascia il pulsante del mouse dopo un click

 mousedown - Emesso non appena si schiaccia il pulsante del mouse sopra

all'elemento

 mousemove - Emesso non appena si muove il mouse sopra all'elemento

 mouseover - Emesso non appena si

posiziona il mouse sopra all'elemento o a

uno dei suoi figli

Una lista completa di tutti gli eventi del DOM la puoi trovare a questo link:

https://developer.mozilla.org/en-US/docs/Web/Events

Così come accade per l'evento

"click" del mouse, anche per questi è possibile

recuperare una serie di informazioni legate all'oggetto “evento”, passato al gestore

dell'evento, sotto forma di parametro dal nome $event.

Tra questi, oltre a quelli visti in precedenza, posso recuperare:

- *clientX* - recupera la coordinata X dove è posizionato il mouse al momento

del verificarsi dell'evento

- *clientY* - recupera la coordinata Y dove è posizionato il mouse al momento

del verificarsi dell'evento

Ad esempio, riprendendo l'applicazione di domotica creata nelle precedenti lezioni,

potrei sfruttare l'azione del

mousedown, per accendere o spegnere le lampadine

presenti nel template, andando a mod

ificare la proprietà stato di un o ggetto

Stanza.

Ora che sappiamo creare dei modelli di dati, potremmo aggiungere una struttura

simile a questa, di nome *stanza.model.ts*, da importare nel mio componente:

```
export class Stanza {
```

stanza: string;

stato: string;

luminosita: string;

}

model/stanza.model.ts

Riprendendo il codice del template

app-component.html visto nelle precedenti

lezioni, l'unica modifica da fare è l'inserimento d

i un ipotetico gestore di evento

onOff() che andrà a monitorare il *mousedown*:

```
<div class="lucistanza">

<ul>

<li *ngFor="let lucisingole of lucidb"

(mousedown)="onOff(lucisingole)" >

<span *ngIf="lucisingole.stato == 'OFF'; else showluce">

<i class="material-icons">lightbulb_outline</i>
```

{{lucisingole.stanza}}

<ng-template #showluce>

<i class="material-icons
on">lightbulb_outline</i>

{{lucisingole.stanza}}

</ng-template>

Al gestore d "evento, passo in ingresso proprio l'oggetto lucisingole, quindi nel

corpo del metodo, andrò a modificare il valore della proprietà stato, a seconda che

la luce sia nello stato di acceso “ON”, o nello stato di spento “OFF”.

```
onOff(lucestanza){

if(lucestanza.stato == 'OFF'){

lucestanza.stato = 'ON';

} else {

lucestanza.stato = 'OFF';

}
```

}

Ecco il codice completo del componente principale dell'app, che include la

definizione del modello di dati:

```
import { Component } from
'@angular/core';

import { Stanza } from
'./../model/stanza.model';

@Component({

selector: 'app-root',

templateUrl: './app.component.html',
```

styleUrls: ['./app.component.css']

})

(continua)

---

export class AppComponent {

show: boolean = true;

lucidb: Stanza[];

constructor() {

this.lucidb = [{

```
stanza: 'Luce Cucine',

stato: 'OFF',

luminosita: 5

},

{

stanza: 'Luce Sala Pranzo',

stato: 'ON',

luminosita: 2

},

{
```

```
stanza: 'Luce Bagno',

stato: 'OFF',

luminosita: 8

}];

}

onOff(stanza){

if(stanza.stato == 'OFF'){

stanza.stato = 'ON';

} else {

stanza.stato = 'OFF';
```

```
}

}

}
```

app.component.ts

8.4 Gestire gli eventi della tastiera

L'oggetto $event visto in precedenza, diventa particolarmente utile non appena si

debba gestire l'insieme d egli eventi legati all'input di dati effettuato tramite i classici

tag di un modulo web (input, textarea).

Parleremo in dettaglio di questi aspetti quando affronteremo la gestione dei moduli

web, in quanto il recupero delle informazioni da questi campi, avviene in mod

o

molto più semplice rispetto alle tecniche usate in JavaScript.

I tipici eventi che è possibile monitorare sono:

 keydown - si verifica non appena si preme un pulsante qualsiasi della tastiera

 keyup - si verifica non appena si rilascia un pulsante qualsiasi della tastiera

 keyenter - si verifica non appena si schiaccia il tasto *enter* della tastiera

Come sempre, dovrò sfruttare la notazione Angular per il collegamento di un evento,

grazie alle parentesi tonde:

<taghtml (eventotastiera)="gestore()">..
</taghtml>

Tra le informazioni che potresti avere la

necessità di recuperare, c"è quella legata al

tipo di tasto schiacciato. Basti pensare ad un gioco che dia la possibilità all"utente di

usare i tasti “freccia” per spostare un oggetto nello schermo.

Così come visto per gli eventi collegati al mouse, sfruttando l'oggetto

$event che

sarà di tipo KeyboardEvent, sono in grado di recuperare queste informazione e non

solo, come indicato qui sotto:

- *key* - il nome del tasto premuto

- *shiftKey* - valore booleano indicando se il tasto SHIFT (maiuscolo) è stato o

meno premuto

- *ctrlKey* - valore booleano indicando se il tasto CTRL è stato o meno premuto

- *altKey* - valore booleano indicando se il tasto ALT è stato o meno premuto

Nell'applicazione seguente, ho inserito un tag input, collegato all"evento *keyup*, e un

gestore d "evento a cui passerò l'oggetto

$event, grazie al quale riuscirò a

valorizzare un array che mi permetterà di visualizzare a video le diverse

informazioni legate all'oggetto passato

, memorizzate all"interno dell"array

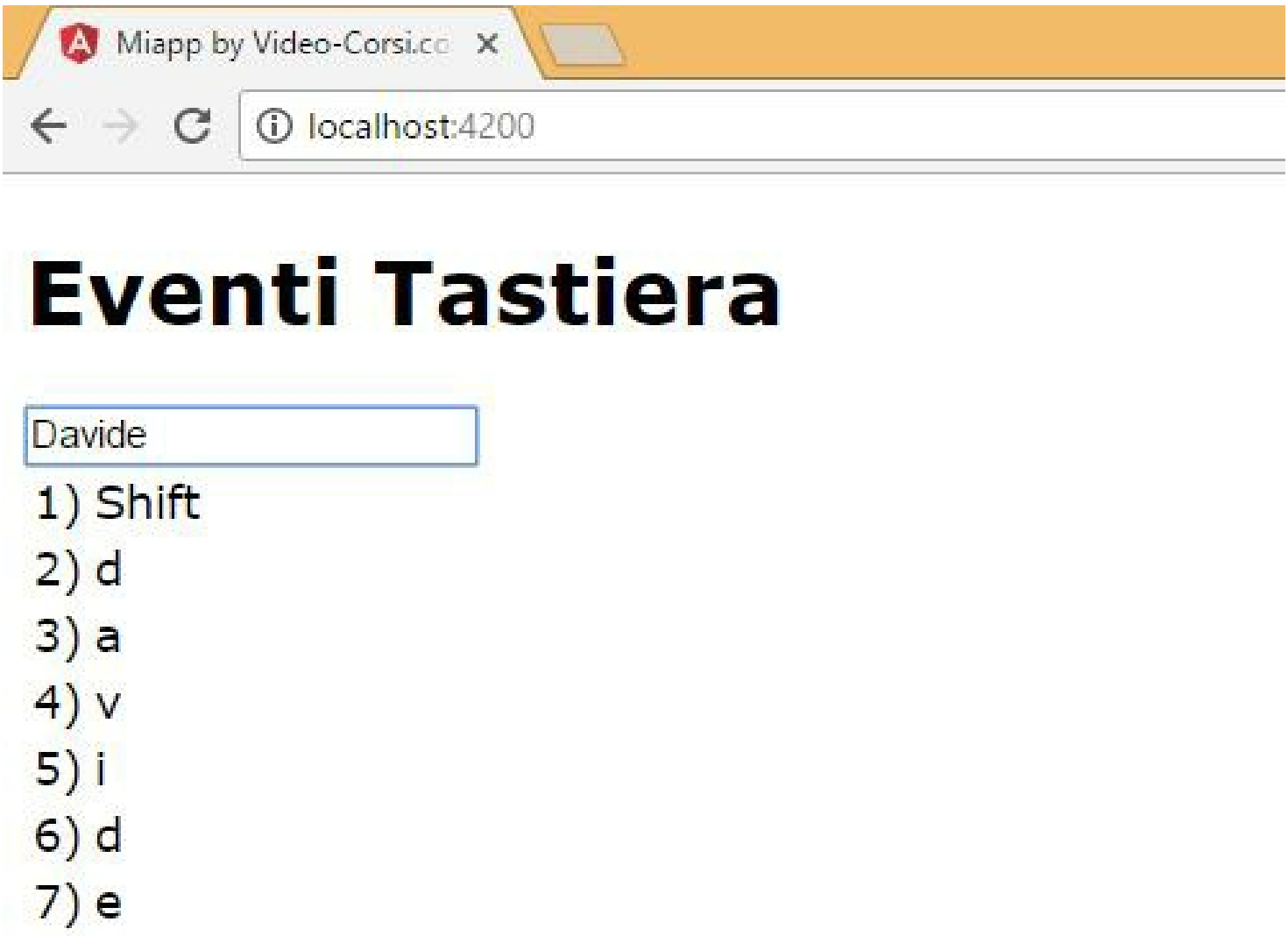
eventiTastiera:

```
import {Component} from
'@angular/core';

@Component({

selector: 'ca-tastiera',
```

```
template: `

<h1>Eventi Tastiera</h1>

<input
(keyup)="checkTasto($event)">

<table><thead></thead>

<tbody>

<tr *ngFor="let tastiera of
eventiTastiera, let i=index">

<td>{{i+1}})</td>

<td>{{tastiera.key}}</td>

</tr>
```

```
</tbody>

</table>

`
```

(continua)



```
96
})
export class TastieraComponent {
eventiTastiera:Array<Event>;
constructor() {
this.eventiTastiera = [];
}
checkTasto(event:KeyboardEvent) {
this.eventiTastiera.push(event);
```

}

}

tastiera/tastiera.component.ts

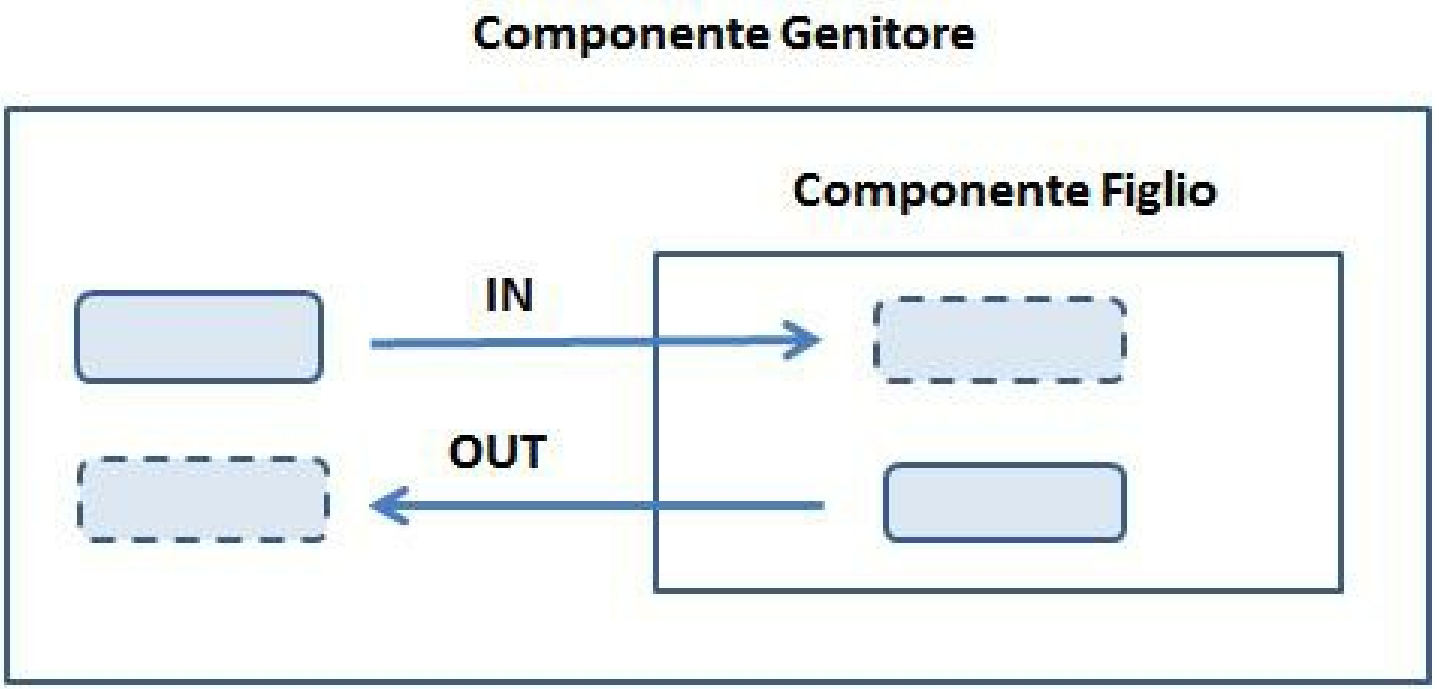
Se provassi a scrivere il mio nome Davide, otterrei:

Osserva come abbia usato una sintassi della direttiva *ngFor leggermente diversa da

quella base vista in precedenza. Grazie ad una seconda variabile i, valorizzata con la

variabile locale index specifica di *ngFor, posso mostrare un conteggio del

numero di volte in cui è ripetuto il ciclo.



Capitolo 9

Progettare Componenti “Intelligenti”

9.1 Cos’è un componente “intelligente”?

Fino ad ora abbiamo creato un'applicazione costituita da due soli componenti: quello

predefinito, che si crea non appena creiamo un nuovo progetto e quello

personalizzato, sviluppato per spiegare i diversi concetti del corso.

Abbiamo poi detto che per poter leggere il contenuto di una variabile definita nel

template o per scriverci dei

dati all"interno, devo sfruttare la tecnica del “Data

Bindigs” e queste operazioni sono

possibili solo all"interno dello stesso componente.

Nelle applicazioni Angular in realtà, avremo molti componenti e ognuno di questi

avrà la necessità di **comunicare co n gli altri** , sia ricevendo dei dati in ingresso

(relazione genitore ->figlio), che restituendo dei dati in uscita

(relazione figlio -

>genitore).

Questa tipologia di progettazione, offre un ulteriore vantaggio, ossia la possibilità di

riutilizzare alcuni co mponenti, in diverse sezioni dell"applicazione. Ad esempio,

l'applicazione che mostrava l'elenco completo delle notizie, potrebbe essere dotata di

altre sezioni, in grado di mostrare solo le notizie di una certa categoria.

In questo modo rendo il component e indipendente dai da ti, che invece spetterà al

genitore o padre, fornire. In sostanza vado a

ridurre le responsabilità del

componente, allineandomi al principio detto “

SRP” (Single Responsibility

Principle), ossia che un componente deve assolvere solo uno specifico compito.

Quelli che inizieremo a progettare nelle prossime pagine, saranno dei component

i

"intelligenti", ossia in grado di comunicare con altri componenti presenti nella vista

che funge da contenitore.

In particolare ci concentreremo su

lle tecn iche per r icevere dei dati in ingresso

proveniente dal padre, e sulle tecniche per comunicare a quest"ultimo il verificarsi di

un opportuno evento.

In questo modo, sarà possibile riutilizzare il componente in più sezioni

di una stessa

applicazione. E" un concetto che ha qualche similitudine con quello della classiche

“funzioni” presente in molti linguaggi di programmazione.

Nell'ipotesi tu abbia progettato due componenti

con selettore rispettivamente

<ca-menu> e <ca-news>, potresti creare dei template simili a questo:

<!--componente menu-->

<ca-menu></ca-menu>

<!--componente news ripetuto-->

<ca-news *ngFor="..." DATI-DA-INIETTARE></ca-news>

E' chiaro che ora dev i imparare il meccanismo con cui iniettare dei dati all'interno

del componente.

Applicando gli stessi concetti all'applicazione MetroChat, potre

sti definire il

componente <ca-metro>, rappresentativo del singolo treno, in questo modo:

<!--componente menu-->

<ca-menu></ca-menu>

<!--componente metro ripetuto-->

<ca-metro *ngFor="..." DATI-DA-INIETTARE></ca-metro>

9.2 Proprietà d’ingresso a un componente con

@Input

Uno dei meccanismi con cui è pos sibile

passare dei dati da un **componente padre** a

un componente

figlio, è quello di sfruttare il "Property Binding"

, ossia il

collegamento di una proprietà presente nel template del padre , con una proprietà

definita nel corpo della classe figlio, tramite il decoratore @Input().

Riprendendo l'applicazione MetroChat, potre

sti trasferire tutte le righe che

permettevano la visualizzazione del singolo treno, all'interno di un componente

dedicato, che divent

erebbe così

figlio del componente

<ca-treni>.

Come sele ttore potre sti usare <ca-metro> e aggiungergli la direttiva *ngFor, in

modo da ripeterlo a seconda di quanti

treni sono stati memorizzati nell'array definito

nel componente padre.

```
import { Metro } from './../model/metro.model';

import { Component, OnInit } from '@angular/core';

import { LISTAMETRO } from './../dati/metroremoti';

@Component({
```

(continua)

```
selector: 'ca-treni',

template: `

<div class=”listatreni”>

<ca-metro *ngFor = "let metro of
listametro" DATI-DA-INIETTARE>

</ca-metro>

</div>

`

})
```

```
export class TreniComponent
implements OnInit {

listametro: Metro[];

constructor() {

this.listametro = [];

}

ngOnInit() {

this.listametro = LISTAMETRO;

}

}
```

treni/treni.component.ts

Il componente con selettore <ca-metro>, viene ripetuto un certo numero di volte

sulla base del numero di elementi presenti all'interno dell'array listametro, che

per ora è stato valorizzato direttamente nel componente, con dei valori provvisori

impostati con la costante LISTAMETRO.

Le operazioni di inizializzazione

, sono state inserite all'interno del

metodo

ngOnInit(), che sappiamo viene eseguito una sola volta, dopo l'esecuzione del

costruttore.

Come faccio a passare in ingresso al componente

con selettore <ca-metro>,

l'oggetto da visualizzare?

Si usa la notazione con le doppie parentesi quadre:

[proprietàIN] = "proprietà_padre"

Nel nostro caso, potremmo inserire una proprietà di nome datiIn, valorizzata con

una proprietà del componente padre, ossia metro, che risulta essere una variabile

definita nel template .

[datiIn] = "metro"

Si dice che datiIn è il “target”, ed è una proprietà che non esiste nel compone nte

<ca-treni> ma solo nel componente <ca-metro>, mentre quello che c'è a destra

dell'uguale è la sorgente, ed è una proprietà del componente padre <ca-treni>.

```
import { Metro } from './../model/metro.model';

import { Component, OnInit } from '@angular/core';

import { LISTAMETRO } from './../dati/metroremoti';

@Component({

selector: 'ca-treni',

templateUrl: `
```

```
<div class=”listatreni”>

<ca-metro *ngFor = "let metro of
listametro" [datiIn]="metro" >

</ca-metro>

</div>

`


})

export class TreniComponent
implements OnInit {

listametro: Metro[];

constructor() {
```

this.listametro = [];

}

ngOnInit() {

this.listametro = LISTAMETRO;

}

}

treni/treni.component.ts

Nel componente figlio, per contrassegnare la proprietà datiIn come d "ingresso e

informare Angular che si tratta di una

proprietà pubblica accessibile dall'interno della

classe, si aggiunge il decoratore @Input(), a cui si fa seguire

il nome di una

proprietà che coincide con quella inserita all"interno del template del padre.

Tale decoratore, dovrà essere aggiunto nel corpo della classe figlio:

@Input() proprietàIN;

Pertanto il codice del componente <ca-metro>, potrebbe diventare:

import { Component, Input, OnInit} from '@angular/core';

import { Metro } from './../model/metro.model';

(continua)

@Component({

selector: 'ca-metro',

template: `

Linea: {{datiIn.linea}}

```
<p>Partenza: {{datiIn.tempo}}</p>

`


})

export class MetroComponent
implements OnInit {

// marchio la proprietà datiIn, come
d'Ingresso

@Input() datiIn: Metro;

constructor() {

}

ngOnInit() {
```

}

}

metro/metro.component.ts

Come vedi ho inserito la riga:

@Input() datiIn: Metro;

che permette ad Angular di recuperare l'oggetto Metro, valorizzato ad ogni ciclo nel

componente genitore.

Così facendo , la variabile datiIn potrà essere usa ta, sia nel corpo della classe del

componente con selettore <ca-metro>, sia nel template per accedere alle singole

proprietà dell'oggetto.

Osserva come all'interno della lista dei moduli da importare da @angular/core, sia

stato inserito Input, al fine di poter accedere al decoratore.

Non c'è un limite al numero di proprietà d''ingresso che è possibile definire, anche se,

si preferisce lavorare con oggetti proprio per avere un codice più pulito.

9.3 Definire più proprietà d’ingresso

Nel caso volessi inserire più proprietà, dovresti aggiungere più blocchi con parentesi

quadre. Nel template del componente padre potresti scrivere:

<ca-metro **[prop1]="valore1" [prop2]="valore2"** ></ca-metro>

mentre nella classe del componente figlio <ca-metro> dovresti scrivere:

export class MetroComponent

implements OnInit {

@Input() prop1;

@Input() prop2;

constructor() {

}

}

metro/metro.component.ts

Per il componente <ca-metro> allora, potre sti aggiungere un ulteriore parametro

d"ingresso rappresentato dall'ora

attuale, in modo che possa ess

ere usato per

calcolare il tempo mancante alla partenza del treno,

se già conosci l'orario di

partenza espresso in millisecondi a partire dalla data di Unix.

Pertanto se la proprietà now è valorizzata con:

this.now = new Date().getTime();

all'interno del template padre, potresti scrivere:

<ca-metro ... [datiIn]="metro"
[ora]="now" ></ca-metro>

e nel componente <ca-metro>,
aggiungere anche questa seconda
proprietà di

ingresso:

import { Metro } from
'./model/metro.model';

import { Component,Input, OnInit} from
'@angular/core';

@Component({

selector: 'ca-metro',

```
template: `<p>Linea: {{datiIn.linea}}
</p>

<p>Partenza: {{datiIn.tempo}}</p>

`


})
```

(continua)

```
export class MetroComponent
implements OnInit {

// marchio la proprietà datiIn, come
d'Ingresso
```

@Input() datiIn: Metro;

@Input() ora: number;

constructor() {

}

}

metro/metro.component.ts

che sfrutteremo più avanti per simulare un conto alla rovescia.

9.4 Cambiare il nome alla proprietà di ingresso

All'interno delle parentesi tonde del

decoratore @Input(), si può inserire qualcosa?

In alcuni casi sì, quando ad esempio hai la necessità di cambiare il nome associato

alla proprietà d"ingresso.

```
export class MetroComponent
implements OnInit {

@Input('propr1') nuovo_nome_prop1;

constructor() {

}

}
```

In questo caso, nel componente

<ca-metro> posso usare il nuovo nome della

proprietà e riferirmi a questa, sia nella classe che nel template.

9.5 Mostrare le informazioni, su un componente

dettaglio interno

Ora che sappiamo trasmettere delle informazioni da un componente padre ad un

componente figlio, potremmo aggiungere ulteriori funzionalità all'applicazione

MetroChat.

In particolare

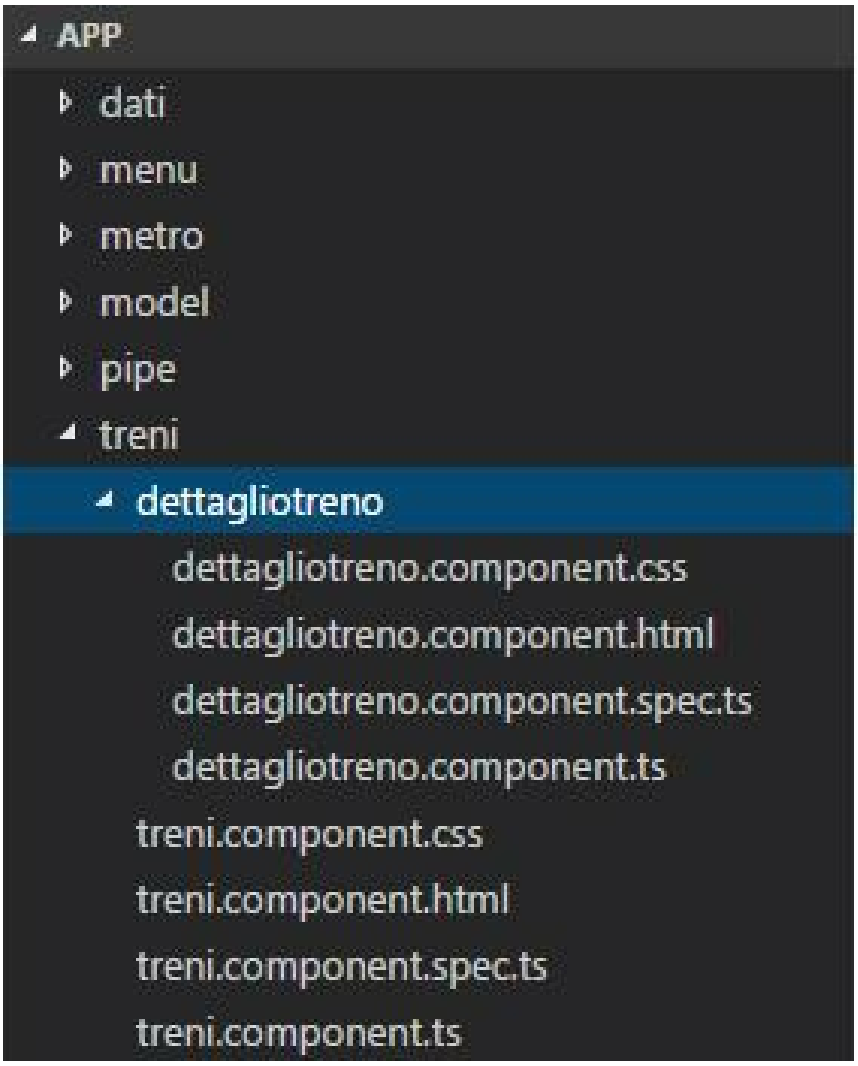
,

potremmo definire un componente

con selettore

<ca-dettaglio> all"interno della cartella *treni*, il cui scopo sarà quello di

mostrare delle informazioni specifiche di dettaglio.



Lo andrò a inserire nel template del componente Treni, proprio sotto alla lista dei

treni, e si attiverà non appena un utente cliccherà sul titolo della linea o in generale,

sul componente <ca-metro>

<ca-metro …></ca-metro>

<ca-dettaglio [treno]="trenoselezionato"></ca-dettaglio>

La proprietà

trenoselezionato, la dovrò valorizzare

chiaramente con un

opportuno gestore di event

o click fatto sul componente

<ca-metro>, mentre

treno, sar à la proprietà d "ingresso definita nel componente <ca-dettaglio> e

che potrò leggere se contrassegnata con il decoratore @Input().

<ca-metro *ngFor="let metro of listametro" [datiIn]="metro"

(click)="setMetro(metro)" >

</ca-metro>

Il metodo setTreno() non farà altro che valorizzare una proprietà della classe, con

il riferimento all'oggetto metro appena visualizzato nella vista/template:

```
setMetro(t) {

this.trenoselezionato = t;

}
```

Prima di svil uppare il codice del

componente dettaglio, mettiamo insieme i diversi

pezzi:

```
import { Metro } from './../model/metro.model';

import { Component, OnInit } from '@angular/core';

import { LISTAMETRO } from './../dati/metroremoti';

@Component({

selector: 'ca-treni',

template: `
```

<ca-metro *ngFor="let metro of
listametro"

[datiIn]="metro"

[ora]="now"

(click)="setMetro(metro)">

</ca-metro>

<!-- aggiungo questo componente di
dettaglio-->

<ca-dettaglio

```
[treno]="trenoselezionato"></ca-
dettaglio>

`


})

export class TreniComponent
implements OnInit {

listametro: Metro[];

trenoselezionato: Metro;

now:number;

constructor() {

this.listametro = [];
```

```
}

ngOnInit() {

this.listametro = LISTAMETRO;

}

setMetro(t) {

this.trenoselezionato = t;

}

}
```

treni/treni.component.ts

Il componente dettaglio invece, mostrerà

le diverse proprietà dell'oggetto ricevuto in

ingresso:

107

```
import { Metro } from
'./../../model/metro.model';

import { Component, Input } from
'@angular/core';

@Component({

selector: 'ca-dettaglio',

template: `
```

<div *ngIf="treno">

<h1>Dettaglio Treno</h1>

ID: {{treno.idt}}

Linea: {{treno.linea}}

<button (click)="chiudi">Chiudi dettaglio</button>

** **

`

})

export class DettagliotrenoComponent {

@Input() treno: Metro;

constructor() {

}

}

treni/dettagliotreno/dettagliotreno.comp

L'oggetto ricevuto in ingresso, in realtà, non è una copia dell'oggetto originario, ma è

proprio lo stesso oggetto visualizzato nel componente pad

re. Ogni eventuale

modifica alle sue proprietà, si rifletterà in automatico anche nella vista del

componente padre.

I più attenti avranno notato che ho aggiunto anche un tag div, con all'interno la

direttiva *ngIf. Questo perché, il componente <ca-dettaglio> è stato inserito

nel template del componente radice, quindi fa parte integrante della vista che viene

generata al caricamento dell'app.

Questa sezione deve essere visualizzata solo dopo che l'utente ha selezionato il treno

d"interesse, quindi solo dopo aver valorizzato la variabile trenoselezionato, che

viene poi collegata alla proprietà d"ingresso del componente figlio.

In assenza della direttiva *ngIf, verrebbe segnalato un errore, perché tale proprietà

è indefinita al caricamento del componente genitore.

Avrei potuto inizializzare la variabile

treno nel costruttore, ma questo

implicherebbe che la sezione dettaglio sarebbe visualizzata sempre, anche in assenza

di treno selezionato, a meno che non metta di nuovo una condizione.

Esercitazione proposta

Progettare il metodo “chiudi” per far scomparire la visualizzazione del dettaglio non

appena l"utente clicca sul pulsante chiudi.

9.6 Definire proprietà di uscita con @Output()

Il passo che ci rimane da fare per creare un componente intelligente a 360 gradi

, è

capire come si possano intercettare dei

dati in uscita. In gergo si dice "

iscriversi"

alle proprietà di uscita, con una terminologia che penso ti sia familiare se sai cosa

sono gli *Observable*.

Noi sappiamo infatti che tutti i membri di un componente so

no visibili solo

all'interno di questo, quindi per fare in modo che un membro interno possa essere

trasmesso anche al componente padre, dovrò "marchiarlo" in modo che Angular lo

consideri affidabile e lo possa usare al proprio interno, sia nella classe che

nel

template.

In analogia a quello fatto per una proprietà d "ingresso, per definire una proprietà di

uscita, la si "marchierà" con @Output():

@Output() proprietàOut

Questa sintassi per ò non è completa, in quanto abbiamo detto che è necessario

iscriversi alla proprietà , ossia trasformare il dato in uscita, in un evento da

monitorare. Ad ogni emissione d'evento

da parte del componente figlio

, il

corrispondente dato associato, verrà offerto al componente padre.

La sintassi completa da usare sarà:

@Output() proprietàOut = **new**

EventEmitter<tipo>()

dove ho usato la classe EventEmitter(), che ci permette di creare degli eventi

personalizzati. A questo punto proprietàOut è a tutti gli effetti un oggetto *evento*,

in grado di **emettere degli eventi ,** a cui posso associare dei valori in base al tipo

specificato nell'EventEmitter.

A livello di componente padre, sappiamo già come si collega un gestore d"evento,

quindi sfrutteremo la classica notazione delle parentesi tonde:


```
<tagHtml …
(proprietàOut)="azionePadre($event)">
</taghtml>
```

Al posto del classico evento “click”, inseriremo il

nome della proprietà in uscita

definita nel componente figlio. Non appena proprietàOut emetterà l"evento con

emit:

proprietàOut.emit(valore_emesso); // il figlio comunica al padre

questo verrà intercettato dal padre, che eseguire l"azione prevista (azionePadre),

recuperando i dati emessi grazie all'oggetto $event.

Applicando questi concetti ai due componenti

<app-root> e <ca-metro>, e

sfruttando il codice sviluppato per il pipe personalizzato, potrò scrivere:

import { Metro } from

'./../model/metro.model';

import { Component, Input, Output, OnInit, EventEmitter} from '@angular/core';

@Component({

selector: 'ca-metro',

template: `

<div [ngStyle]="stato">

Linea: {{datiIn.linea}}

Partenza: {{datiIn.tempo}}

{{attesa | mmss}}

```
<hr/>

</div>`

})

export class MetroComponent
implements OnInit {

@Input() datiIn: Metro;

@Input() ora: number;

// marchio la proprietà inpartenza, come
d'Uscita

@Output() inPartenza = new
EventEmitter<string>();
```

```
stato:Object;

orapartenza: number;

attesa:number;

constructor() { }

ngOnInit() {

// in millesecondi da Unix time

this.orapartenza = this.datiIn.tempo;

// tempo mancante alla partenza

this.attesa = this.orapartenza-this.ora;

let x = setInterval(() => {
```

this.attesa-=1000;

if (this.attesa<=0) {

(continua)

110

// blocco il timer e mando l'evento in uscita

clearInterval(x);

// notifico il cambio di dato passando l'id del treno

this.inPartenza.emit(this.datiIn.idt);

```
// modifico lo stato di visualizzazione
del componente

this.stato = {'display':'none'};

}

},1000);

}

}
```

metro/metro.component.ts

Come vedi, ho aggiunta la proprietà di uscita con nome inPartenza, assegnandogli

l'istanza della classe EventEmitter, in modo da trasformarla in un oggetto in grado

di emettere degli eventi con dei dati associati.

@Output() inPartenza = new EventEmitter<string>();

Il dato da emettere è l'identificativo del treno, che mi servirà per elencare, nel

componente padre, tutti i treni partiti:

this.inPartenza.emit(this.datiIn.idt);

Al fine di rendere il tutto più interessante, ho inserito questa riga

all'interno di una

funzione setInterval(), che decrementa di 1000 la proprietà

attesa di ogni

oggetto Metro, in modo da simulare il trascorrere del tempo.

Ti ricordo che la proprietà tempo è valorizzata con l''ora di partenza espressa in

millisecondi. Questo il motivo per cui ho decrementato di 1000.

Ogni componente <ca-metro> avrà un conteggio separato dagli altri, e non

appe na

si arriverà a 0, verrà bloccato il conteggio ed emesso il valore in uscita.

Nota come abbia usato il pipe creato in precedenza - *mmss.pipe.ts* - per formattare il

tempo d'attesa, in minuti e secondi:

{{ attesa | mmss }}

111

Oltre a questo , ho aggiunto una direttiva ngStyle per cambiare la visibilità

dell'elemento, impostando la proprietà

CSS *display* a none.

Per poter intercettare il valore in uscita dal componente figlio

<ca-metro>, dovrò

inserire un gestore dell"evento “inpartenza” direttamente nel template del padre:

{{trenipartiti}}

// mostro i treni partiti

<ca-metro *ngFor="let metro of listametro"

(inPartenza)="partiti($event)"

[datiIn]="metro"

[ora]="now"

(click)="setMetro(metro)">

</ca-metro>

<!-- aggiungo questo componente di dettaglio-->

<ca-dettaglio [treno]="trenoselezionato"></ca-dettaglio>

dove inPartenza è proprio la proprietà contrassegnata nel componente figlio e

inserita come se fosse un evento pe

rsonalizzato da monitorare. Il valore emesso,

verrà gestito come dato in ingresso al metodo partiti(), il quale si preoccuper à

solo di valorizzare la proprietà trenipartiti, aggiungendo via via la stringa id di

ogni treno.

partiti(id:string) {

```
this.trenipartiti += "|" + id;

}
```

Il codice, completo delle varie inizializzazioni presenti nel costruttore, diventa:

```
import { Metro } from
'./../model/metro.model';

import { Component, OnInit } from
'@angular/core';

import { LISTAMETRO } from
'./../dati/metroremoti';

@Component({
```

selector: 'ca-treni',

template: `

{{trenipartiti}}

<ca-metro *ngFor="let metro of listametro"

(inPartenza)="partiti($event)"

(continua)

[datiIn]="metro"

```
[ora]="now"

(click)="setMetro(metro)">

</ca-metro>

</div>

<!-- aggiungo questo componente di
dettaglio-->

<ca-dettaglio
[treno]="trenoselezionato"></ca-
dettaglio>

`


})
```

```
export class TreniComponent implement
OnInit {

listametro: Metro[];

trenoselezionato: Metro;

trenipartiti:string;

now:number;

constructor() {

this.trenipartiti = '';

this.listametro = [];

this.now = new Date().getTime();
```

```
}

ngOnInit() {

this.listametro = LISTAMETRO;

}

setMetro(t) {

this.trenoselezionato = t;

}

partiti(id:string) {

this.trenipartiti += "|" + id;

}
```

}

treni/treni.component.ts

9.7 Accedere a membri di altri componenti con

@ViewChild

Abbiamo più volte detto che i membri interni ad un componente (proprietà e metodi)

sono accessibili solo dall'interno di questo, a meno che non applichi le tecniche viste

nei paragrafi precedenti.

Quindi un qualsiasi componente genitore, senza opportune tecniche, non potrà mai

accedere a proprietà o metodi interni ad un componente figlio.

E' tuttavia possibile, con opportuni decoratori, accedere alla classe di un componente

figlio, direttamente dal contenitore padre.

Si sfrutta il decoratore @ViewChild, con la seguente sintassi:

@ViewChild(cComponenteFiglio)
proprietà: ComponenteFiglio;

Il decoratore @ViewChild() è una funzione che riceve in ingresso il nome del

componente figlio e trova il corrispondente selettore presente nel

template del

componente padre (per l"esattezza il primo, nel caso ce ne siano più di uno e siano

uguali).

In questo modo , nel componente padre

possiamo usare un"istanza del componente

figlio che ci permetterà di accedere ai metodi o alle proprietà interne alla classe.

Ad esempio, potresti creare un componente figlio di nome *popup.component.ts*, il cui

scopo è quello di mostrare un"ipotetica scritta con un bottone per nasconderla.

Nel template del componente padre, potresti richiamarlo così:

<ca-popup>Messaggio in PopUp</ca-popup>

mentre il corpo del componente, potrebbe essere sviluppato in questo modo:

```
import { Component } from '@angular/core';

@Component({

selector: 'ca-popup',

template: `

<div *ngIf="mostramsg">

<ng-content></ng-content>

<button (click)="nascondi()">OK</button>
```

```
</div>

`

})

export class PopupComponent {

mostramsg = false;

mostra() {

this.mostramsg = true;

}
```

(continua)

```
nascondi() {

this.mostramsg = false;

}

}
```

popup.component.ts

Come puoi osservare il metodo mostra(), appartiene al componente e agisce sulla

proprietà di nome mostramsg del componente stesso.

Potresti però avere la necessità di acce

dere a questo metodo dal

componente

contenitore - o padre -, al fine di far apparire la scritta , dopo il click su un elemento

della vista del componente padre.

Per fare questo, nel componente

genitore dovrai inserire il decoratore

@ViewChild(), “iniettando” l''istanza del componente figlio all'interno di una

proprietà.

Nell"ipotesi quest"ultima abbia nome msg, scriverò:

@ViewChild(PopupComponent) msg: PopupComponent;

La proprietà msg, a desso, potrai usarla per accedere direttamente al metodo

mostra() del componente figlio:

this.msg.mostra();

I più attenti di voi, avranno notato la presenza di un tag non descritto in precedenza:

<ng-content></ng-content>

A cosa serve?

Serve per visualizzare all"interno di un componente figlio, del

contenuto inserito nel template del componente padre. Tale contenuto deve apparire

all"interno del tag html rappresentativo del componente:

<ca-popup>Messaggio in PopUp</ca-popup>

Nel nostro caso, sopra al bottone di “OK” , verrà visualizzata la scritta : “Messaggio

in popup”, perché è qui che ho inserito il tag <ng-template>.

Avrei potuta ometterlo e scriver e il testo direttamente all"interno del template del

componente <ca-popup>, ma ho colto l"occasione al volo per introdurre anche

questo concetto di Angular.

Il codice completo del componente padre diventa:

```
import { Component, ViewChild } from
'@angular/core';

import { PopupComponent } from
'./popup.component';

@Component({

selector: 'app-root',

template: `

<ca-popup>Messaggio in popup</ca-
popup>

<button (click)="mostroMsg()">Mostro
PopUp</button>

`
```

```
})

export class AppComponent {

@ViewChild(PopupComponent) msg:
PopupComponent;

mostroMsg() {

this.msg.mostra();

}

}
```

app.component.ts

Così come fatto per @Input() e @Output(), ricordati d"importare i

riferimenti al

decoratore @ViewChild() dalla libreria principale di Angular.

9.8 Accedere a membri di un componente figlio

direttamente dal template

Nell'esempio precedente, siamo riusciti ad accedere a

i membri di un componente

figlio.

Angular offre un"ulteriore possibilità, ossia quella di usare una **variabile locale** nel

template, che ci permette di accedere alla proprietà del componente a cui è collegata,

sfruttando la notazione del cancelletto #.

<taghtml #nomevariabile></taghtml>

E' una tecnic a che vedremo anche quando parleremo di form. La differenza rispetto

alla tecnica vista in precedenza, è che posso riferirmi a proprietà e metodi del

componente figlio, solo nel template del componente padre e non nella classe.

Se all'interno del componente figlio <ca-popup> aggiungessi la variabile locale

con

nome #pp, potrei accedere ai metodi e alle proprietà definite internamente

, con la

notazione:

{{pp.mostramsg}} // accedo alla proprietà mostramsg

(click) = "pp.mostra()" // accedo al metodo mostra

Pertanto potrei progettare il componente padre, in questo modo:

import { Component } from

```
'@angular/core';

import { PopupComponent } from
'./popup.component';

@Component({

selector: 'app-root',

template: `

<ca-popup #pp>Messaggio in
popup</ca-popup>

<button (click)="mostroMsg()">Mostro
PopUp</button>

<p>Proprietà del componente figlio
{{pp.mostramsg}}</p>
```

`

})

export class AppComponent {

mostroMsg() {

this.msg.mostra();

}

}

app.component.ts

Tutte le volte che la propr

ietà mostramsg verrà modificata

all'interno del

componente figlio, automaticamente il valore

sarà aggiornato nel template del

componente padre. Questa tecnica può essere usata in combinata con il decoratore

@ViewChild() per r eferenziare singoli elementi de l template (es. bottoni). Sarà

sufficiente inserire come argomento del decoratore proprio il nome della proprietà

scelta per identificare il tag html

- es. @ViewChild('closebtn')

btn:ElementRef – e accedendo nel corpo della classe con la notazione

this.btn.nativeElement.

La navigazione in Angular

10.1 Il concetto di Routing e Route

La gestione della transizione tra

diversi stati di un"applicazione, è uno dei più

importanti passi di progettazione che devi affrontare perché coinvolge sia un

aggiornamento dell"url, che dello stato dell"applicazione.

La navigazione tra pagine di un sito

web, è un concetto che penso tu conosca molto

bene. Non appena inserisci un tag HTML di questo tipo:

Link alla pagina X

il navigatore è in grado di prelevare dal server la paginaX e visualizzarla all'interno

del browser. Il browser dal canto suo, mostrerà nella barra degli url, l'indirizzo

completo della pagina, in relazione al dominio che si sta navigando, in modo che

l'utente possa copiare l'indirizzo per memorizzarlo nei preferiti, inviarlo nei propri

canali social, inserirlo in messaggi email etc.

Questo chiaramente nell'ipotesi il mio sito non sia una

Single Page Application

(SPA), ossia una singola pagina.

Ogni pagina quindi è c aratterizzata da un percorso fisico (url) ben preciso. Tipici url

potrebbero essere:

www.miosito.com/

/paginaX.htm

/contattaci.php

/catalogo/sedie.php

Nel caso di An gular sappiamo che il concetto di pagina non esiste, ma esiste il

concetto di **vista o template**.

Essendo le viste tutte interne all'applicazione, non ci saranno chiamate al server,

come avviene per un classico sito web. Questo permette un caricamento sicuramente

più rapido di ogni sezione di un"applicazione o sito.

Tutti gli attuali browser supportano infatti la navigazione tra la cronologia degli url

visitati dall"utente, questo grazie all"oggetto history e a dei metodi come

“history.pushState”. Quest"ultimo ci permette di andare avanti e indietro

sfruttando le classiche frecce di navigazione presenti in ogni browser.

Possiamo così

cambiare l"url (es. sfruttando JavaScript), senza inviare una richiesta al server.

In Angular, quindi, per implementare un siste ma di navigazione, non è sufficiente

inserire un link come si fa per le classiche pagine web, ma è necessario sudare un po"

di più e sfruttare quello che si chiama in gergo tecnico **Router**.

Quest'ultima è una libreria opzionale

, costituita da una combinazi one di

diversi

"attori" che impareremo a conoscere via via nel corso delle prossime pagine.

Il meccanismo di navigazione tra le diverse *viste*, viene chiamato **Routing** e consiste nel predisporre un insieme di stringhe rappresentative del singolo indirizzo url a cui

si vuole accedere, con associato un componente da visualizzare . Ognuna di queste

stringhe viene chiamata **oggetto Route** o semplicemente Route.

 Routing > La navigazione in Angular

intesa come passaggio da uno stato

all'altro

 Route > Oggetto definito con due proprietà: una che

identifica l''url e la

seconda che identifica il componente con la vista da visualizzare

NB: Noi invece di usare la traduzione dall'inglese di *Route*, ossia "percorso,

indirizzo", useremo semplicemente il termine route.

Pertanto i l processo di configurazione o

Routing in un'applicazione Angular, si

effettua andando a configurare un Router con la lista delle Route, in modo che in

corrispondenza a lla navigazione ai diversi url definiti, il Router crei l"istanza del

componente associato e mostri la relativa vista.

Questa configurazione è possibile solo se sfruttiamo un metodo presente nel modulo

RouterModule, che dovremo quindi importare all"interno dell" *app.module.ts*

Il punto di partenza è lavorare all'interno del file *app.module.ts*, e aggiungere questo

elemento.

10.2 Configurare il file AppModule

Abbiamo già detto in precedenza che il punto di partenza per l'introduzione di nuove

funzionalità per un'applicazione Angular è il modulo radice, che per convenzione è

chiamato *AppModule*.

Per gestire la navigazione, è necessario importare : 1) il modulo RouterModule

dalla libreria

@angular/router, che ci permetterà di sfruttare il metodo

RouterModule.forRoot() per impostare l

"array di route definit e per

l'applicazione e 2) Routes per permet tere ad Angular di riconoscere

l"array di

oggetti che creeremo e che identificherà l"insieme delle diverse route.

import { RouterModule, Routes } from '@angular/router';

All'interno dell'array della proprietà

import di NgModule, dovremo richiamare il

metodo forRoot(), per definire un modulo, contenente il *Router* configurato:

import: [

BrowserModule,

RouterModule.forRoot(LISTA_ROUTES

]

dove LISTA_ROUTES, vedremo dopo, sarà un array di tipo Routes, con all"interno

le divere route

in grado d

i mostrare una particolare vista

associata ad un

componente.

Il codice completo dell'AppModule dell"applicazione MetroChat, pertanto diventerà:

```
import { NgModule } from '@angular/core';

import { BrowserModule } from '@angular/platform-browser';
```

import { RouterModule, Routes } from '@angular/router';

...

@NgModule({

imports: [

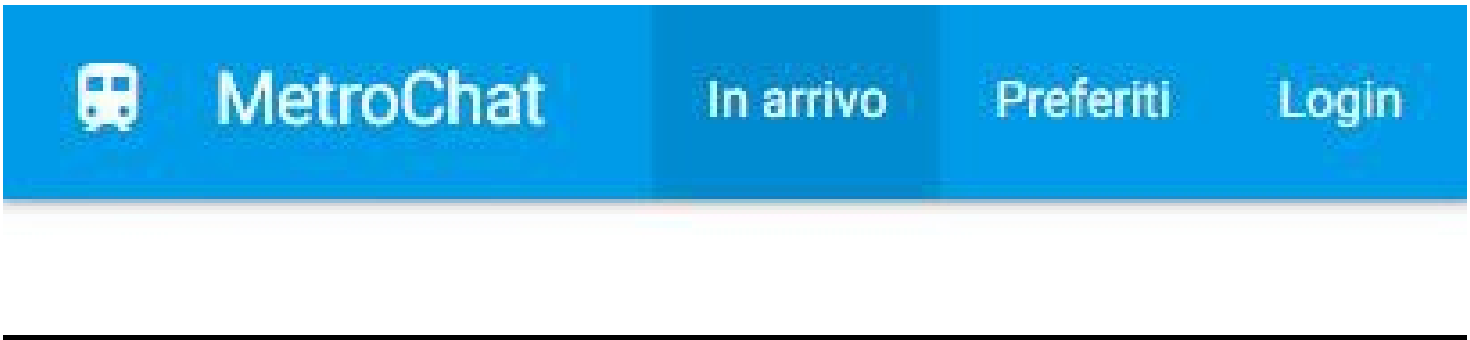
BrowserModule,

RouterModule.forRoot(LISTA_ROUT

],

declarations: [MiaAppComponent, MenuComponent, ...],

(continua)




```
bootstrap: [ MiaAppComponent ]
})
export class AppModule {}
```

app.module.ts

Questa tecnica viene di solito usata quando non si hanno tante

route da definire.

Diversamente, al fine di non creare dei file AppModule troppo ricchi di codice, si

dovrà creare un modulo separato da importare, cosa che per ora non faremo.

Il secondo passo da fare è definire le Routes.

10.3 Progettare l’array di route

L'applicazione vista fino ad ora prevedeva un componente <ca-menu> con definiti

all'interno i diversi link per accedere alle ipotetiche pagine.

Nel caso dell'applicazione MetroChat, i link identificativi delle diverse viste

potrebbero essere:

- www.miosito.com/
- www.miosito.com/preferiti
- www.miosito.com/login

Come faccio a dire ad Angular quale vista mostrare e che indirizzo associargli?

Si definisce un array c he conterrà la lista di

configurazioni per ogni route, dove

ognuna di queste, sarà un oggetto literal JavaScript con un insieme di proprietà:

// singola route per mostrare la pagina di ingresso

{ path: '', component: TreniComponent }

Come vedi , ho inserito due proprietà per l'oggetto JavaScript, l e proprietà path e

component. La prima identifica il percorso relativo al dominio principale che dovrà

essere visualizzato nel browser, mentre la seconda, il nome scelto per identificare la

classe del componente che dovrà essere visualizzato in termini di vista o template.

121

Il nome del la classe del componente NON deve comparire racchiusa tra virgolette e

chiaramente deve esistere all'interno dell'applicazione.

Un controllo che ti invito a fare spesso è verificare che il componente associato alla

route sia stato importato

, ossia inserito all'interno dell'array della proprietà

declarations di @ngModule().

Esistono diverse altre proprietà che è possibile specific are ma, per ora, limitiamoci a

queste. La configurazione delle

route per l "applicazione MetroChat, assumerà la

forma di un array di tipo Routes:

[

{ path: '', component: TreniComponent

},

{ path: 'preferiti', component:
PreferitiComponent },

{ path: 'login', component:
LoginComponent }

]

Ho presupposto che l'applicazione sia costituita da due componenti aggiuntivi

rispetto a quello principale, e i tre percorsi individuati, corrispondano proprio agli url

definiti all'inizio. Ho creato il mio primo array di route.

Questo array costituisce proprio l'argomento del metodo forRoute(), che abbiamo

inserito nell'AppModule.

Pertanto la successiva modifica da fare, sarà aggiungere tali righe all'interno del file

AppModule:

```
import { NgModule } from
'@angular/core';

import { BrowserModule } from
'@angular/platform-browser';

import { RouterModule, Routes } from
```

'@angular/router';

import { AppComponent } from
'./app.component';

import { MenuComponent } from
'./menu/menu.component';

**import { TreniComponent } from
'./treni/treni.component';**

**import { LoginComponent } from
'./login/login.component';**

**import { PreferitiComponent } from
'./preferiti/preferiti.component';**

@NgModule({

imports: [

BrowserModule,

RouterModule.forRoot([

{ path: '', component: TreniComponent },

{ path: 'preferiti', component: PreferitiComponent },

{ path: 'login', component: LoginComponent }])

],

(continua)

declarations: [AppComponent, MenuComponent, **TreniComponent,**

PreferitiComponent, LoginComponent],

bootstrap: [AppComponent]

})

export class AppModule {}

app.module.ts

In questo modo, non appena l'utente inserirà nell'url un link corrispondente a quelli

visti all'inizio, Angular troverà una c
orrispondenza con i percorsi indicati nel file e

sarà in grado di visualizzare la vista del relativo componente.

La domanda che potresti farti è: ma dove verrà visualizzato il template del

componente? Un componente , abbiamo sempre detto , è solo un piccolo

pezzo

dell'interfaccia di un'applicazione.

Quest"ultima avrà sempre un componente principale, avviato al

caricamento dell'app,

tramite il file *index.html*.

Con riferimento al

la nostra applicazione

, il primo componente caricato

è

AppComponent, che ha questo template:

<ca-menu><ca-menu>

<h1>MetroChat</h1>

...

Fino ad ora, sotto la scritta MetroChat , ho aggiunto all"occorrenza il selettore del

template in fase di progettazione, nell"ipotesi l"applicazione avesse solo una sezione.

Nel caso di più sezioni, d evo trovare un sistema in grado di far visualizzare sotto al

menu, il componente

TreniComponent

o

PreferitiComponent

o

LoginComponent, sempre usando lo stesso template del componente radice.

10.4 Indicare dove visualizzare le route con

RouterOutlet

Per ind icare ad Angular, dove visualizzare il componente associato ad una

particolare route, ci viene in aiuto una direttiva fornita da

RouterModule.

Quest"ultimo è già stato inserito in precedenza

nel file AppModule quindi è

disponibile per ogni componente dell'app.

La direttiva si chiama <router-outlet> e s i inserisce direttamente nel template

del componente radice, come una sorta di "segnaposto" per tutta l"applicazione:

<ca-menu><ca-menu>

<h1>MetroChat</h1>

<router-outlet></router-outlet>

Ora Angular sa prà che **sotto** al tag <router-outlet>, dovrà aggiungere il

template del componente che corrisponde all"url di volta in volta

presente nella barra

degli indirizzi del browser.

Nel caso dell"applicazione MetroChat, avremo queste corrispondenze:

1) **Url: www.miosit o.com** - Corrispondenza con il path vuoto ", quindi mostro

TreniComponent

<ca-menu><ca-menu>

<h1>MetroChat</h1>

<router-outlet></router-outlet>

<!-- qui sotto Angular mostrerà il template di TreniComponent-->

2) **Url: www.miosito.com/preferiti** - Corrispondenza con il path 'preferiti', quindi

mostro PreferitiComponent

<ca-menu><ca-menu>

<h1>MetroChat</h1>

<router-outlet></router-outlet>

<!-- qui sotto Angular mostrerà il template di PreferitiComponent-->

3) **Url: www.miosito.com/login** -

Corrispondenza con il path 'login', quindi mostro

LoginComponent

<ca-menu><ca-menu>

<h1>MetroChat</h1>

<router-outlet></router-outlet>

<!-- qui sotto Angular mostrerà il template di LoginComponent-->

Ora, presi dall'entusiasmo , potremmo testare l'applicazione e cliccare s

ui diversi

menu del componente <ca-menu>:


```
<nav>

<li><a href="">In arrivo</a></li>

<li><a href="/preferiti">Preferiti</a>
</li>

<li><a href="/login">Login</a></li>

</nav>
```

Ahimè non accadrebbe nulla, questo perch é in Angular, per formare un link in grado

di comunicare con il Router è necessario utilizzare una direttiva, sempre fornita da

RouterModule. E" la direttiva RouterLink.

Come puoi constatare, le cose sono leggermente più complesse di quelle che s

ei

abituato a fare con le classiche pagine web.

A questo punto, ecco altre domande che potresti farti:

 Come faccio a creare dei link per accedere alle diverse route?

 Posso aggiungere più <router-outlet> nel template?

 Cosa succede se non viene trovata alcuna corrispondenza tra url e route

definite?

 Come posso reindirizzare il navigatore ad un'altra vista/pagina?

 Come posso aggiungere dei parametri query all'url?

Tutte cose a cui risponderemo nelle prossime pagine.

10.5 Configurare i link nel template con

RouterLink

Per spostarci tra le diverse route, a bbiamo bisogno di un menu. Questo menu è già

stato progettato ma abbiamo visto che non funziona, nel senso che non ci permette di

spostarci su una nuova pagina o vista.

Vediamo allora che modifiche apportare , al fine di farlo funzionare. Abbiamo detto

che è necessario sfruttare la direttiva RouterLink.

Ecco la sintassi base:

<a routerLink="PERCORSO">Link alla Route

125

Quando parleremo di link dinamici, vedremo come sfruttare un array al posto d

i

PERCORSO. Come puoi osservare, è stato sostituito al class ico href, il nome della

direttiva.

Pertanto il componente <ca-menu>, potrebbe diventare:

```
<nav>

<li><a routerLink="">In arrivo</a>
</li>

<li><a
routerLink="/preferiti">Preferiti</a>
</li>

<li><a routerLink="/login">Login</a>
</li>

</nav>
```

Ricordati che questi url sono fittizi, nel senso che non fanno riferimento a pagine

reali sul server, ma a percorsi relativi alla stessa app. Questo è il motivo per cui nel

file *index.html* è presente il tag HTML <base href="/">

Avendo anteposto ad ogni url il sim

bolo /, sono sicuro che il percorso fa

sempre

riferimento alla radice dell'app.

Se invece il link si trova all'interno di un componente visualizzato accedendo ad una

route diversa dalla radice dell'app, varranno sempre le regole:

 / faccio sempre riferimento alla radice dell'app

 ./ o nullo, faccio riferimento allo stesso livello del componente

 ../ salgo di un livello rispetto alla posizione attuale

Fortunatamente ce la siamo cavata con una modifica di poco conto, andando a

sostituire a href, la nuova diretti

va. Nella realtà, quest'ultima offre diverse

funzionalità che impareremo a sfruttare meglio quando parleremo di

parametri

dinamici o url con query.

A questo punto se provassi a cliccare su

i diversi link, finalmente riuscire sti a

visualizzare le diverse "pagine" o sezioni dell"applicazione.

Una domanda lecita che potresti farti è: nel caso volessi progettare un'app in cui non

tutte le pagine devono mostrare il menu,

come posso fare? Prova pensarci un p

o",

prima di continuare.

126

10.6 Progettare Template indipendenti dal

componente radice

Proviamo a rispondere alla domanda fatta un precedenza: come faccio a creare

un'app in cui alcune pagine hanno il menu e altre pagine non ce l'anno?

Abbiamo detto che tutto passa per il template del componente radice, che è il primo

che viene caricato:

<ca-menu><ca-menu>

<h1>MetroChat</h1>

<router-outlet></router-outlet>

Così facendo, tutte le route che andrò a creare saranno associate a dei componenti, i

cui template saranno visualizzati sotto la direttiva

<router-outlet>. Q uesto

implica che in ogni pagina ci sarà sia il menu, che la scritta MetroChat.

Per ovviare a questo comportamento, la soluzione è molto semplice perch

é sarà

sufficiente eliminare tutti i tag dal componente radice, e lasciare solo la direttiva

<router-outlet>.

<router-outlet></router-outlet>

In questo modo, ogni componente associato alle diverse

route, verrà visualizzato

senza venire "sporcato" dagli elementi presenti nel template del componente radice,

potendo così decidere se inserire in ogni templat e, un menu di navigazione

oppure

no.

Per l"applicazione MetroChat, potrei allora progettare dei template, che prevedano

l"inserimento manuale del componente <ca-menu>, in modo da essere libero di

crearne alcuni anche senza, proprio perché ho eliminato tutt i i tag dal componente

radice.

1) Template **componente radice** AppModule (<app-root>)

<router-outlet><router-outlet>

2) Template **componente TreniComponent** (<ca-treni>) con all'interno altri due

componenti

<ca-menu></ca-menu>

<h1>Treni in arrivo</h1>

<!--lista componenti metro -->

<ca-metro *ngFor="..." ...></ca-metro>

<ca-dettaglio ...></ca-dettaglio>

3) Template **componente** **PreferitiComponent** (<ca-preferiti>)

<ca-menu></ca-menu>

<h1>Preferiti</h1>

<!--lista preferiti-->

<ca-chat *ngFor="..." ...></ca-chat>

4) Template **componente** **LoginComponent** (<ca-login>)

<ca-menu></ca-menu>

<h1>Login App</h1>

<form>

```
<label>Username</label>

<input type="text" name="username">

<label>Password</label>

<input type="password"
name="password">

<button type="submit">Login</button>

</form>
```

10.7 Progettare Route di Redirect

L"ipotesi che sta alla base delle route progettate fino ad ora, è che l'utente acceda alle

diverse sezioni dell"applicazione, usando i link del menu o accedendo direttamente

dall'url.

Una delle o perazioni che in alcuni casi si ha la necessità di fare è reindirizzare il

navigatore che accede ad un particolare url, verso una pagina specifica in seguito ad

un evento specifico.

Tipico caso è quello legato ad un redirect non appena l'utente apre l'appl

icazione,

quindi accede all'url p redefinita, oppure dopo che si è aggiornato un dato su un

database remoto.

Nel primo caso è necessario creare una

corrispondenza tra l'url predefinito associato

al caricamento dell'app, e la path della

route vuota, al fine

di reindirizzare il

navigatore ad una path diversa.

La sintassi da usare per progettare una route di questo tipo sarà:

{ path: '', redirectTo: '/NUOVOURL', pathMatch: 'full' }

dove NUOVOURL, è il nome di una stringa rappresentativa dell'url associato

ad una

path di una route già definita.

Come vedi, sono state aggiunte nuove proprietà all'oggetto

Routes, tra cui

redirectTo, che mi permette di specificare a quale url indirizzare il navigatore e

pathMatch, per indicare che il confronto deve essere fatto s

u tutto l'url e non su

sottosezioni di questo.

Non appena l'utente apre l'applicazione, quindi

non appena viene caricato l'url

predefinito, Angular trova la corrispondenza con la path vuota '' e viene fatto un

indirizzamento ad una nuova route.

Nel caso de lla nostra applicazione, potremmo reindirizzare l'utente alla

route con

path impostato alla stringa "inarrivo" che mostrerà la lista dei treni in arrivo, quindi

il componente TreniComponent.

Il file di configurazione delle route dell"app, pertanto diventerà:

```
[

{ path: 'inarrivo', component: TreniComponent },

{ path: 'preferiti', component: PreferitiComponent },

{ path: '', redirectTo: '/inarrivo', pathMatch: 'full' }

]
```

mentre il menu di navigazione, cambierà

solo in corrispondenza al primo link:


```
<nav>

<li><a routerLink="/inarrivo">In arrivo</a></li>

<li><a routerLink="/preferiti">Preferiti</a></li>

<li><a routerLink="/login">Login</a></li>

</nav>
```

Nel caso tu voglia ottenere un re -

indirizzamento, non sulla base di un url specifico,

ma sul la base del verificarsi di un particolare evento,

devi sfruttare il metodo

navigate() di Router, che dovrà essere iniettato nel costruttore con un

meccanismo che impareremo meglio quando parleremo di Services.

La classe Router è da importare dal package @angular/router:

```
import {Router} from '@angular/router';
```

La sintassi da usare sarà:

this.router.navigate(['PERCORSO']);

dove router, è il nome che ho scelto per l'oggetto iniettato nel costruttore della

classe.

class MioComponente {

constructor(**private router: Router**) {

}

reindirizzoHome() {

this.router. **navigate(['/']);**

}

}

Il parametro impostato per il metodo navigate(), è un array con specificato il

percorso/path a cui reindirizzare l'utente

. Nel l"esempio indicato sopra,

reindirizzeremo ad una pagina precedente

già visitata e

memorizzata nella

cronologia del browser.

Chiaramente questo è l"uso base del metodo, ma più avanti vedremo come passare

anche dei parametri query nell"url.

10.8 Progettare Route di Pagina non trovata

Nel caso l'utent e inserisca nell'url un percorso non configurato nell'array di path,

devo prevedere un percorso alternativo che rimandi il navigatore ad una

classica

pagina del tipo: "Errore 404: pagina non trovata"

Ipotizzando di aver creato il componente *Error404.component.ts,* in questo modo:

```
import { Component} from '@angular/core';

import { Router } from '@angular/router';

@Component({

template: `
```

```
<h1>Pagina Non Trovata</h1>

<button (click)="backToHome()">Torna
alla Home</button>`

})

class Error404Component {

constructor(private router:Router) {

}

backToHome() {

this.router.navigate(['/']);

}
```

}

error404/error404.component.ts

La sintassi da usare sarà:

{ path: '**', component:
Error404Component }

Al posto della classica stringa identificativa del percorso, ho aggiunto due simboli di

asterisco, che stanno ad indicare la corrispondenza con una qualsiasi Route.

NB: E' importante allora assicurarsi che questa sia l'ultima delle righe di

configurazione delle diverse path, altrimenti l'applicazione visualizzerà sempre e

solo il template del componente Error404Component.

In alternativa, potre sti usare un re - indirizzamento verso un percorso specifico (es.

errore),

a cui sarà associat

a

la visualizzazione del

componente

Error404Component:

{ path: '**', redirectTo: '/errore',
pathMatch: 'full' }

10.9 Progettare Route con figli

L' applicazione MetroChat, prevede un menu di navigazione ad un solo livello di

profondità, dove con profondità , intendo l'equivalente numero di sottocartelle di un

classico sito web.

Potrebbe però capitare di avere la necessità di creare degli url con più livelli, come

nel caso qui sotto:

/inarrivo

1° livello di profondità (padre)

/inarrivo/dettaglio

2° livello di profondità (figlio)

/inarrivo/dettaglio/ID

3° livello di profondità (figlio)

La sezione dettaglio, è una sottosezione dell'url

inarrivo, così come quella con il

parametro *ID*, che potrebbe essere associato ad una query di interrogazione quindi

essere un parametro dinamico.

Per progettare Routes come queste e fare in modo che Angul

ar trovi la

corrispondenza tra l'url inserito e la path configurata, si utilizza una particolare

sintassi, che fa uso della proprietà

children di Routes, andando a inserire

all'interno di un array, i percorsi dei

componenti figlio da visualizzare.

{path: 'STRINGA', component: 'COMPONENTE_PADRE', children: [ALTRI_PATH]}

Pertanto il file di configurazione diventa:

{ path: 'inarrivo', component: TreniComponent},

{ path: 'inarrivo/dettaglio', component:

DettagliotrenoComponente, **children: [**

{'path': '', redirectTo: 'error', pathMatch: 'full' },

{'path': 'rosso', component: TrenorossoComponent },

{'path': 'giallo', component: TrenogialloComponent }

]

},

{ path: 'preferiti', component: PreferitiComponent },

{ path: 'login', component: LoginComponent },

{ path: '', redirectTo: '/inarrivo', pathMatch: 'full' },

{ path: 'error', component: Error404Component },

{ path: '**', redirectTo:'error', pathMatch: 'full' }

Nell'esempio qui sopra,

"/inarrivo" rappresenta la

route padre, mentre

"inarrivo/dettaglio"

una seconda route, associata al componente

DettagliotrenoComponent.

Quest"ultimo avrà definito nel proprio template un nuovo segnaposto

<router-outlet>, per indicare dove visualizzare il figlio,

nel nostro caso i

componenti abbinati alle ipotetiche route "giallo" e "rosso".

<ca-menu></ca-menu>

<h1>Dettaglio Treno</h1>

<a routerLink='giallo'>Treno Giallo-

<a routerLink='rosso'>Treno Rosso

<router-outlet></router-outlet>

La ricerca della corrispondenza tra l'url e le route, avviene in sequenza, andando a

vedere prima se c'è la

route con path *"inarrivo/dettaglio"* e se soddisfatta,

proseguendo con l'analisi dei figli.

In assenza di altri dati nell'url oltre a *"inarrivo/dettaglio"* , reindirizzo il navigatore

ad un url con path "error" che mostrerà il componente di pagina non trovata.

In presenza di un " ulteriore stringa nell"url - “giallo” o “rosso” - aggiungerò alla

visualizzazione del componente

DettagliotrenoComponent, anche il

componente associato a queste due route.

All'interno della proprietà children, ho inserito un array di

route, con la stessa

struttura di una route classica, con la differenza che all'interno di path, ho aggiunto

solo la parte di url che segue la stringa "*inarrivo/dettaglio"* .

Quando parleremo di route con parametri o

route dinamiche , vedremo come

sostituire alle due ipotetiche stringhe “giallo” e “rosso” , una stringa generica, che ci

permetterà di creare una corrispondenza tra diversi url e un singolo componente.

Questo processo lo posso ripete re per tutte le route figlio che intendo associare a

particolari percorsi url.

In particolare abbiamo inserito anche la route vuota ", al fine di reindirizzare il

navigatore ad una pagina di errore evitando così che l"utente veda una pagina senza

dei dettagli.

In questo modo ora riusciamo a gestire tutti gli url di seguito:

/ > componente TreniComponent

/inarrivo > componente
TreniComponent

/inarrivo/dettaglio > componente
Error404Component

/inarrivo/dettaglio/giallo > componente
DettagliotrenoComponent/TrenorossoCo

/inarrivo/dettaglio/rosso > componente
DettagliotrenoComponent/TrenogialloCo

/preferiti > componente
PreferitiComponent

/login > componente LoginComponent

/error > componente
Error404Component

/urlgenerica > componente
Error404Component

10.10 Impostare il Template della route di

dettaglio per l'applicazione MetroChat

Una delle modifiche che potremmo fare all'applicazione, potrebbe riguardare i dati di

dettaglio di un treno, che fino ad ora, venivano visualizzati nell o stesso template del

componente con selettore <ca-treni>.

Ogni treno sarà identificato da un identificativo alfanumerico, quindi potrei pensare

di visualizzare il dettaglio, non appena si digita un url simile a questo:

/inarrivo/dettaglio/XXX

dove al pos to di XXX , dovrà essere sostituita di volta in volta, una stringa

corrispondente all'identificativo del treno. Ancora non riusciamo a creare delle route

con parametri dinamici, quindi per ora

abbiamo fissato questo valore a due ipotetiche

stringhe, per intercettare almeno questo url e testare il funzionamento della route con

children.

Il componente di dettaglio DettaglioComponente, ha un template come quello

qui sotto:

```
<ca-menu></ca-menu>

<h1>Dettaglio Treno</h1>
```

<a routerLink='giallo'>Treno Giallo-

<a routerLink='rosso'>Treno Rosso

<router-outlet></router-outlet>

che ci permette di

visualizzare i dati del treno

, in corrispon

denza al

<router-outlet>. Il template del componente

TrenorossoComponent,

potrebbe essere progettato in questo modo:

134

ID Treno: Rosso

Ricapitolando, il file *app.module.ts* co i nuovi componenti, diventa:

import { NgModule } from '@angular/core';

import { BrowserModule } from '@angular/platform-browser';

import { RouterModule, Routes } from '@angular/router';

```
import { AppComponent } from
'./app.component';

import { MenuComponent } from
'./menu/menu.component';

import { TreniComponent } from
'./treni/treni.component';

import { DettagliotrenoComponent }
from

'./treni/dettagliotreno/dettagliotreno.comp

import { TrenorossoComponent } from

'./treni/dettagliotreno/trenorosso/trenc

import { TrenogialloComponent } from
```

'./treni/dettagliotreno/trenogiallo/treno

import { PreferitiComponent } from

'./preferiti/preferiti.component';

import { LoginComponent } from
'./login/login.component';

import { Error404Component } from
'./error404/error404.component';

@NgModule({

imports: [

BrowserModule,

RouterModule.forRoot([

```
{ path: 'inarrivo', component:
TreniComponent},

{ path: 'inarrivo/dettaglio', component:
DettagliotrenoComponent, children: [

{ path: '', redirectTo: '/error',
pathMatch: 'full'},

{ path: 'rosso', component:
TrenorossoComponent},

{ path: 'giallo', component:
TrenogialloComponent},

]

},
```

```
{ path: 'preferiti', component:
PreferitiComponent },

{ path: 'login', component:
LoginComponent },

{ path: '', redirectTo: '/inarrivo',
pathMatch: 'full' },

{ path: 'error', component:
Error404Component },

{ path: '**', redirectTo:'error',
pathMatch: 'full' }

])

],
```

declarations: [

AppComponent,

MenuComponent,

TreniComponent,

DettaglioTrenoComponent,

PreferitiComponent,

LoginComponent,

TrenorossoComponent,

```
],
```

(continua)


```
bootstrap: [ AppComponent ]
})
export class AppModule {}
```

app.module.ts

in cui, oltre alla definizione della nuova

route, ho im portato anche i nuovi

componenti

DettagliotrenoComponent,

TrenorossoComponent,

TrenogialloComponent aggiungendoli all'array della proprietà declarations.

Questi ultimi due, verranno sostituiti da un unico componente dettaglio, non appena

vedremo come creare route con parametri dinamici.

Il componente *dettagliotreno.component.ts*, sarà:

```
import { Component } from '@angular/core';

@Component({

selector: 'ca-dettagliotreno',

template: `<ca-menu></ca-menu>

<h1>Dettaglio Treno</h1>

<a routerLink='giallo'>Treno Giallo</a>-

<a routerLink='rosso'>Treno Rosso</a>
```

<router-outlet></router-outlet>`

})

export class DettagliotrenoComponent {

constructor() {}

}

treni/dettagliotreno/dettagliotreno.comp

mentre i due componenti provvisori

trenorosso.component.ts

e

trenogiallo.component.ts, avranno una

struttura simile:

```
import { Component } from
'@angular/core';

@Component({

selector: 'ca-trenorosso',

template: `<p>Treno Rosso</p>`

})

export class TrenorossoComponent {

constructor() {}

}
```

Ora che abbiamo capito come si creano url che possano essere intercettati con

opportune route figlio, vediamo di applicare queste tecniche per visualizzare su una

nuova "pagina", il dettaglio di ogni treno, non appena l'utente clicca sul componente

<ca-metro>.

Non conoscendo a priori quale sarà l'identificativo associato a ciascun treno,

dev

i

imparare la tecnica per creare una route con parametri dinamici.

10.11 Progettare Route con parametri dinamici

Ora che abbiamo capito il meccanismo con cui creare delle route con figli, vediamo

di realizzare la sezione dettaglio dell'applicazione MetroChat.

Abbiamo detto che questa sezione dovrà essere visualizzata in modo separato

dalla

lista dei treni, in modo che l'utente, una volta cliccato sul treno, acced a ad una nuova

sezione dell'app che mostri le ulteriori informazioni associate al treno.

E' chiaro che il primo passo da fare è progettare le route in modo che Angular sia in

grado di trovare la corrispondenza con url dinamici del tipo:

inarrivo/dettaglio/ID

dove con ID , intendo una qualsiasi

stringa rappresentativa
dell'identificativo del

treno.

Pertanto URL validi potrebbero essere:

inarrivo/dettaglio/AFD

inarrivo/dettaglio/FDE

etc.

E chiaro che , a priori, io non conosco l'identificativo associato al treno, quindi devo

avere un sistema in grado di creare una sorta di “segnaposto” nella definizione

della

route.

La sintassi che si usa è questa:

{ path: 'percorso/**:id**', component: COMPONENTE }

Come vedi ho inserito il parametro :id al posto di una stringa generica.

Questa notazione permette ad Angular di capire che al posto di :id, dovrà aspettarsi

una serie di parametri variabili, che

potranno essere stringhe o numeri.

NB: E' importante specificare i due punti che devono essere attaccati al nome

scelto per identificare il segnaposto del parametro dinamico. Quest'ultimo può

essere indicato con una qualsiasi stringa valida, non necessariamente con id.

Tornando allora alla lista di route configurate in precedenza, potrò aggiungere un

componente generico DettaglioComponent, che verrà associato ad una path con

un parametro dinamico.

```
{ path: 'inarrivo', component:
TreniComponent},

{ path: 'inarrivo/dettaglio', component:
DettagliotrenoComponente,

children: [

{'path': '', redirectTo: 'error', pathMatch:
'full' },

{ path: 'giallo', component:
GialloComponent},

{ path: 'rosso', component:
TrenorossoComponent},
```

{'path': ':id', component: DettaglioComponent }

]

},

{ path: 'preferiti', component: PreferitiComponent },

{ path: 'login', component: LoginComponent },

{ path: '', redirectTo: '/inarrivo', pathMatch: 'full' },

{ path: 'error', component: Error404Component },

{ path: '**', redirectTo:'error', pathMatch: 'full' }

NB: Fai attenzione che in questo caso l'ordine di apparizione delle diverse

path, ha una sua importanza. Difatti, se inserissi la path dinamica, prima della

path associata al componente Giallo o Rosso, questi ultimi non verrebbero

mai visualizzati: Angular troverebbe subito una corrispondenza con la path

dinamica e si fermerebbe nella ricerca di altre path valide.

Tutto funziona se digito manualmente

l'url nel browser. Quello che ci serve però è un

link da cliccare.

La domanda che sorge spontanea è: c ome faccio a creare dei link con parametri , da

inserire all'interno del template costituito dalla lista di treni?

Non posso usare la notazione che

si usa nel mondo delle pagine web

, ossia

aggiungere un parametro query all''url,

come evidenziato qui sotto:

Link dettaglio

Devo ricordarmi invece della direttiva RouteLink, che ora verrà usata nella forma

estesa, ossia con l'aggiunta di u n array al posto della classica stringa indicativa del

percorso:

<a [routerLink] = "[PERCORSO, PARAMETRO]"> ...

L'array è inserit o all'interno delle doppie virgolette, ed è costituito da

lle coppie

PERCORSO e PARAMETRO, che dovranno essere personalizzate.

Ad esempio per l "applicazione MetroChat, che richiede un link cliccabile all"interno

del componente *treni.component.ts*, potrei modificare il template e scrivere:

<a [routerLink] = "['inarrivo/dettaglio', metro.idt]"> ...

dove inarrivo/dettaglio è l"identificativo

della path, mentre metro.idt è

proprio l"identificativo associato ad ogni treno e che potrai sfruttare in una fas e

successiva per recuperare i dati di dettaglio.

Per indirizzare il navigatore allo stesso link, con un"istruzione inserita nel corpo del

componente e non nel template, si sfrutta il metodo navigate(), che avevamo

visto quando abbiamo parlato di route di Redirect:

this.router.navigate([PERCORSO, PARAMETRO]);

Il componente *treni.component.ts*, potrebbe quindi diventare:

```
import { Metro } from './../model/metro.model';

import { Component, OnInit } from '@angular/core';

import { Router } from '@angular/router';

@Component({

selector: 'ca-treni',
```

```
template: `

<ca-menu></ca-menu>

<h1>{{title}}</h1>

<p>{{trenipartiti}}</p>

<ca-metro *ngFor="let metro of
listametro"

(inPartenza)="partiti($event)"

[datiIn]="metro"
```

(continua)

```
[ora]="now"

(click)="setMetro(metro.idt)" >

</ca-metro> `

})

export class TreniComponent
implements OnInit {

listametro: Metro[];

trenipartiti:string;

constructor(private router: Router) {

this.trenipartiti = '';
```

```
this.listametro = [];

this.now = new Date().getTime();

}

ngOnInit() {

this.listametro = LISTAMETRO;

}

setMetro(id: string) {

this.router.navigate(['inarrivo/dettagli
id]);

}
```

```
partiti(id: string) {

this.trenipartiti += '|' + id;

}

}
```

treni/treni.component.ts

dove il grosso del cambiamento è avvenuto all'interno della funzione setMetro(),

in cui il parametro ricevuto in ingresso, rappresentativo dell'identificativo del treno, è

stato usato per indirizzare il navigatore

ad un url del tipo:

inarrivo/dettaglio/N

A questo p unto Angular, sulla base delle route configurate all'inizio, è in grado di

trovare la corrispondenza con l'url così creato e visualizzare il componente Dettaglio.

E' chiaro che ora non possiamo più passare in ingresso un

oggetto, ma dovremo

sfruttare l'informazione passata tramite l'url per recuperare tutti i dettaglio del

treno.

Questo passaggio lo vedremo quando parleremo di recupero dati da una sorgente

esterna.

10.12 Recuperare i parametri da un url

Al fine di visualizzare la pagina dettaglio, dobbiamo prima di tutto imparare a

recuperare l'informazione passata tramite l'url, nel nostro caso

l'identificativo del

treno.

Eravamo giunti infatti a questo punto:

1. Click sul co mponente <ca-metro> che richiama il metodo setMetro(),

passando un parametro id

2. Apertura di un url del tipo: *inarrivo/dettaglio/XXX* dove al posto di XXX

verrà visualizzato l'identificativo associato al treno

3. Intercettazione dell'url tramite le route

con parametri

4. Visualizzazione componente dettaglio

setMetro(id: string) {

// indirizzo l’utente alla sezione dettaglio

this.router.navigate(['inarrivo/dettaglio', id]);

}

Purtroppo ancora non abbiamo gli strumenti per recuperare i dati di dettaglio de

l

treno sulla base di un identificativo, ma possiamo per ora limitarci a visualizzare un

messaggio di alert (), che mostri il parametro recuperato dall'url, con il quale

successivamente potrò interrogare un servizio remoto.

Se ti ricordi, il componente dett aglio creato fino ad ora aveva queste caratteristiche

in termini di route:

{ path: 'inarrivo/dettaglio', component: DettagliotrenoComponente,

```
children: [

{'path': '', redirectTo: 'error', pathMatch:
'full' },

{ path: 'giallo', component:
GialloComponent}

{ path: 'rosso', component:
TrenorossoComponent}

{'path': ':id', component:
DettaglioComponent }

]

}
```

Al fine di poter recuperare dall'url il

parametro passato e valorizzare

così una

proprietà interna al componente, è necessario iniettare nel costruttore della classe, un

“Service” (vedi capitolo dedicato)

- per la precisione

ActivatedRoute e

ParamMap - che ha nno all'interno una serie di proprietà e metodi (snapshot e

get) che permettono di accedere a tutte le informazioni sulla route.

Una delle sintassi che potresti usare per il recupero di un singolo parametro

è la

seguente:

this.idtreno = oggettoRouter.snapshot.paramMap.get('i

Come puoi osservare, ho

passato al metodo get(), lo stesso identificativo id

presente nella definizion e della r oute

dinamica, mentre

oggettoRouter è la

variabile privata passata al costruttore, con il meccanismo delle

“Dipendency

Injection”, che vedremo in seguito quando parleremo di

“Service”. Il dato sarà di

tipo stringa.

Questa, come dicevo, è una possibile sintassi, che viene usata quando l"utente

necessita di visualizzare solo un"informazione, per poi passare ad altre sezioni

dell"applicazione.

Quando parleremo di “Observable”, vedremo che questa modalità di recupero, potrà

essere modificata e resa più efficiente . Potresti infatti riutilizzare il componente per

visualizzare più dati in successione, in funzione di un cambio della query ottenuto

con dei link interni allo stesso

componente.

In questi casi Angular, ad ogni cambio qu

ery non crea una nuova istanza del

componente, ma riutilizza quella creata alla prima chiamata. Questo chiaramente se

l"utente rimane nella stessa pagina e non clicca su altre sezioni dell"applicazione.

Giusto per farti vedere anche questa sintassi, dovrem

o sfruttare l'iscrizione ad un

Observable – restituito dalla proprietà paramMap - per tenere monitorato ogni

eventuale cambiamento dell'url fatto da azioni interne al componente.

```
oggettoRouter.paramMap.subscribe(para
=> {

this.idtreno = params.get('id');

});
```

Tornando al nostro obiettivo, l e ulteriori librerie che dovremo aggiungere saranno

prelevate dallo stesso package di Router quindi da @angular/router:

import { ActivateRoute, ParamMap } from '@angular/router';

Il componente *dettaglio.component.ts* pertanto, potrà essere modificato aggiungendo

queste istruzioni:

import { Component, OnInit } from '@angular/core';

import { ActivatedRoute, ParamMap } from '@angular/router';

@Component({

```
selector: 'ca-dettaglio',

template: `<p>ID: {{idtreno}}</p>

<p>Lista Chat…</p>`

})

export class DettaglioComponent
implements OnInit {

idtreno:string;

constructor(private route:
ActivatedRoute) {

this.idtreno =
this.route.snapshot.paramMap.get('id')
```

```
// oppure in alternativa

//
this.route.paramMap.subscribe(params
=> {

// this.idtreno = params.get('id');

// });

}

ngOnInit() {}

}
```

treni/dettaglio/dettaglio.component.ts

Navigando all"url del tipo:

inarrivo/dettaglio/XXX

corrispondente alla path:

inarrivo/dettaglio/:id

verrà recuperato l'id associato

al parametro passato tramite l'url

(XXX), che

valorizzerà una proprietà idtreno, collegata al template del componente.

NB: Ricorda che il dato così recuperato è di tipo stringa, quindi eventualmente puoi

usare il simbolo + per trasformarlo in numero intero.

this.idtreno =
+oggettoRouter.snapshot.paramMap.get('

Con queste righe implementate, ora abbiamo le basi per recuperare un qualsiasi dato

da un"API remota, cosa che vedremo in uno dei prossimi capitoli.

Capitolo 11

Separare le funzionalità con i Service

11.1 Perché usare i Service?

Abbiamo più volte detto che un'applicazione Angular non è altro che un insieme di

componenti aventi una struttura ad albero, partendo dal componente radice e via via

proseguendo verso quelli interni.

Un principio base per lo sviluppo di un componente è che deve svolgere una

specifica mansione , quindi deve contenere un numero limitato di righe: in poche

parole deve essere semplice e sfruttare eventuali funzionalità specifiche richieste per

il suo funzionamento, prelevandole da servizi esterni.

Questo nel rispetto del principio visto nel Capitolo 9 detto “

SRP” (Single

Responsibility Principle)

Ad esempio, quando un"impresa costruisce un"abitazione, potrebbe decidere di

produrre tutto in loco, dai mattoni, alla

malta, ai tubi, alle mattonelle etc.

E" chiaro che sarebbe poco efficiente, perché la complessità nel gestire tutte le

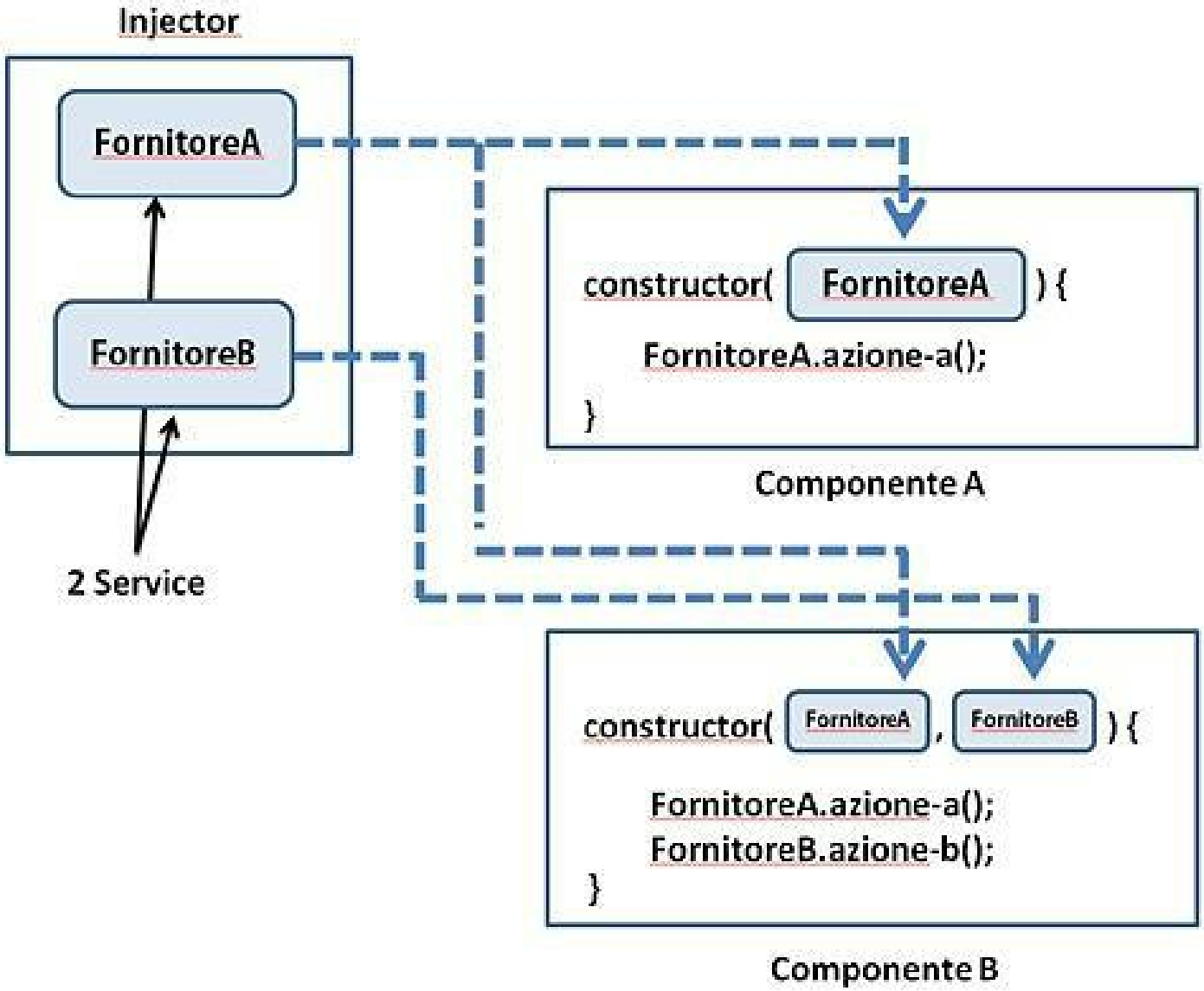
maestranze, creerebbe troppi colli di bottiglia.

Meglio delegare a servizi esterni e far fare i diversi elementi a ditte specializzate,

mentre in loco, limitarsi solo ad assemblarli e fare poche lavorazioni.

Nella figura qui sotto, puoi vedere questi concetti applicati ad un"applicazione

Angular costituita da due componenti e da due fornitori di servizi.



In Angular questo concetto si concretizza

con la realizzazione di “Service”, o servizi

in italiano e tramite il meccanismo chiamato "Dependency Injection" o iniezione

delle dipendenze. I Service sono delle classi TypeScript.

Non entreremo nel dettaglio della filosofia di progettazione legata a q uesti concetti e

nemmeno parleremo di quello che succede a livello tecnico, ma ci limiteremo a dire

che grazie a questa tecnica, non sarà più

compito di ogni componente recuperare i

diversi “pezzi” necessari al suo funzionamento, ma tutto sarà centralizzato.

Sarà quindi compito dell"applicazione creare e fornire i “pezzi” ai diversi

componenti che li richiederanno.

Centralizzando il processo inoltre, anche tutta la fase di test di un'applicazione viene

semplificata, perch é basterà cambiare una configurazion

e a livello centrale per

rifletterla a tutta l'applicazione.

Abbiamo già incontrato alcuni esempi d "uso dei Service, quando abbiamo parlato di

navigazione e recupero dei parametri d a un url. Ora cercheremo di chiarire meglio

questi concetti e di applicarli a un caso reale : il **recupero dei dati da un a sorgente**

esterna.

In tutti gli esempi visti fino ad ora , che coinvolgevano la visualizzazione dei dati,

abbiamo sempre ipotizzato

che questi fossero memorizzati all'interno di una

costante.

Questa non è certamente una situazione ideale, in quanto i dati non saranno mai delle

costanti, ma saranno soggetti a cambiamenti. Inoltre

spesso avrò la necessità di

effettuare delle operazioni di filtro e manipolazione,

per estrarre solo alcune

informazioni.

Grazie ai Servi ce e al meccanismo della "Dependency Injection", invece di far fare

tutto ad uno specifico componente e ricopiare

le stesse funzionalità

su altri

componenti, si lascia che tutte le

operazioni che coinvolgono i dati

, siano fatte da

una classe specifica, o diverse classi, che poi forniranno il proprio materiale (i dati o

le funzionalità) a coloro che li richiederanno.

Il modello che si sfrutta è quello della creazione di una singola istanza,

o

“SingleTon” come viene detto in gergo tecnico.

Un utilizzo frequente dei Services, è proprio legato a tutte le operazioni di IO (Input

e Output).

Cerchiamo allora una soluzione per rendere indipendenti le funzionalità legate a dati,

e poterle poi fornire (iniettare) all"occorrenza a chi ce le richiederà.

11.2 Come creare un Service

Tutti i Service c

he avremo la necessità di creare

, sono delle normali classi

TypeScript, che per convenzione vengono memorizzate con un nome di file che

termina con l"estensione *.service*, sempre con lettere minuscole per evitare problemi

nel caso in cui l'applicazione venga erogata da server sensibili ai nomi in maiuscol o

e minuscolo.

L'unica differenza rispetto alla classica classe TypeScript è che dovrà contenere un

particolare decoratore **@Injectable()** che permetterà ad Angular di "ricono scerla"

come classe da poter usare grazie al meccanismo del “Dependency Injection”.

Come tutte le classi, che non siano dei

componenti, non presenta un template ed è

presente la parola export, perché la dobbiamo esportare come un modulo.

La sintassi da cui partire per creare un Service sarà:

```
import { Injectable } from '@angular/core';

@Injectable()

export class NomeClasse {
```

constructor(parametri da iniettare) {

}

}

nomeclasse.service.ts

Nota la presenza della riga per importare

, dalla libreria

@angular/core,

Injectable e n ota come il decoratore sia stato inserito

facendolo seguire dalle

parentesi, senza indicare per ora, alcun argomento.

Lo stesso decoratore @Component() è un particolare sottotipo di @Injectable()

Un Service può chiaramente avere un costruttore, delle prop

rietà e dei metodi,

esattamente come tutte le classi viste fino ad ora. L' unica differenza sarà la modalità

con cui potrò accedere a questi membri dall'esterno, ossia da altri componenti.

Si dovranno applicare gli stessi concetti

visti per

il service nativo di Angular

ActiveRouting, i cui metodi venivano utilizzati solo dopo avere "iniettato"

l'oggetto nel costruttore del componente.

Tornando all'esempio dell'app ch e stiamo sviluppando, al fine d"isolare i dati dal

codice del componente principale, potre mmo creare un Service dedicato proprio al

recupero di questi da un database remoto.

Per ora limitiamoci a inserirli manualmente, così come fatto in precedenza, quando

abbiamo creato i dati rappresentativi dell"elenco dei treni e dei messaggi scambiati.

Creiamo una costante da esportare di nome LISTAMETRO, che salveremo all"interno

del file *listametro.ts* nella cartella *dati:*

```
import { Metro } from
'./../metro/metro.model';

export const LISTAMETRO: Metro[]
= [
```

```
{idt:'ASD', linea:'Rossa',
numchatting:23, tempo:500},

{idt:'AKE', linea:'Verde',
numchatting:33, tempo:900},

{idt:'BFD', linea:'Gialla',
numchatting:13, tempo:1500}

];
```

dati/listametro.ts

Vediamo anche un possibile esempio di sviluppo del Service, che potremmo salvare

con nome *treni.service.ts* nella cartella *service* interna alla cartella principale *app*: **import { Injectable } from '@angular/core';**

```
import { Metro } from './../metro.model';

import { LISTAMETRO } from './../dati/listametro';

@Injectable() // non ho ancora indicato chi creerà il service

export class TreniService {

constructor() {}

getListaMetro(): Metro[] {
```

```
// recupero i dati statici per ora

return LISTAMETRO;

}

getDettaglioMetro(id: string): Metro {

// recupero uno specifico elemento
dell'array

}

}
```

service/treni.service.ts

Come puoi osservare, per ora il costruttore non dipende dal altri

Service, quindi non

ha delle dipendenze e sono presenti solo due metodi,

getListaMetro() e

getDettaglioMetro(), usati rispettivamente per recuperare la lista completa de i

treni e il dettaglio, sempre sfruttando

dei dati statici già inseriti nell'app e che

dovranno essere sostituiti con dati recuperati in tempo reale.

Con il progredire delle funzionalità

dell'app, potrò aggiungere altri metodi specifici,

come ad esempio, la memorizzazione dei treni preferiti.

Fai attenzione che , come per tutte le classi, fino a

quando non creo l'istanza di

questa, non succederà nulla. La domanda da farsi è quindi: chi crea l''istanza di un

service e come? Vediamo allora l' ultimo passaggio da fare per poter sfruttare i n tutti

i componente, i metodi interni alla

classe.

11.3 Registrare un Service

Definire una classe non serve a molto se

poi non si crea l"istanza di questa.

Fortunatamente in Angular non ci dobbiamo preoccupare di

questa operazione,

perché viene fatta in automatico, a p atto di effettuare la cosiddetta “**registrazione di**

un Service” .

Abbiamo infatti detto che lo scopo dei Service è quello di semplificare le mansione

di un componente, andando a creare delle funzionalità gestite in modo centralizzato,

che all'occorrenza veng ono fornite, t ramite un processo di “iniezione”, ai diversi

componenti di un'applicazione.

A partire dalla versione 6 .0 di Angular, sarà sufficiente aggiungere al decoratore

@Injection, un oggetto (metadato) con la chiave providerIn, valorizzata con

una stringa che identifica chi dovrà creare il Service.

@Injectable({

prividerIn: ‘root’

})

Con la stringa „root", demandiamo all" *Application Injector* l"onere di creare il

Service. In questo modo sarà disponibile per tutti i componenti dell"applicazione

tramite il meccanismo dell" “Iniezione delle dipendenze” (“Dependency Injection”),

a patto di importarlo.

Nelle versioni di Angular antecedenti alla 6.0 , si p oteva omettere questo metadato ,

ma era necessario fare delle modifiche al file app.module.ts. In rete troverai molti

esempi di codice che sfruttano ancora questa sintassi, quindi è bene conoscerla.

Si aggiunge il nome del Service

alla proprietà

providers del decoratore

@NgModule() presente nel

modulo radice dell"applicazione. Ad esempio ,

nell"ipotesi tu abbia due Service di nome *fornitoreA* e *fornitoreB*, scriveremo:

import { fonitoreA,fornitoreB } from '/servizi';

@NgModule({

declaration:[MioComponent],

provider: [fornitoreA,fornitoreB]

}

)

app.module.ts

All"interno della proprietà provider, si defin isce un array con l"elenco dei

“fornitori” da usare. Questa sintassi è ancora ammessa nella versione 6 e successive ,

anche se noi useremo quella che sfrutta la proprietà providerIn. Per velocizzare la

creazione di un service, si può sfruttare direttamente la linea di co mando CLI, che

creerà gran parte del codice visto sopra:

ng g service treni

11.4 Come usare un Service

All"interno del componente che ha la necessità di usare le funzion alità del Service, si

aggiunge al costruttore un argomento, corrispondente ad una proprietà privata di tipo

pari al Service (nome della classe) che necessito.

Ad esempio, se avessi creato un Service con nome fornitoreA, e questo mi

servisse nel componente Cucina, potrei scrivere:

```
import { fornitoreA } from
'./fornitoreA.service';

@component({

selector: 'cucina',

templateUrl: '...'

})

export CucinaComponent {

constructor(private
pavimenti:fornitoreA) {
```

// ora posso usare il service con this.pavimenti

}

}

cucina.component.ts

Al costruttore della classe, h

o aggiunto un a rgomento di nome pavimenti

indicando come tipo di dato, il nome scelto per il Service.

Affinché Angular sia in grado di recuperare la definizione di fornitoreA,

lo dovrò

importare con la classica istruzione di import.

Una volta che il service è reso disponibile al componente, posso util izzare i diversi

metodi definiti all"interno che lavorano sull"istanza originaria (singola).

Questa istanza è quella che si crea grazie all"indicazione del *provider*, inserito

direttamente come argomento al decoratore @Injection. Questo , ripeto, se adottiamo

la sintassi introdotta a partire dalla versione 6.0 di Angular.

11.5 Gestire Dati Remoti con i Service

Tornando all'esempio dell'app

MetroChat, modifichiamo il codice visto in

precedenza in modo da aggiungerci anche l "oggetto con la proprietà providerIn.

NB: Questa operazione non è necessario se hai creato il service sfruttando la linea di

comando.

```
import { Injectable } from
'@angular/core';

import { Metro } from './../metro.model';

import { LISTAMETRO } from
'./../dati/listametro';

@Injectable({

providerIn: ' root'

})

export class TreniService {
```

```
constructor() {}

getListaMetro(): Metro[] {

// recupero i dati statici per ora

return LISTAMETRO;

}

getDettaglioMetro(id:string): Metro {

// recupero uno specifico elemento
dell'array

}

}
```

service/treni.service.ts

A questo punto, siamo pronti per poter sfruttare il meccanismo del "Dependency

Injection", per iniettare il Service

all'interno del componente responsabile della

visualizzazione della lista dei treni in arrivo.

```
import { Component, OnInit } from '@angular/core';

import { Router } from '@angular/router';
```

```
import { Metro } from
'./../model/metro.model';

import { TreniService } from
'./../service/treni.service';

@Component({

selector: 'ca-treni',

template: `

<ca-menu></ca-menu>

<p>{{trenipartiti}}</p>

<ca-metro *ngFor="let metro of
listametro"
```

```
(inPartenza)="partiti($event)"

[datiIn]="metro"

[ora]="now"

(click)="setMetro(metro.idt)">

</ca-metro>

`})

export class TreniComponent
implements OnInit {

listametro: Metro[];

trenipartiti:string;
```

```
now:number;

constructor(private router: Router,
private treniservice: TreniService) {

this.trenipartiti = '';

this.listametro = [];

this.now = new Date().getTime();

}
```

(continua)

```
ngOnInit() {
```

```
this.listametro =
this.treniservice.getListaMetro();

}

setMetro(id:string) {

this.router.navigate(['inarrivo/dettaglio',
id]);

}

partiti(id:string) {

this.trenipartiti += '|' + id;

}

}
```

treni.component.ts

All'interno di ngOnInit(), ora potrò richiamare il metodo del Service, che abbiamo

provveduto ad iniettare nel costruttore della classe, tramite una variabile privata di

nome treniservice.

Il costruttore ora ha due argomenti, uno usato per accedere a Router e l'altro per

accedere a TreniService.

Visto che il metodo del Service getListMetro() restituisce proprio un

array di

oggetti Metro, posso usarlo direttamente per valorizzare la proprietà listametro.

Analogo discorso per il componente di dettaglio, che anch'esso avrà biso

gno del

Service, per sfruttare il metodo getDettaglioMetro(id) al fine di recuperare i

parametri del treno sulla base dell'identificativo passato.

....

```
import { Metro } from
'./../../../model/metro.model';

import { ActivatedRoute, ParamMap }
from '@angular/router';

import { TreniService } from
'./../../../service/treni.service';

@Component({

selector: 'ca-dettaglio',

template: `<div *ngIf="treno">

<p> Treno Linea: {{treno.linea}} </p>

<p> ID: {{treno.idt}}</p>
```

```
</div> `

})

export class DettaglioComponent
implements OnInit {

idtreno: string;

treno: Metro;

constructor(private route:
ActivatedRoute, private treniservice:

TreniService) {

}
```

(continua)

```
ngOnInit() {

this.idtreno =
this.route.snapshot.paramMap.get(' id'
);

this.getDettaglioMetro(this.idtreno);

}

// Richiamo il metodo
getDettaglioMetro() del Service

getDettaglioMetro(idtr:string) {

this.treno =
```

this.treniservice.getDettaglioMetro(id

}

}

dettaglio.component.ts

Dovendo recuperare dei dati, in modalità asincrona,

da una sorgente remota

caratterizzata da un certo tempo di latenza , è preferibile spostare le righe interne al

costruttore, all"interno del metodo ngOnInit().

Per comodità ho creato un metodo interno con lo stesso nome di quello presente nel

service - getDettaglioMetro() - , a cui ho passato il parametro recuperato dall'url

e rappresentativo dell"identificativo del treno selezionato.

Il metodo getDettaglioMetro() del Service invece, restituisce un oggetto di tipo

Metro, che userò per valorizzare la proprietà treno di tipo Metro.

Inoltre dovrò accertarmi che, nel momento in cui verrà visualizzato il

template, la

variabile treno non sia nulla, qu esto per evitare che Angular seg nali un errore

legato alla mancanza dell'oggetto.

Posso agevolmente sfruttare la direttiva *ngIf:

template: `<div *ngIf="treno">

 Treno Linea: {{treno.linea}}

 ID: {{treno.idt}}

<h2>Lista Chat</h2>`

Il passo successivo , per liberarci definitivamente dei dati statici memorizzati

all'interno del file *datiremoti.ts*, è quello di

scoprire cosa mette a disposizione

Angular, per gestire l'accesso a sorgenti

di informazioni memorizzate su s

ervizi

remoti.

Capitolo 12

Accedere a dati remoti

12.1 Manipolare dati via HTTP

Un'applicazione che non sia in grado di interagire con l'utente e di rispondere alle

richieste, è' come una macchina senza motore: serve a poco.

Noi non ci preoccuperemo di realizzare un'applicazione stile WhatsApp, quindi

un"app in grado di comunicare in tempo reale ; cercheremo di gettare le basi per

capire come avviene lo scambio d

"informazioni tra un'applicazione Angular e un

ipotetico servizio remoto, ad esempio un"API in grado di interrogare un database e di

scriverci all"interno.

Attraverso queste basi, potrai poi proseguire per creare la tua applicazione in tempo

reale, sfruttando ad esempio servizi come “Firebase Database”.

Il primo passo per poter comunicare con A

PI raggiungibili tramite la rete internet

sfruttando il protocollo HTTP, è **configurare** l'applicazione per sfruttare una serie di

servizi non inclusi nei moduli standard di Angular.

Questi servizi sono già stati creati e per poterli sfruttare con il meccani

smo del

"Dependency Injection", dovremo importare un modulo, invece di usare la proprietà

provider, come visto per i classici

Service.

Il primo passo sarà quello di andare a modificare la proprietà import del decoratore

@NgModule() presente nell' AppModule, in modo che

in automatico

venga

effettuata la registrazione del service.

A partire dalla versione 4.3 di Angular, è stata introdotta una nuova classe per gestire

tutte le richieste da inviare in rete tramite

HTTP. Questa classe prende il nome di

HttpClient, e sostituisce la precedente

Http. Sono state aggiunte diverse

funzionalità, ma noi vedremo solo le principali, finalizzate al nostro obiettivo.

Le due classi in realtà sono molto simili e potrebbero essere usate in modo indistinto

per le necessità del nostro progetto, ma chiaramente basiamoci su questa versione

potenziata.

Analizziamo allora le modifiche da apportare al codice del file

app.module.ts del

progetto, al fine di poter interrogare API remote e inviare dati via HTTP.

...

```
import { BrowserModule } from '
@angular/platform-browser';
```

```
// 1) Importazione di HttpClientModule

import { HttpClientModule } from
'@angular/common/http';

...

@NgModule({

declarations: [

...

],

// 2) Aggiunta di HttpClientModule alla
proprietà

imports: [
```

BrowserModule,

HttpClientModule

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

app.module.ts

Tralasciando le sezioni già viste e indicandole con

dei puntini di sospensione ...,

possiamo notare:

1) la classica riga per importare un modulo, che è HttpClientModule dal

package @angular/common/http, e poi

2) l'inserimento di questo, all'interno della proprietà imports.

E" importante osservare anche l"ordine in cui è stata inserita la riga all"interno di

imports. Questa infatti deve seguire quella relativa all"importazione del modulo

BrowserModule.

Aggiunte queste due righe al file,

possiamo procedere con l"utilizzo del Service

HttpClient, essendo ora disponibile per tutti i componenti dell'app,

tramite il

meccanismo della "Dependency Injection".

Questo Service ci permetterà di accedere alla rete sfruttando

XMLHttpRequest

(XHR).

In questo modo elimineremo definitivamente il file locale sfruttato in tutti gli esempi

visti fino ad ora.

Le modifiche che dovremo fare , pertanto, riguarderanno il Service *metro.service.ts* e

in futuro anche il Service che svilupperemo per recuperare la lista dei messaggi della

chat.

12.2 Recuperare dati in GET

La prima domanda da farsi è : che forma avranno i dati restituiti dal servizio remoto

che andremo a interrogare?

Nella quasi totalità dei casi, dovremo manipolare dei dati rappresentati nel formato

JSON, quindi la prima ipotesi che faremo è che i dati dei diversi treni in arrivo, siano

rappresentati co n la notazione JSON costituit a da una chiave /proprietà di nome

dati, con all"interno un array di oggetti metro:

{

"dati": [

{

"idt":"12345",

"linea":"Rossa",

"direzione":"Centraal Station",

"numchatting":"12",

"tempo":"1488053502000",

"stazione":"Diemen Zuld"

},

{

"idt":"34523",

"linea":"Gialla",

"direzione":"Centraal Station",

"numchatting":"23",

"tempo":"1498053534000",

"stazione":"Zuld"

}

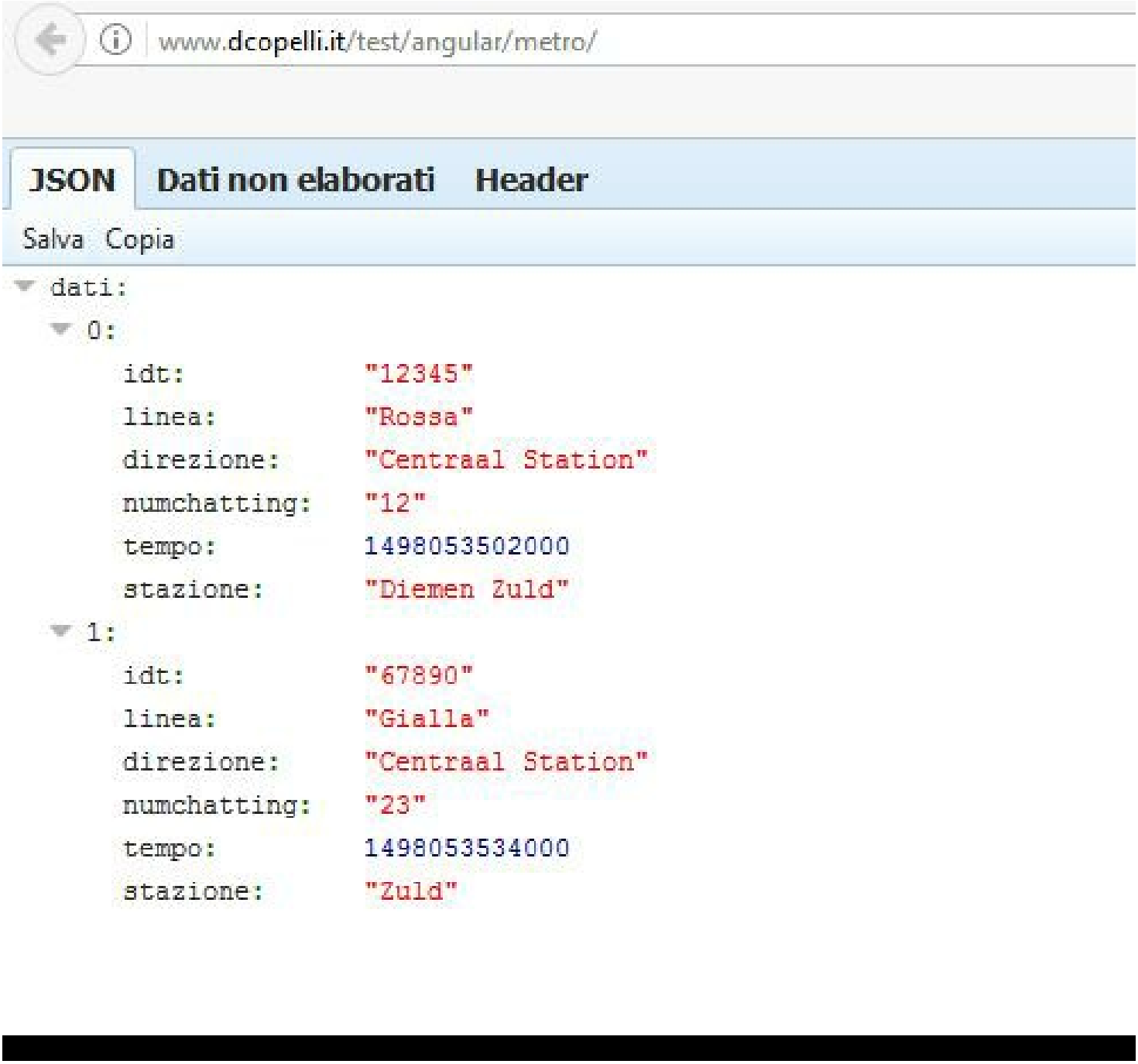
]

}

In questo modo potrò sfruttare proprio

la chiave dati per scorrere tra tutti gli

elementi dell"array e visualizzare così l"elenco dei treni.



Se tu provassi ad accedere all"url dell"API adibita al recupero di questo

elenco (vedi

appendice), visualizzeresti proprio il dettaglio di due treni:

fig.12.2.1

Nell"ipotesi il server remoto sia dotato di interprete PHP, potresti abbozzare un

programma in grado di restituire tale stringa, scrivendo:

…

// preparo la connessione ad database

// e interrogo la tabella ipotetica treni

```
while($row =
mysqli_fetch_array($query)) {

$results['idt'] = $row['idtreno'];

$results['linea'] = $row['linea'];

$results['direzione'] = $row['direzione'];

$results['numchatting'] =
$row['numchatting'];

$results['tempo'] = $row['unixtime'];

$results['stazione'] = $row['stazione'];

array_push($strout, $results);

}
```

```
header('Content-Type:
application/json');

echo '{"dati":' . json_encode($strout) .
'}';
```

metro/index.php

157

dove ho supposto che tutti i dati siano memorizzati all"interno di un database

MySQL. Analogo discorso potresti fare con Node.

Per poter accedere a dati remoti via

HTTP, Angular fornisce il metodo

get(),

presente all'interno del Service

HttpClient, che dovrà essere iniettato nel

costruttore della classe . Il metodo ha diverse “firme”

, ma noi useremo quello

standard, che prevede il passaggio del solo parametro obbligatorio URL.

La sintassi

di una tipi

ca richiesta via HTTP,

sarà caratterizzata da due

concatenamenti: i metodi get() e pipe(). Quest"ultimo è un operatore che fa parte

della libreria RxJS (Reactivex for JavaS

cript), che permette di

eseguire in

successione più funzioni, senza usare la classica notazione della concatenazione:

oggettoHttp.get<TIPO>(URL) //
restituisce un Observable

.pipe(

map(),

catchError()

);

URL è l'indirizzo dell"API da interrogare , mentre map() è anch"esso un operatore

della libreria RxJS, che permette di eseguire una trasformazione su ogni singolo dato

dell" **Observable,** proprio quello restituito da get(). Ad esempio potremmo estrarre

solo una certa chiave dai dati forniti.

Essendo un dato Observable, al fine di poter accedere ai dati restituiti, dovrò

effettuare l"operazione di subscribe().

A livello predefinito, i l corpo della risposta fornito da http.get(), è un oggetto

JSON senza tipo. Al fine di

indicare il tipo di dato restituito

, si può inserire lo

specificatore di tipo opzionale con la notazione <TIPO> (es. <Metro[]>).

Infine è presente l"operatore RxJS

catchError(), per poter intercettare eventuali

errori di comunicazione e gestirli nel subscribe(), perché anche catchError()

restituirà un Observable.

Applicando questi concetti al service *metro.service.ts* e ipotizzando di creare due

metodi ex novo, il primo getListaMetroObservable(), da usare al posto del

precedente getListaMetro() e il secondo getDettaglioMetroObservable()

da usare al posto di getDettaglioMetro(), otterremo il seguente codice:

```
import { Injectable } from
'@angular/core';

import { Metro } from
'./../model/metro.model';
```

(*continua)*


```
import { LISTAMETRO } from
'./../dati/metroremoti';

// 1

import { HttpClient } from
```

'@angular/common/http';

import { throwError, Observable } from 'rxjs';

import { catchError, map, tap } from 'rxjs/operators';

@Injectable()

export class MetroService {

// 2

private apiGetUrl = 'https://www.dcopelli.it/test/angular/m

// 3

```
constructor(private http: HttpClient)
{}

getListaMetro(): Metro[] {

return LISTAMETRO;

}

getListaMetroObservable():
Observable<Metro[]> {

return this.http.get<Metro[]>
(this.apiGetUrl)

.pipe(

map(risposta => risposta['dati']),
```

```
catchError(this.handleErrorObs)

);

}

getDettaglioMetroObservable(idt:
string): Observable<Metro> {

return this.http.get<Metro>
(this.apiGetUrl + '?idt='+idt)

.pipe(

map(risposta => risposta['dati']),

catchError(this.handleErrorObs)

);
```

```
}

private handleErrorObs(error: any) {

return throwError(error.message ||
error);

}

getDettaglioMetro(idt: string): Metro {

for(let i = 0; i < LISTAMETRO.length;
i++) {

if(LISTAMETRO[i].idt == idt)

{

return LISTAMETRO[i];
```

```
}

}

return

}

getPreferitiMetro(): Metro[] {

// recupero la lista dei preferiti

return;

}
```

(continua)

```
setPreferitiMetro(id:string): boolean {

// imposto la metro tra i preferiti

return;

}

}
```

metro.service.ts

L'interrogazione dell'API per il recupero del dettaglio del treno, avviene passando

all'url remoto un parametro query,

rappresentativo dell'id entificativo del treno . Ho

costruito l'url concatenando alla stringa

apitGetUrl indicata nel punto //2, la

stringa rappresentativa della query:

this.apiGetUrl + '?idt=' + idt // oppure usare i backtick

per ottenere così un url del tipo:

https://www.dcopelli.it/test/angular/metr idt=XXX

I valori restituiti dai due metodi

, ti ricordo, sono degli Observable senza tipo , e

Angular non è in grado di sapere di quale oggetti si tratti. Questo è il motivo per cui

ho chiamato il metodo get() specificando il tipo.

Al fine di poter accedere a questi oggetti, dovrò effettuare il

subscribe() ai due

metodi, quindi modificare il componente

dettaglio.component.ts, come vedremo

successivamente .

Altre sezioni interessanti

da commentare sono il

punto //1 che ci permette di

prelevare il tipo Observable, il gestore degli errori dalla libreria rxjs , oltre che gli

operatori dalla libreria

RxJS da rxjs/operators , e i l punto //3 che ci permette

d"iniettare il Service HttpClient.

Analizzando in dettaglio il metodo getListaMetroObservable(), vediamo che

restituisce un Observable, di cui è stato fatto il casting ad un array di oggetti

Metro[], da qui la notazione:

getListaMetroObservable(): Observable<Metro[]> {...}

Questo perché, come già detto più volte, http.get() nella forma che useremo noi,

restituisce sempre un Observable rappresentato da un oggetto JSON senza

tipo

,

costituito solo dalla sezione body della risposta e non da altre intestazioni.

Tale oggetto ci servirà per valorizzare la variabile locale di nome risposta.

Chiaramente è possibile richiamare il metodo

get() con ulteriori parametri

opzionali, come le intestazioni da inviare o dei parametri.

Per la lista completa di tutte le firme

ammesse per il metodo , consulta questo link:

https://angular.io/api/common/http/HttpC

L'informazione da prelevare restituita dall"API, si trova all'interno della

chiave

dati, quindi essendo la risposta un oggetto non noto ad Angular, utilizzeremo la

notazione con le parentesi quadre e non quella con il punto, tipica degli oggetti:

map(risposta => risposta[' dati'])

In alternativa, avremmo potuto creare un"interfaccia per definire questo tipo di dati,

potendo poi sfruttare la notazione del punto.

L"array di oggett i Metro così

recuperato, dovrà avere la stessa sequenza di dati progettati nel modello interno

all'applicazione, ma chiaramente queste informazioni sono stabilite da chi ha

realizzato l"API.

Analizzando la *figura 12.2.1* vista

all"inizio del capitolo, scoprirai che ci sono due

chiavi aggiuntive non previste nel modello creato in precedenza.

Pertanto dovremo integrare questo modello, andando ad aggiungere i

campi

mancanti direzione e stazione, ma lascio a te questo facile esercizio.

Nel caso ci sia un errore di comunicazione o una risposta negativa del server, tramite

cacthError() riesco a eseguire un metodo

interno al Service, che recupera

l'oggetto di tipo HttpErrorResponse e accede a proprietà interne come message,

che contiene il messaggio d"errore inviato dal server remoto.

Dovendo restituire un Observable, dovrò usare il metodo

throwError()di RxJS

per comunicare a subscribe() l'informazione da mostrare in caso di errore : il

messaggio, nel caso sia presente, oppure

l'oggetto completo. Chiaramente lo dovrò

importare dalla libreria rxjs.

return throwError(err.message ||
err.error);

Come dicevamo, il passo successivo sarà quello di integrare i metodi del Service, sia

all'interno del componente

treni.component.ts, che all''interno del componente

dettaglio.component.ts

Per il primo , le modifiche riguarderanno la creazione di un metodo per il

subcribe() dell'Observable, che restituisce la lista dei treni:

...

@Component({

selector: 'ca-treni',

template: `

<ca-menu></ca-menu>

```
<p>{{trenipartiti}}</p>

<div class=”listtreni”>

<ca-metro *ngFor="let metro of
listametro"

(inpartenza)="partiti($event)"

[datiIn]="metro"

[ora]="now"

(click)="setMetro(metro.idt)">

</ca-metro>

<div *ngIf="errormsg">{{errormsg}}
</div>
```

```
</div>`

})

export class TreniComponent
implements OnInit {

listametro: Metro[];

trenipartiti: string;

errormsg;

constructor(private router: Router,
private treniservice: TreniService) {

this.trenipartiti = '';

this.listametro = [];
```

```
this.now = new Date().getTime();

}

ngOnInit() {

this.getListaMetroObservable(); //1

}

//2

getListaMetroObservable() {

this.treniservice.getListaMetroObservabl

.subscribe(

risp => this.listametro = risp,
```

error => this.errormsg = error

);

}

(continua)

162

setMetro(id: string) {

this.router.navigate(['inarrivo/dettaglio', id]);

}

partiti(id: string) {

this.trenipartiti += "|" + id;

}

}

treni.component.ts

Nel punto //1, a

ll'interno di ngOnInit() ho r ichiamato un metodo definito

internamente alla classe, che

dovrà effettuare il subscribe()
all'Observable

restituito da getListaMetroObservable().

In questo modo, po trai usare tale valore
per valorizzare l'array listametro con

oggetti Metro da collegare all'interno
del template del componente.

risp => this.listametro = risp

In caso di errore, v

errà valorizzata una proprietà

errormsg, da mostrare nel

template, con una classica direttiva *ngIf.

error => this.errormsg = error

Ora se tu provassi l'applicazion e, utilizzando il link del server remoto visto all'inizio

del capitolo, riceveresti un errore del tipo:

Questo perch é il browser, per ragioni di sicurezza,

impedisce di effettuare delle

chiamate esterne al proprio computer . Se stai usando Chrome, puoi sc aricare

una

delle tante estensioni, che permettono di bypassare questa sicurezza.

163

Devi cercare "Allow-Control-Origin" su Chrome Web Store e installarne un a per poi

abilitarla all'occorrenza. Per maggiori informazioni puoi consultare questo articolo:

https://www.video-corsi.com/creareapp/debug_app_in_chro

Per il componente dettaglio, si sf

ruttando le stesse considerazioni

fatte in

precedenza:

....

```
import { Metro } from
'./../../../model/metro.model';

import { TreniService } from
'./../../../service/treni.service';

@Component({

selector: 'ca-dettaglio',
```

```
template: `<div *ngIf="treno">

<p> Treno Linea: {{treno.linea}} </p>

<p> ID: {{treno.idt}}</p>

</div>

<h2>Lista Chat</h2>

`


})

export class DettaglioComponent
implements OnInit {

idtreno: string;
```

```
treno: Metro;

errormsg;

constructor(private route:
ActivatedRoute,

private treniservice:TreniService) {

}

ngOnInit() {

this.idtreno =
this.route.snapshot.paramMap.get('id');

// 1)

this.getDettaglioMetroObservable(this
```

}

getDettaglioMetroObservable(idt: string) {

this.treniservice.getDettaglioMetroOb

.subscribe(

risp => this.treno = risp[0],

error => this.errormsg = error

);

}

getDettaglioMetro(idtr: string) {

```
this.treno =
this.treniservice.getDettaglioMetro(idtr);

}

}
```

dettaglio.component.ts

In questo caso, il valore restituito dall'API remota, è un array con un singolo oggetto

Metro, come visualizzato qui sotto per l'id treno “12345”:

Pertanto dovrò valorizzare la variabile treno solo con il primo elemento:

risp => this.treno = risp**[0]**

Ora che siamo riusciti a recuperare dei dati da una sorgente esterna, per mostrare sia

la lista di treni che i

l dettaglio di un treno, vediamo come si

possa eseguire

un'operazione d "invio dati in modalità POST, in modo da avere una panoramica

completa sulle tecniche per comunicare con un server remoto.

12.3 Inviare dati in modalità POST

Non appena avremo la neces

sità d "inviare delle informazioni da memorizzare,

cancellare, aggiornare, su un server remoto, entra in gioco la modalità POST, che a

sua volte può essere suddivisa nella modalità PUT e DELETE per le quali Angular

ha previsto dei metodi dedicati.

Senza scomodare troppo la fantasia e senza perdere tempo in chiacchiere, la sintassi

da usare per interrogare una sorgente di dati esterna inviando delle informazioni

dall'app Angular, è leggermente più complessa rispetto a GET, perché entra in gioco

anche la tipologia di intestazione da inviare al server.

In Angular è previsto il metodo

post() del Service Http, che come detto per il

metodo get(), presenta diverse firme. Noi useremo quella base, che ci permetterà di

inviare e ricevere i dati considerandoli nel formato JSON:

oggettoHttp.post(URL, DATI)

Come visto per get(), anche post() **restituisce un Observable** , quindi per

recuperare il dato dovremo effettuare un

subscribe(). Come vedi nulla di

trascendentale, tranne alcune considerazioni su come confezionare i da ti da inviare e

le intestazioni nel caso tu debba inviare dei dati in un formato diverso dal JSON.

Analizziamo in dettaglio i diversi argomenti da passare al metodo:

1) URL: stringa rappresentativa dell"url a cui inviare i dati.

2) DATI: nel nostro caso confezi oneremo i dati come se fossero un

oggetto

literal JavaScript, quindi nell"ipotesi d"inviare un solo campo *id* valorizzato

alla stringa „ABC", dovremmo scrivere:

{ id: 'ABC' }

Non sempre i dati da inviare e ricevere sono nel formato JSON, quindi è possibi le

usare anche la firma estesa del metodo

, che prevede un terzo parametro a cui

passare un oggetto JavaScript:

oggettoHttp.post(URL, DATI, OPZIONI)

In quest"ultimo, si possono specificare le intestazioni, il tipo di risposta che si vuole

ricevere, dei parametri aggiuntivi da inviare nell"url e tenere traccia dell"andamento

di operazioni come l"upload di un file o il relativo download.

Ad esempio, nel caso tu voglia impostare delle intestazioni HTTP particolari, d ovrai

creare un oggetto JavaScript inser endo la proprietà headers,valorizzata con un

oggetto di tipo Httpheaders.

Sfrutteremo quindi la classe HttpHeaders - che dovrà essere importata da

@angular/common/http - e tramite il metodo set(), imposteremo il tipo di

intestazione richiesto dall"API remota ed eventualmente altri dati.

Nel caso volessi impostare il „Content-Type" della richiesta, dovrai scrivere:

```
new HttpHeaders().set('Content-Type', 'application/TIPO');
```

dove TIPO è il tipo del formato di dato da inviare (es. json/xml, etc.)

L"informazione legate alle intestazioni è di fondamentale importanza quando nel

server remoto è installato PHP, in quanto i dati inviati in POST sono

tipicamente

interpretati come se arrivassero da un modulo web, quindi con un Content

-type

impostato a "application/ x-www-form-urlencoded" e non sarà possibile usare

$_POST['nomecampo'] per il recupero dei campi se il Content-type della richiesta

è diverso.

Il metodo post(), nella prima forma vista all"inizio, invierà i dati i mpostando in

automatico il „Content -type" al tipo JSON, quindi dovrai solo assicurati di creare

un"applicazione PHP in grado di recuperare i dati leggendo da *php://input*,

bypassando così il problema dell"uso di $_POST[].

Giusto per farti vedere come inserire questi dati opzionali, n el caso volessi impostare

manualmente la tipologia di contenuto da inviare, userai l a classe HttpHeaders,

per confezionare un oggetto literal JavaScript valorizzando la proprietà *headers*:

```
// 1) Oltre a HttpClient importo HttpHeaders

import {HttpClient, HttpHeaders } from '@angular/common/http';

...
```

// 2) Creo le intestazioni

instestaz = new HttpHeaders().set('Content-Type','application/json');

...

// 3) Le aggiungo come terzo parametro OPZIONALE

oggettoHttp.post(URL, {'id': 'ABC'}, **{'headers':intestaz}**)

Vediamo come si possano applicare questi concetti all'applicazione MetroChat.

Quali dati dovremo inviare tramite

l'applicazione?

Ebbene, dobbiamo aggiungere una nuova sezione, quella delle chat, per poter capire

cosa ci serve. Infatti come puoi osservare dall'immagine qui sotto, l'utente che clicca

in corrispondenza al treno, accede alla sezione dell'app, che mostra l'elenco dei

messaggi scambiati tra tutti gli utenti di quel particolare treno e questo per ogni treno

selezionato.

Chat

Metro Chat

Linea Rossa (ID: 12345)
Direzione: Centraal Station
15 A2

Oggi è il mio primo giorno di lavoro e ho dimenticato la merenda!

Se vuoi ti porto un panino con il salame!

Ciao Martina, piacere di...incontrarti. Come stai?

Ciao Angela, che dici se ci vediamo sta sera, dopo lezione?

Messaggio

167

168

E' chiaro che, per poter inviare il nostro messaggio, dobbiamo avere un campo input

dove inserire l'informazione e, sfruttando il metodo

post(), inviarla in remoto al

fine di memorizzarla nel database e renderla accessibile a tutti.

Le informazioni che dovrò memorizzare nell'ipotetico database rem

oto, dovranno

essere:

- il testo del messaggio

- l"identificativo dell"utente che lo sta inviando

- e l'identificativo del treno associato alla chat

Per ora ipotizzeremo che l"identificativo dell"utente sia noto, ma nella realtà dovrai

recuperarlo dopo che l"utente ha effettuato il login nell"apposita pagina.

Se nel server remoto, fosse presente PHP, un possibile programma per il recupero

delle informazioni passate in formato JSON potrebbe essere:

```
<?php

$json = file_get_contents('php://input');

$mypost = json_decode($json,true);

// recupero i dati passati da Angular

$idt = $mypost['idtreno'];

$messaggio = $mypost['messaggio'];
```

```
// TODO: salvo i dati nel database e poi...

// ....

// mostro una stringa nel formato JSON con l'id recuperato

header('Content-Type: application/json');

echo '{"dati":' . $idt . '}';
```

index.php

dove dovrei aggiungere tutte le righe per il salvataggio dei dati sul database.

La stringa restituita dal programma PHP

è in formato JSON, quindi potrà essere

sfruttata dall'applicazione Angular per mostrar

e un messaggio all'utente o un

eventuale errore.

I due dati che dovr

ai confezionare, sono indicati con le proprietà

idtreno,

idutente - per ora inserito manualmente al valore 99 - e messaggio.

Nell'ipotesi

tu

abbia recuperato le informazioni

idtreno='AFG'

e

messaggio='Messaggio di test', dovrai scrivere:


```
const dati = {

idtreno: 'AFG',
```

idutente: '99'

messaggio: 'Messaggio di test'

}

Siamo pronti pe r aggiungere la funzionalità d"invio dati ad un nuovo Service, che

gestirà tutta la sezione Chat della nostra applicazione.

Possiamo ricopiare gran parte delle righe dell' equivalente servizio usato per

mostrare la lista dei treni, aggiungendo anche il metodo

sendMsgChatObservable(obj) creato con le nozioni viste fino ad ora.

Il nuovo Service lo potremmo chiamare *chat.service.ts*, e si occuperà di visualizzare

l'elenco dei messaggi remoti e d "inviare il messaggio all'applicazione remota, quindi

dovremo specificare i due url dell'API che ci permetteranno di ottenere questo.

Il codice del Service, sarà:

...

import { HttpClient, HttpHeaders } from

'@angular/common/http';

import { throwError, Observable } from
'rxjs';

import { catchError, map, tap } from
'rxjs/operators';

@Injectable({providerIn: ' root' })

export class ChatService {

// 1

private apiGetUrl =
'https://www.dcopelli.it/test/angular/ch

private apiPostUrl =
'https://www.dcopelli.it/test/angular/ch

// 2

constructor(private http: HttpClient) {}

getListaChat(): Chat[] {

// recupero i dati statici per ora

return LISTAMSG;

}

// 3

getListaChatObservable(idt: string): Observable<Messaggio[]> {

return this.http.get<Messaggio[]> (this.apiGetUrl + '?idt='+idt)

```
.pipe(

map(risposta=>risposta['dati']),

catchError(this.handleErrorObs));

}
```

(continua)

```
// 4

sendChatMsgObservable(obj:
Messaggio): Observable<Messaggio>
{
```

```
return this.http.post<Messaggio>(

apiPostUrl,

{ idtreno: obj.idt,

idutente: obj.idu,

messaggio: obj.testo },

)

.pipe(

map(risposta => risposta['dati']),

catchError(this.handleErrorObs));

}
```

```
// 5

private handleErrorObs(error:any) {

return throwError(error.message ||
error);

}

}
```

chat.service.ts

dove al posto dell'oggetto Metro, ho s ostituito l'oggetto Messaggio, il cui modello

messaggio.model.ts, era stato progettato quando abbiamo parlato di modellazione

dati.

Le parti interessanti sono quelle indicate con i punti //3 e //4.

```
getListaChatObservable(idt: string): Observable<Messaggio[]> {

return this.http.get<Messaggio[]> (this.apiGetUrl + '?idt='+idt)

.pipe(

map(risposta=>risposta['dati']),

catchError(this.handleErrorObs));

}
```

L'interrogazione dell'API per il recupero dei messaggi

presenti ne lla chat di un

particolare treno, deve essere fatta inviando l'identificativo del treno e l"identificativo

dell"utente. Questo al fine di evidenziare eventuali messaggi inseriti tra i preferiti

dell"utente di test con id ,,99"; ho costruito pertanto manualmente l'url, concatenando

alla stringa apitGetUrl, la stringa rappresentativa della query, per ottenere:

https://www.dcopelli.it/test/angular/chat
idt=XXX&idu=99

Questa tecnica l"abbiamo già vista quando abbiamo recuperato i dati del treno. In

alternativa potresti passare questo p arametro al metodo get(), sotto forma di dato

opzionale, sfruttando la classe HttpParams e il metodo set().

La classe dovrà essere importata sempre da @angular/common/http:

const param = new HttpParams().set('idt',idt).set('idu', '99');

e inserendo il dato all"interno del metodo get() in questo modo:

getListaChatObservable(idt: string): Observable<Messaggio[]> {

return this.http.get<Messaggio[]>(

this.apiGetUrl,

{params: this.param}

).pipe(

map(risposta => risposta['dati']),

```
catchError(this.handleErrorObs)

);

}
```

Per quanto riguarda invece l"invio del messaggio, la cosa interessante da osservare è

che i l valore restituito da sendChatMsgObservable(), è un Observable di tipo

Messaggio, che po

trò usare per

recuperare l"identificativo del

messaggio

memorizzato nel server remoto e restituito dall"API.

Ora che abbiamo terminato il Servic e che si occ uperà di tutto ciò che riguarda i

messaggi della chat, vediamo come possa essere usato all'interno dell'applicazione.

Potresti creare una pagina dedicata alla lista dei messaggi chat, ma, avendo già

sviluppato una pagina dettaglio, potresti agganciarti a quest"ultima.

L"idea è di sfruttare questa pagina per visualizzare, sotto ai dati del dettaglio del

treno, la lista dei messaggi chat che i vari utenti si stanno scambiando in quel treno

(vedi figure a pag. 2).

Il codice sviluppato per il componente di dettaglio, si limitava a mostrare alcuni dati

del treno, prelevati tramite il metodo

getDettaglioMetroObservable() del

Service *MetroService*.

```
...

import { Metro } from
'./../../../model/metro.model';

import { TreniService } from
'./../../../service/treni.service';

@Component({

selector: 'ca-dettaglio',

template: `<div *ngIf="treno">
```

(continua)

```
<p> Treno Linea: {{treno.linea}} </p>

<p> ID: {{treno.idt}}</p>

</div> `

})

export class DettaglioComponent
implements OnInit {

idtreno: string;

treno: Metro;

errormsg;

constructor(private route:
ActivatedRoute,
```

```
private treniservice:TreniService) {

}

ngOnInit() {

this.idtreno =
this.route.snapshot.paramMap.get('id');

this.getDettaglioMetroObservable(this.id

}

getDettaglioMetroObservable(idt:
string) {

this.treniservice.getDettaglioMetroObser

.subscribe(
```

```
risp => this.treno = risp[0],

error => this.errormsg = error);

}

getDettaglioMetro(idtr: string) {

this.treno =
this.treniservice.getDettaglioMetro(idtr);

}

}
```

dettaglio.component.ts

Quello che potremmo fare è aggiungere a lla visualizzazione di queste

informazioni,

anche la lista di tutti i messaggi scambiati in chat dagli utenti di quel treno.

Questo ci permet terà di capire come si possano creare e gestire N componenti che

comunicano con http, all'interno di uno stesso componente padre.

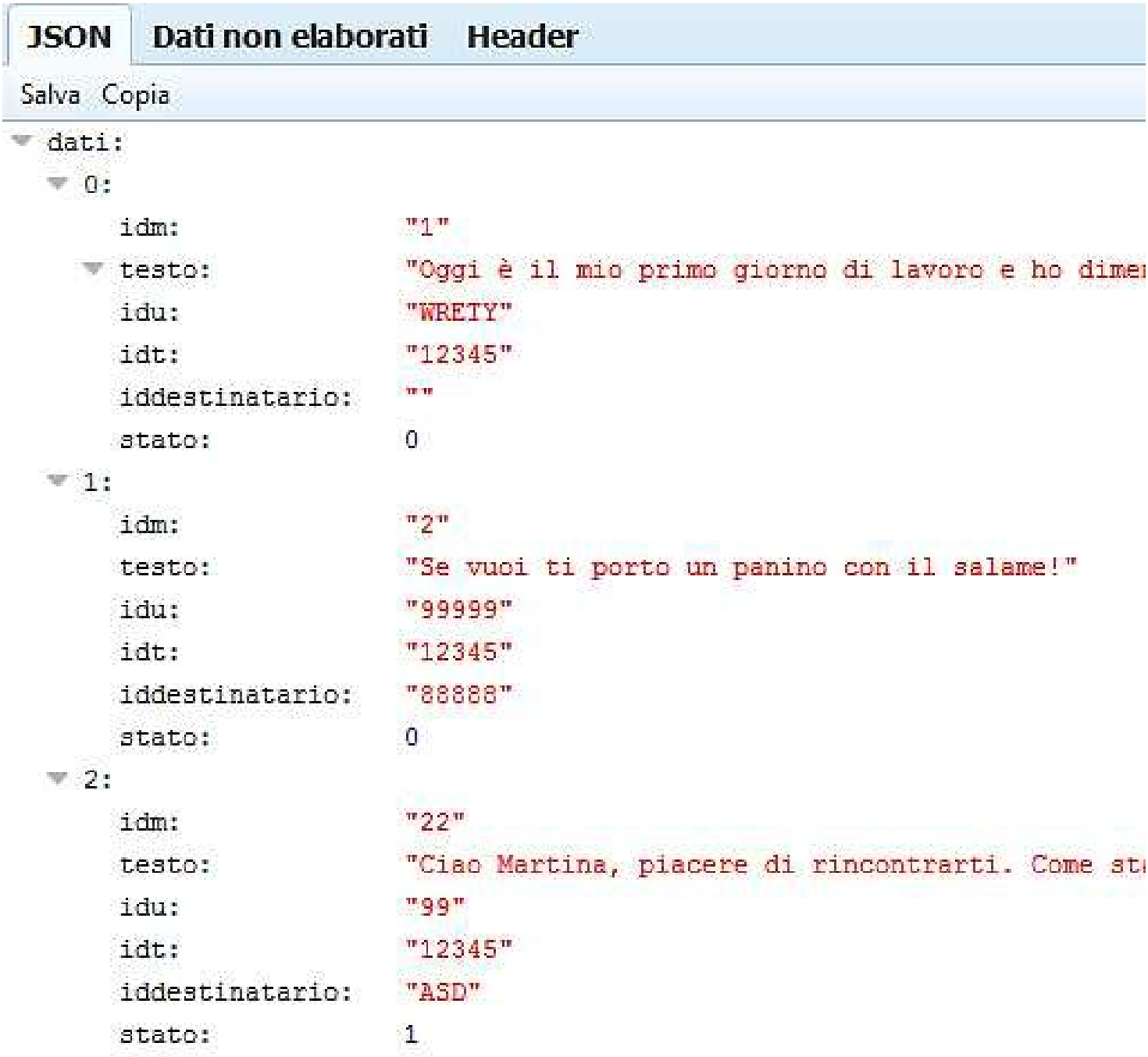
I messaggi restituiti dall'API potrebbero avere il formato indicato qui sotto (array di

oggetti Messaggio), in cui ancora non

tutte le proprietà del singolo messaggio

vengono utilizzate dall'applicazione, ma che potrebbero essere sfruttate per ampliare

le funzionalità di questa.



173

Ad esempio, l a proprietà idu, dovrebbe far parte dell'applicazione non appena avrai

inserito anche u

na sezione di login

, al fine di identificare l'utilizzatore

dell'applicazione mentre iddestinatario, nel caso tu vo glia inviare dei messaggi

privati solo a quel particolare utente.

Anche la proprietà stato non è stata ancora inserita nel modello dei dati e ci servirà

per contrassegnare un messaggio appena inserito nella lista dei preferiti di quel

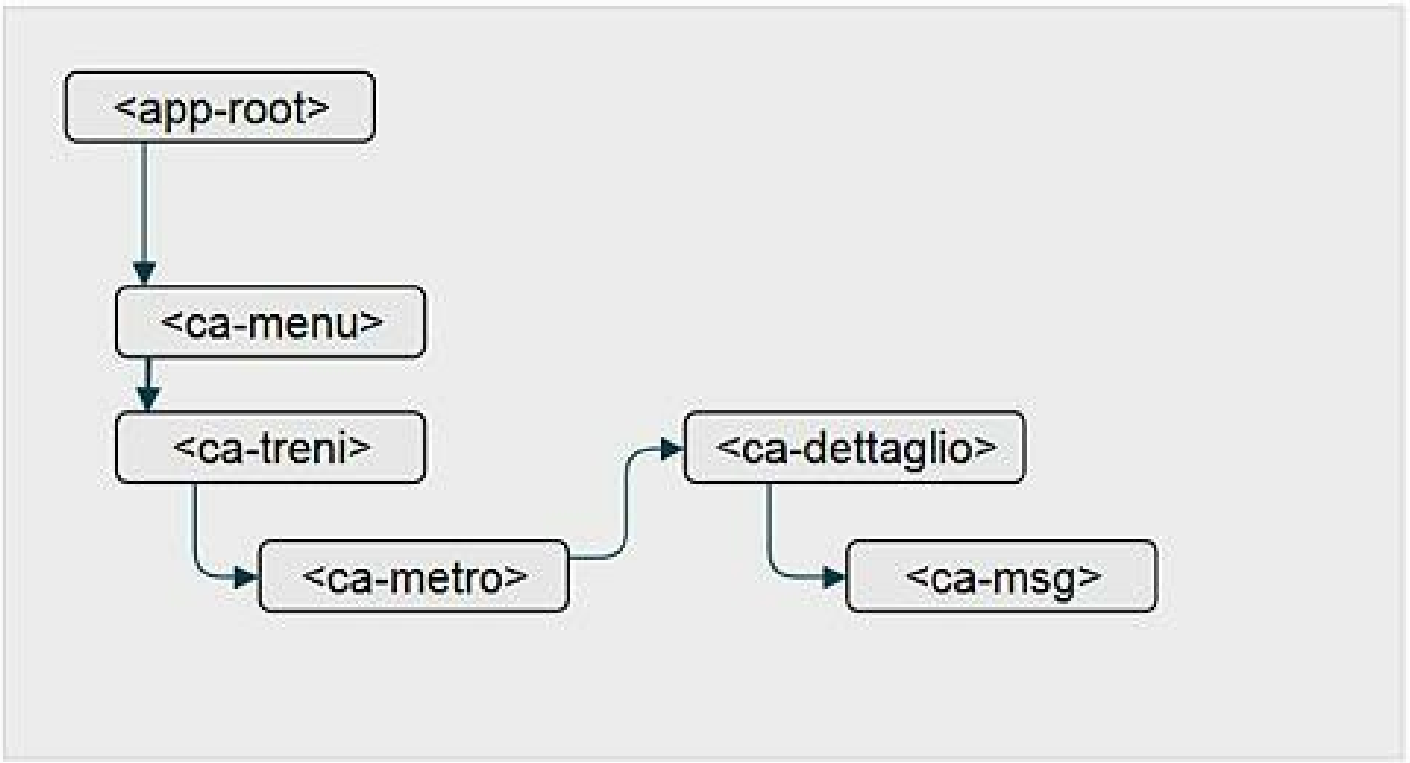
particolare utente.

Così come fatto per la lista treni, potremmo creare un componente dedicato per

visualizzare la lista dei messaggi, quindi, un componente "inte

lligente" che riceva

dei dati in ingresso.



[redacted]

<ca-chat>

A livello di selettore, potremmo inserirlo nel componente dettaglio con

una

notazione di questo tipo:

```
template: `

<div *ngIf="treno">

<p> Treno Linea: {{treno.linea}} </p>

<p> ID: {{treno.idt}}</p>

</div>

<ca-chat *ngFor="let chat of listachat"
[msgIn]="chat"></ca-chat>

`
```

che ricorda quello visto per il componente

metro.component.ts, dove listachat,

sarà un array valorizzato con il metodo

getListaChatObservable(idt)

progettato per il service chat.service.ts

Il componente dettaglio pertanto diventa:

```
import { Component, OnInit } from '@angular/core';

import { ActivatedRoute, ParamMap } from '@angular/router';
```

```
import { Metro } from
'./../../../model/metro.model';

import { TreniService } from
'./../../../service/treni.service';

import { Messaggio } from
'./../../../model/messaggio.model';
```

(continua)

```
import { ChatService } from
'./../../../service/chat.service';

@Component({
```

```
selector: 'ca-dettaglio',

template: `

<div *ngIf="treno">

<p> Treno Linea: {{treno.linea}} </p>

<p> ID: {{treno.idt}}</p>

</div>

<ca-chat *ngFor="let chat of
listachat" [msgIn]="chat"></ca-chat>

`


})
```

```
export class DettaglioComponent
implements OnInit {

idtreno: string;

treno: Metro;

errormsg;

listachat: Messaggio[];

// 1)

constructor(

private route: ActivatedRoute,

private treniservice: TreniService,
```

```
private chatservice: ChatService) {

}

ngOnInit() {

this.idtreno =
this.route.paramMap.get('id');

this.getDettaglioMetroObservable(this.id

// 2)

this.getListChatObservable(this.idtren

}

getDettaglioMetroObservable(idt:
string) {
```

```
this.treniservice.getDettaglioMetroObser
.subscribe(
risp => this.treno = risp[0],
error => this.errormsg = error);
}
getListChatObservable(idt: string) {
this.chatservice.getListaChatObserval
.subscribe(
risp => this.listachat = risp,
error => this.errormsg = error);
```

```
}

getDettaglioMetro(idtr:string) {

this.treno =
this.treniservice.getDettaglioMetro(idtr);

}

}
```

dettaglio.component.ts

L'unica os servazione da fare, è la presenza del parametro

idt del metodo

getListChatObservable() che corrisponde proprio all'id recuperato dall'url

dell'applicazione e trasmesso al server remoto al fine di selezionare solo i messaggi

chat corrispondenti a quel treno.

Infine, il componente con selettore

<ca-chat>, dovrà ricevere in ingresso un

parametro, quindi lo progetterò con le tecniche che già conosco. Lo posso salvare

all'interno della cartella *chat*, con il classico nome *chat.component.ts*

```
import { Component, OnInit, Input } from '@angular/core';

import { Messaggio } from './../model/messaggio.model';

@Component({

selector: 'ca-chat',

template: `<p>{{msgIn.idu}}-{{msgIn.testo}} </p>`

})

export class ChatComponent implements
```

OnInit {

@Input() msgIn: Messaggio;

constructor() {}

ngOnInit() {}

}

chat.component.ts

Il parametro d"ingresso è stato chiamato msgIn ed è di tipo Messaggio, in modo da

poter visualizzare le singole proprietà come idu e testo.

Il componente padre, potrebbe subire solo una piccola variazione nel testo mostrato

all'utente, che invece di essere "Dettaglio Treno", diventerebbe "Chat Treno".

Eventualmente potresti anche cambi

argli il nome, visto che

dettagliotreno.component.ts potrebbe creare confusione.

Lascia a te questo come utile esercizio, perc

hé spesso vedrai ti capiterà di dover

cambiare dei nomi ad un componente.

Bene, siamo riusciti a completare l'applicazione in modo che ora i dati siano

visualizzati prelevandoli da una sorgente esterna. Ci rimane da vedere come dare la

possibilità all'utente di inviare un nuovo messaggio, ma questo lo tratteremo quando

parleremo delle basi dei moduli web.

12.4 Aggiornare dati in PUT

Oltre a POST e GET, Angular offre la possibilità di inviare i dati con intestazioni

specifiche di modifica dati. La sintassi è sostanzialmente identica a quella di POST:

oggettoHttp.put(URL, DATI);

Nella forma base che useremo noi,

il metodo restituisce un

Observable di tipo

Object che può essere usato con tutte le tecniche viste fino ad ora.

I dati sono

sempre interpretati nel formato JSON.

Un utile esercizio potrebbe essere quello di implementare un pulsante di "Aggiungi

ai preferiti", in corrispondenza ai messaggi inviat

i in chat, in modo da salvare

all'interno di una pagina specifica

“preferiti”, la lista delle persone che

hai

individuato per eventuali future chat.

In questo modo accedendo alla pagina "preferiti" potresti visualizzarli sotto forma di

elenco, oltre che avere la possibilità di cancellarli, azione che impareremo a fare con

il metodo delete().

E' chiaro che ogni utente potrà avere la propria lista di preferiti, pertanto ipotizzando

che il parametro idutente sia noto (es. 99), i dati che potre sti memorizzare nel

database remoto dopo il click sono, idutente, idmessaggio e lo stato.

{idutente: '99', idmessaggio: 'AAA', stato:1}

All'interno del service

chat.service.ts, potresti aggiungere il metodo

setChatPreferiti(), che riceve in ingresso i tre parametri da salvare, e

restituisce un Observable di tipo number, che ci servirà per indicare se il

messaggio

dovrà apparire con a fianco un'icona selezionata, oppure deselezionata, a seconda del

valore restituito 1 o 0:

setChatPreferiti(idu: string,idm: string,stato: number): Observable<number> {

return **this.http.put<number>** (this.apiPreferitiUrl,

{idutente: idu, idmessaggio: idm, stato: stato}

).pipe(

map(risposta => risposta['stato']),

catchError(this.handleErrorObs));

}

178

Chiaramente apiPreferitiUrl sarà valorizzata con l"url dell"API predisposta per

effettuare il salvataggio dei dati (vedi appendice).

Così facendo, all'interno del

componente <ca-chat>, potrei aggiungere il pulsante per salvare il

messaggio tra i

preferiti, oppure un'icona che cambia stato e quindi colore, esattam

ente come fatto

per la lampadina, quando abbiamo parlato di direttive.

Iniettando il Service nel costruttore e aggiungendo le relative righe di

import,

ottengo:

import { Component, OnInit, Input } from '@angular/core';

```
import { Messaggio } from
'./../model/messaggio.model';

import { ChatService } from
'./../service/chat.service';

@Component({

selector: 'ca-chat',

template: `<p>{{msgIn.idu}}-
{{msgIn.testo}}

<span *ngIf="msgIn.stato==1; else
show"

(click)="setMsgPreferiti(msgIn.idm,0)">

<i class="material-icons
```

```
on">favorite</i>

</span>

<ng-template #show>

<span
(click)="setMsgPreferiti(msgIn.idm,1)">

<i class="material-
icons">favorite_border</i>

</span>

</ng-template>

</p>

<p>{{errormsg}}</p>`
```

```
})

export class MsgComponent implements
OnInit {

@Input() msgIn: Messaggio;

errormsg;

// Inietto il Service

constructor(private
chatservice:ChatService) {

}

ngOnInit() {

}
```

```
setMsgPreferiti(idm, newstato) {

// recupero lo stato attuale, per
reimpostarlo in caso di errore

const statoprec = this.msgIn.stato;

this.msgIn.stato = newstato;

this.chatservice.setChatPreferiti('99',
idm, newstato)

.subscribe(

risp => {if (risp!=newstato) {

// risposta negativa del server

alert("Errore");
```

```
this.msgIn.stato = statoprec;
```

(continua)

179

```
} else {
alert("Ok");
this.msgIn.stato = newstato;
}},
error => this.errormsg = error);
}
```

}

chat.component.ts

dove nel template ho aggiunto un gestore di evento sul click di nome

setMsgPreferiti() passandogli in ingresso l' id del messaggio da inserire tra i

preferiti e lo stato (1= aggiunta; 0=eliminazione)

Il metodo del service, invierà questi dati in modalità PUT, insieme all'ipotetico

identificativo dell"utente di test (99), al fine di memorizzarli nel database o di

rimuoverli.

Per cambiare lo stat o dell'icona associata ai preferiti, senza aspettare la risposta del

server, ho valorizzato la proprietà stato dell"oggetto msgIn, con il valore passato al

metodo e al ricevimento del dato effettivo, mostro un messaggio di

alert() per

verificare se la stato inviato dal server, coincide con quello impostato.

In caso di errore, reimpost

o il valore di

stato a quello precedentemente

memorizzato in statoprec, per evi tare che l'utent e veda un messaggio selezionato

quando nel server non lo è, a seguito di un errore.

12.5 Cancellare i dati con DELETE

Altra operazione spesso necessaria è la cancellazione dei dati. Anche qui Angular

offre un metodo specifico delete() che ha la stessa sintassi del metodo get():

oggettoHttp.delete(URL)

Questa è la forma base, ma anche in questo caso esiston

o numerose firme, che

prevedono l''inserimento di parametri opzionali.

Lascio a te fare delle sperimentazioni, ora che penso tu abbia capito come sfruttare le

tecniche di recupero e invio informazioni.

Come dicevo in precedenza, un possibile sviluppo dell'app che stiamo

progettando, è

quello di aggiungere la sezione "Preferiti", che ci dia la possibilità di mostrare

180

l'elenco dei messaggi contrassegnati in precedenza, oltre che la possibilità di

cancellarli dalla lista dei preferiti.

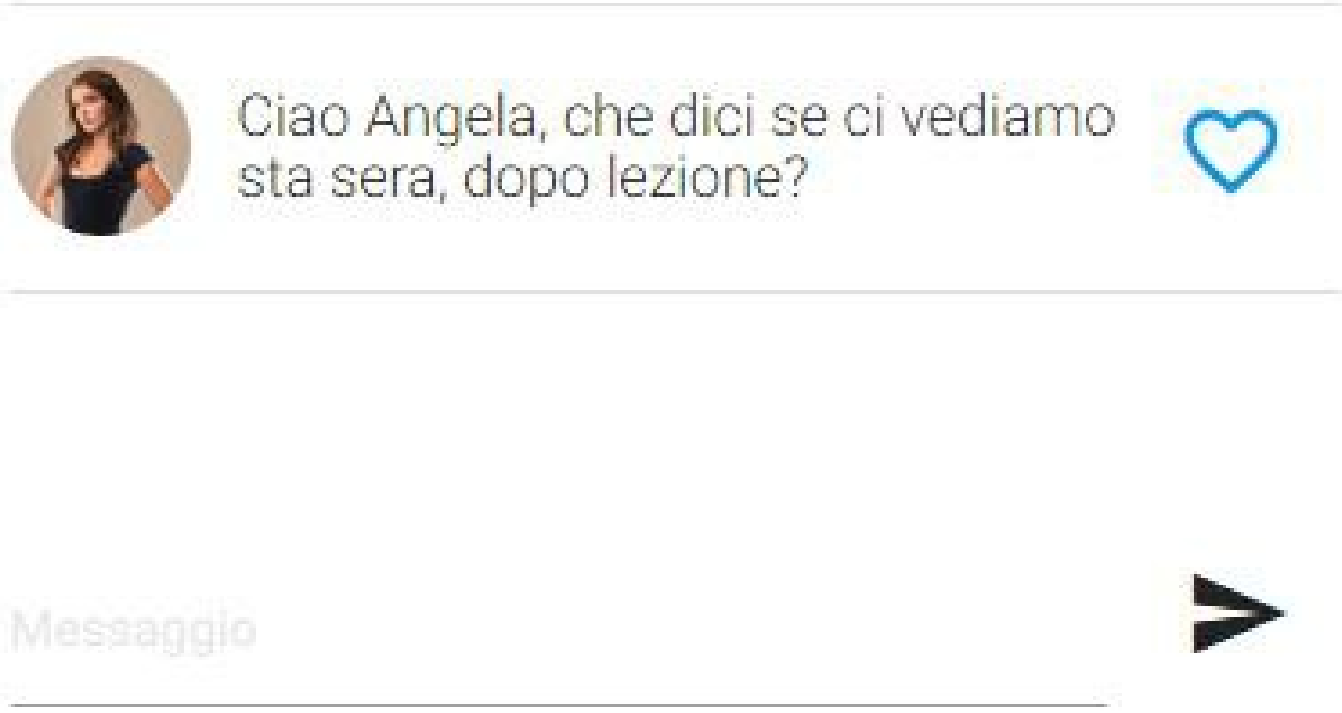
Lascio a te lo sviluppo c ome utile esercizio per capire se tutti i concetti spiegati fino

ad ora sono stati correttamente assimilati.

Ti consiglio di partire con dei dati locali per poi passare ad un

a eventuale API

remota.



Le basi dei form

13.1 I Form in Angular: introduzione

Come ogni sit o web che si rispetti, anche un'applicazione Angular necessita di

interagire con l'utente.

Questo avviene tipicamente o sotto forma di tocco /click, o sotto forma di dati da

inserire in un campo di testo

, ad esempio per effettuare delle ricerche o per

trasmetterli ad un'API in grado di comunicare con un database remoto.

Nell'attesa che si possa interagire con le proprie applicazioni grazie a comandi

vocali, vediamo come si possa aggiungere il classico modulo web

all"interno di un

componente, al fine di recuperare delle informazioni necessarie ad interagire con

l'applicazione.

Nel caso dell'app che stiamo progettando, l'unica sezione che richiede

l'inserimento

di dati da parte dell"utente, è quella d "invio del messaggio nella chat, quindi un solo

campo di input che potrebbe essere gestito con una variabile locale nel template.

La gestione dei form in Angular è molto flessibile, in quanto è possibile scegliere

due strade completamente differenti, che dovranno essere v

alutate sulla base della

complessità del modulo.

La prima strada, quella chiamata "Template Driven" , va bene quando il modulo web

è costituito da uno o due campi e non sono necessarie complesse azioni di verifica

dati.

La seconda, chiamata "Reactive Form"

, va bene quando hai dei moduli web

"complessi" ossia , non solo costituiti da

diversi campi, ma con la necessità di

verifica di coerenza dei dati basata su più fattori collegati.

Entrambe le strade offrono vantaggi e svantaggi, ma se provieni dallo sviluppo di siti

web, forse la prima è quella che più si avvicina al tuo modo di lavorare. Con il tempo

vedrai che la seconda è più immediata da applicare.

In ogni caso,

ti fornirò le

basi per l"uso della prima tecnica,

lasciando

l"approfondimento della seconda a specifici tutorial che pubb licherò prossimamente

nell"area online.

Vediamo prima di tutto a cosa fanno riferimento le due terminologie:

1. **Gestione Form "Template Driven"**

– In questo caso, l

a logica di

interazione con i diversi campi è gestita in gran parte n

el template, così

come avviene per le classiche pagine web

2. **Gestione Form "Reactive Form"**

– In questo caso, l a logica è gestita

direttamente con proprietà interne al corpo della classe del componente.

13.2 Cosa verificare prima di usare i Form

Al fine di p oter usare le funzionalità

offerta da Angular legate ai

moduli web , il

primo passo da fare è andare a verificare/modificare il file

app.module.ts presente

nella cartella *app* del progetto, aggiungendo le righe per poter importare due moduli

dal package @angular/core.

Nel caso volessi implementare il form usando il modello "Template Driven", al file

app.module.ts dovrai aggiungere il modulo FormsModule.

Nel file *app.module.ts* dell"applicazione pertanto, dovrai scrivere:

```
import { BrowserModule } from
'@angular/platform-browser';
```

(continua)

```
import { NgModule } from
'@angular/core';
```

```
// 1) riga per importare il Modulo
FormModule

import { FormsModule } from
'@angular/forms';

import { AppComponent } from
'./app.component';

@NgModule({

declarations: [

AppComponent

],

imports: [
```

BrowserModule,

// 2) rendo disponibile il modulo a tutta l'app

FormsModule,

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

app.module.ts

Se hai creato il progetto sfruttando la linea di comando, queste righe potrebbero

essere già state inserite per te. Parlo al condizionale, perché in alcune versioni di CLI

potrebbe non accadere. Dovrai quindi verificare che ci siano,

o procedere con

l"inserimento nel caso mancassero.

Nel caso volessi invece

sfruttare la tecnica “Reactive” , dovrai inserire un altro

modulo, che prende il nome di ReactiveFormsModule.

```
import { BrowserModule } from '@angular/platform-browser';

import { NgModule } from '@angular/core';

// 1) riga per importare il Modulo FormModule

import { ReactiveFormModule } from '@angular/forms';

import { AppComponent } from './app.component';

@NgModule({
```

declarations: [

AppComponent

],

imports: [

BrowserModule,

// 2) rendo disponibile il modulo a tutta l'app

```
],
providers: [],
bootstrap: [AppComponent]
})
export class AppModule { }
```

app.module.ts

NB: Ricordati che l'operazione d’aggiunta di un nuovo modulo

all'interno

dell'array indicato dalla proprietà import, deve essere sempre accompagnata

dalla riga di importazione della relativa classe.

13.3 I passi per creare il Form

Così come avviene per una classica pagina web, se dovessi progettare un semplice

modulo web che richiede l'inserimento del nome e dell'email, potresti scrivere:

<form action="" method="post">

```
<label for="nome">Nome</label>

<input type="text" name="nome"
value="" placeholder="Nome">

<label for="email">Email</label>

<input type="email" name="email"
value="" placeholder="Email">

<button type="submit">Invia</button>

</form>
```

In Angular non è più necessario utilizzare l'attributo *action* e *method*, perché il tutto

sarà gestito da funzionalità interne alla

classe del componente. Analogo discorso per

l'attributo *value* di ogni campo.

In Angular infatti, ogni modulo web, è gestito da una rappresentazione indipendente

dall'interfaccia, chiamata "Form Model", che consiste di tre blocchi fondamentali:

- FormControl
- FormGroup
- FormArray

Grazie all'uso di opportune direttive, sar

ai in grado di collegare gli elementi del

DOM con questo modello e gestire le operazioni di recupero delle informazioni così

come la visualizzazione nel form.

La prima modifica allora che dovrai fare, rispetto alla classica progettazione di un

modulo per una pagina web, sarà l"eliminazione degli attributi citati sopra:

<form>

```
<label for="nome">Nome</label>

<input type="text" name="nome"
placeholder="Il tuo nome">

<label for="email">Email</label>
```

(continua) 185

```
<input type="email" name="email"
placeholder="La tua email">

<button type="submit">Invia</button>

</form>
```

A questo punto dobbiamo capire come collegare gli elementi del DOM

, con il

modello usato da Angular per la gestione del form.

13.4 Gestire i dati da un campo <input>

Se nel mondo delle pagine web, il recupero dei dati da un modulo avviene inviandoli

ad un'applicazione scritta in qualche linguaggio come PHP,

Ruby,Python,ASP.net,NodeJS, nel caso di Angular devi dimenticarti tutto ques

to,

perché i dati sono recuperati direttamente nel client, e poi eventualmente inviati

all'applicazione remota.

Come già detto, non ci occuperemo di realizzare un'applicazione stile “WhatsApp”,

ma di capire come avviene il processo d''invio dei dati dai campi di un modulo web.

NB: E' necessario prestare attenzione alla posizione in cui stai

recuperando/visualizzando i dati del modulo, perché dovrai usare notazioni

diverse a seconda tu stia lavorando sul template o sul corpo della classe.

Il form di partenza è quella visto in precedenza, a cui dovremo aggiungere un gestore

di evento per l'azione di

submit e un'opportuna sintassi per poter recuperare lo

specifico campo inviato e mostrato, sia nel template che nel corpo della classe.

La prima cosa da fare è tras

formare ogni campo del modulo in un oggetto

"intelligente", in particolare in un oggetto

FormControl, sfruttando la direttiva

ngModel, che è automaticamente disponibile all'interno di Angular, senza dover fare

alcuna importazione.

La sintassi da usare per un generico campo input, sarà:

<input type="text" name="nome" ngModel>

o in modo del tutto equivalente:

<input type="text" name="nome" [ngModel]>

NB: Non devi assolutamente dimenticarti di inserire l'attributo *name* per ogni

campo del form.

Vedremo c he in realtà la direttiva offre la possibilità di

collegare in entrambe le

direzioni il campo, quindi non solo per leggerne il contenuto (dal template al

componente), ma anche di impostarlo (dal componente al template).

Con la notazione indicata sopra, riusciamo a collegare una proprietà del template con

il componente, ma non viceversa.

Per poter leggere e settare un dato, dovremo "incastrare" la direttiva all'interno della

notazione detta **"banana in a box" ,** perché i simboli usati, ossia le parentesi quadre

e tonde, assomigliano a una scatola [ ] con all'interno due banane rappresentate

dalle

perentesi tonde ( ).

Tradotto in termini pratici, dovrai scrivere:

<input type="text" name="nomecampo" **[(ngModel)]="nomecampo_modello"** > La notazione in realtà fa rif

erimento a qualcosa che abbiamo già visto

, ossia i l

collegamento ad una proprietà (che viene fatto con le parentesi quadre), e la gestione

di un evento (che viene fatto con le parentesi tonde

). Il tutto sotto il controllo di

Angular e grazie a ngModel.

La riga precedente potrebbe essere infatti scritta così:

<input type="text" name="nomecampo"

[value]="nomecampo_modello"

(input)="nomecampo=$event.target.value

Fai attenzione che nomecampo_modello è il riferimento usato nel corpo della

classe per **assegnare** al campo un valore, mentre nomecampo, è il valore usato nel

corpo della classe per **recuperare** il valore.

Nel template invece, si fa sempre riferimento a nomecampo_modello. Spesso si fa

confusione perché vengono fatti coincidere, ma sono due soggetti diversi.

Fatte queste modifiche, dobbiamo procedere con l'ultimo passo, ossia capire come

inviare queste informazioni, visto che nel modulo è stato tolto l'attributo action.

Per fare questo , si aggiunge al tag <form> un riferimento, rappresentato da una

variabile locale con la classica notazione #variabile, valorizzata con ngForm e un

gestore di evento ngSubmit, a cui passerò tale variabile.

Pertanto il modulo web precedente diventerà:

```
<form #mioform="ngForm"
(ngSubmit)="invio(mioform)" >

<label for="nome">Nome</label>

<input type="text" name="nome"

placeholder="Il tuo nome"

[(ngModel)]="nome_modello">

<label for="email">Email</label>

<input type="email" name="email"

placeholder="La tua email"

[(ngModel)]="email_modello">
```

<button type="submit">Invia</button>

</form>

Le cose più interessanti da notare sono:

1. la **presenza in ogni campo**

della direttiva ngModel che, come detto,

permette di "potenziare" og ni campo del form trasformandolo in un oggetto

FormControl.

2. la presenza di una **variabile template** assegnata a l tag <form>, chiamata a

mio piacere #mioform, valorizzata con

ngForm. In questo modo abbiamo un

riferimento a tutti i campi "potenziati" del modu

lo.

3. la presenza della notazione (evento)="gestore" che abbiamo già visto

quando abbiamo parlato di eventi in Angular. In questo caso l'evento è

ngSubmit (già predefinito) e il gestore di tale evento è un metodo che dovrò

progettare nel corpo della classe

del componente. Osserva il nome

del

parametro di ingresso, che coincide proprio con quello scelto per la variabile

template che referenzia il form.

In linea di massima, con queste tre semplici modifiche, sei in grado di trasformare un

classico modulo web, adatto per un sito html, in un modulo web in grado di essere

gestito in Angular.

Completiamo il codice, andando a creare un semplice componente di test, dal nome

formiscriviti.component.ts, per capire come poter leggere i dati inseriti

dall'utente nei due campi di input.

```
import { Component } from
'@angular/core';

import { NgForm } from
'@angular/forms';

@Component({
```

```
selector: 'ca-form-iscriviti',

template: `

<form #mioform="ngForm"
(ngSubmit)="invio(mioform)" >

<label for="nome">Nome</label>

<input type="text" name="nome"

placeholder="Il tuo nome"

[(ngModel)]="nome_modello">

<label for="email">Email</label>

<input type="email" name="email"
```

```
placeholder="La tua email"

[(ngModel)]="email_modello">

<button type="submit">Invia</button>

</form>

<p>Stai scrivendo nel campo nome:
{{nome_modello}}</p>

<p>Stai scrivendo nel campo email:
{{email_modello}}</p>

`


})

export class FormIscrivitiComponent {
```

```
invio(form: NgForm) {

alert("Tutto il form: " +
JSON.stringify(form.value));

console.log("Nome inserito è " +
form.controls['nome'].value);

console.log("Email inserita è " +
form.controls['email'].value);

}

}
```

form-iscriviti.component.ts

Come puoi osservare, prima di tutto ho importato dal package

@angular/form,

NgForm, e poi ho creato il metodo invio(), che riceve in ingresso proprio il

riferimento alla variabile locale del modulo, quindi un oggetto ngForm, che posso

usare per recuperare i singoli campi, sfruttando la notazione:

oggetto.controls['nomecampo'].value

dove nomecampo è il nome dell'

attributo "name" associato all'elemento del

modulo nel DOM.

Quest"ultimo sarà proprio il nome del corrispondente oggetto FormControl creato

da Angular, che potrai usare nel corpo della classe, per il recupero effettivo del dato

inserito dall"utente.

```
▼<ca-form-iscriviti _ngcontent-c0>
  ▼<form novalidate class="ng-pristine ng-invalid
  ng-touched">
      <label for="nome">Nome</label>
      <input name="nome" placeholder="Il tuo nome"
      required type="text" ng-reflect-required ng-
      reflect-name="nome" class="ng-pristine ng-
      invalid ng-touched">
```

Qui sotto puoi osservare come Angular vada a modificare l'elemento del DOM, con

l'aggiunta non solo di

ng-reflect-name, coincidente proprio con il nome

assegnato all'attributo name, ma anche con l'aggiunta di una serie di attributi di

classe, che potranno essere sfruttati per cambiare l''aspetto del campo, in funzione

dello stato in cui si troverà.

Nota anche la presenza nel template, di

questi due paragrafi:

Stai scrivendo nel campo nome:
{{nome_modello}}

Stai scrivendo nel campo email:
{{email_modello}}

in cui ho usato l'inte

rpolazione per collegare l

e proprietà nome_modello e

email_modello , assegnate a ngModel.
Questo mi permette di accedere in tempo

reale, al valore che l'utente sta inserendo
nel campo input, e visuali

zzarlo così nel

template.

Se provi a fare la stessa cosa in JavaScript

, ti renderai conto della potenza offerta

dalla direttiva ngModel e dalla notazione "banana-in-a-box"

Se avessi usato la direttiva nella notazione

[ngModel], quindi con un

comportamento monodirezionale, non avrei potuto valorizzare in tempo reale

la

proprietà collegata nel template.

E' possibile sfruttare l'oggetto ngForm anche per recuperare l'insieme di tutti i campi,

accedendo alla proprietà value, come visto nella riga che mostra l'alert:

invio(**form: NgForm**) {

alert("Tutto il form: " +
JSON.stringify(form.value));

...

}

Una seconda tecnica, consiste nell'usare la variabile locale

#mioform, che

identifica il form , per accedere direttamente all a proprietà value di ngForm, che

verrà passata come argomento al metodo invio():

<form #mioform="ngForm" (ngSubmit)="invio(**mioform.value**)">

In questo modo p

uoi accedere direttamente al valore associato all'oggetto

FormControl, con la sintassi:

oggettoNgForm.nomecampo

dove nomecampo, coincide proprio con nome inserito all'interno dell'attributo name

di ogni elemento del form.

L"unica cosa a cui devi fare attenzione è che il parametro passato al metodo

invio(), ora non è più di tipo NgForm, ma sarà generico ossia any, in quanto passo

direttamente il valore dei diversi campi.

Il codice del componente diventerebbe:

```
import { Component } from
'@angular/core';

import { NgForm } from
'@angular/forms';

@Component({

selector: 'ca-form-iscriviti',

template: `

<form #mioform="ngForm"
(ngSubmit)="invio(mioform.value)">
```

```
<label for="nome">Nome</label>

<input type="text" name="nome"

placeholder="Il tuo nome"

[(ngModel)]="nome_modello">

<label for="email">Email</label>

<input type="email" name="email"

placeholder="La tua email"

[(ngModel)]="email_modello">

<button type="submit">Invia</button>

</form>
```

```
<p>Stai scrivendo nel campo nome:
{{nome_modello}}</p>

<p>Stai scrivendo nel campo email:
{{email_modello}}</p>

`


})
```

(continua)

```
export class FormIscrivitiComponent {

constructor() {}
```

```
invio(form: any) {

console.log("Il nome inserito è: " +
form.nome);

console.log("L'email inserita è: " +
form.email);

}

}
```

form-iscriviti.component.ts

Un'altra proprietà d ell'oggetto ngForm che è possibile sfruttare direttamente nel

template del componente, è la proprietà valid, che ti permetterà di oscurare

sezioni

del form, in funzione della validità dei dati inseriti.

Ad esempio, potresti disabilitare il bottone d "invio, fino a che tutti i dati inseriti nei

diversi campi non siano validi, come indicato qui sotto:

<button type="submit" [disabled]="**!mioform.form.valid**">Invia</button>
dove ho reso obbligatori i campi di input con l"aggiunta di required:

import { Component } from '@angular/core';

```
import { NgForm } from
'@angular/forms';

@Component({

selector: 'ca-form-iscriviti',

template: `

<form #mioform="ngForm"
(ngSubmit)="invio(mioform.value)">

<label for="nome">Nome</label>

<input type="text" name="nome"

placeholder="Il tuo nome"

[(ngModel)]="nome_modello"
```

required>

<label for="email">Email</label>

<input type="email" name="email"

placeholder="La tua email"

(ngModel)]="email_modello"

maxlength="64" **required**>

<button type="submit"
[disabled]="!mioform.form.valid"
>Invia</button>

</form>

<p>Stai scrivendo nel campo nome:

{{nome_modello}}

Stai scrivendo nel campo email:
{{email_modello}}

`

})

(continua)

---

export class FormIscrivitiComponent {

invio(form:any) {

console.log("Il nome inserito è: " +

```
form.nome);

console.log("L'email inserita è: " +
form.email);

}

}
```

form-iscriviti.component.ts

Fino a che l"utente non ha inserito dei valori nei d ue campi, i l pulsante di invio non

sarà attivo.

13.5 Accedere ad un campo di <input> tramite

una variabile nel template

Sulla base delle considerazioni viste fino ad ora, siamo pronti per inserire il campo di

input che ci servirà per inviare il messaggio all'interno della chat scelta.

Hai chiaramente diverse possibili soluzioni, tra cui

- la più semplice - quella di

aggiungerlo direttamente all'interno del template del componente dettaglio.

In alternativa, potresti trasformarlo in un componente, nel caso avessi la necessità di

usarlo in più sezioni dell'app.

Analizzando così il codice qui sotto associato al template del componente

dettaglio.component.ts sviluppato in precedenza, puoi osservare una nuova

notazione, che in realtà non è poi così nuova perché l"abbiamo già in contrata più

volte:

<div *ngIf="treno">

```
<p> Treno Linea: {{treno.linea}}</p>

<p> ID: {{treno.idt}}</p>

</div>

<ca-chat *ngFor="let chat of listachat"
[msgIn]="chat"></ca-chat>

<input type="text" #msg required>

<button (click)="invio(msg.value);
msg.value="">Invia</button>
```

Invece di usare ngModel, sfruttiamo una caratteristica dei template in Angular, che

ci permette di usare una variabile locale

contrassegnata dal simbolo #.

In questo modo, limitandoci ad usarla solo nel te mplate, p uoi accedere al valore

inserito dall'utente nel campo di input

associato, e inviarlo come parametro di

ingresso ad un metodo definito nella classe.

<button (click)="invio(msg.value); msg.value="">Invia</button>

L'evento che monitoriamo è il c

lick e nota come abbia inserito un'espressione

costituita dal richiamo del metodo invio() e dall a valorizzazione de lla proprietà

value del campo input identificato da msg.

Queste due istruzioni, vengono eseguite in successione, avendo così la possibilità d i

cancellare la stringa inserita dall'utente, subito dopo che il dato è stato inviato.

Una volta ricevuto il dato, cosa dobbiamo fare? Dovremo visualizzarlo,

quindi

aggiungerlo all'array attualmente presente nella pagina e contemporaneamente

inviarlo a server remoto al fine di memorizzarlo nel database.

Per quest'ultima operazione dovremo avvalerci del Service creato in precedenza,

a

cui aggiungeremo un metodo per l'invio dei dati in POST.

Una bozza di codice potrebbe essere:

...

@Component({

selector: 'ca-dettaglio',

template: `<div *ngIf="treno">

Chat Treno: {{treno.linea}}

 ID: {{treno.idt}}

<ca-chat *ngFor="let chat of listachat"
[msgIn]="chat"></ca-chat>

<input type="text" #msg required>

<button
(click)="invioMsg(msg.value);msg.value

```
`

})

export class DettaglioComponent
implements OnInit {

idtreno: string;

treno: Metro;

listachat: Messaggio[];

chatmsg: Messaggio;

errormsg;

constructor(private route:
ActivatedRoute,
```

```
private treniservice:TreniService,

private chatservice:ChatService) {

}

ngOnInit() {

this.idtreno =
this.route.snapshot.paramMap.get('id');

this.getDettaglioMetroObservable(this.id

this.getListChatObservable(this.idtreno);

});

}
```

(continua)

194

// 1) Costruisco l'oggetto Messaggio con i dati noti

invioMsg(testomsg: string) {

// utente di test generico pari a 99 e idm impostato a 0

this.chatmsg = {'idm': 0,

'idt': this.idtreno,

'idu': '99',

```
'testo': testomsg};

this.sendMsgObservable(this.chatmsg)

}

// 2) Attendo che l'API risponda con
oggetto Messaggio

sendMsgObservable(msg: Messaggio)
{

this.chatservice.sendMsgChatObserva

.subscribe(

risp => this.listachat.push(risp[0]),

error=>this.errormsg = error);
```

```
}

getListChatObservable(idt: string) {

this.chatservice.getListaChatObservable(

.subscribe(

risp => this.listachat = risp,

error=>this.errormsg = error);

}

getDettaglioMetroObservable(idt:
string) {

this.treniservice.getDettaglioMetroObser
```

```
.subscribe(

risp => this.treno = risp[0],

error=>this.errormsg = error);

}

}
```

dettaglio.component.ts

Il codice si commenta da solo, nel senso che abbiamo confezionato l'oggetto

chatmsg, rispettivamente con i dati del treno, dell'ipotetico utente, il messaggio, e

un idm impostato a 0 perch é ancora non sappiamo il reale id entificativo che verrà

assegnato dopo il salvataggio in remoto.

Questo oggetto è stato poi passato al met odo interno sendMsgObservable(), che

attenderà la risposta del relativo metodo impostato nel

chat.service.ts. La risposta

dell'API, sarà un array costituito da un singolo oggetto, ed è questo il motivo per cui

dovrai accedere al primo elemento

dell'array, aggiungendolo all'attuale lista.

this.listachat.push(risp[0])

195

Chiaramente questa è una possibile soluzione, ma

avendo il controllo dell'API

remota, posso decidere di adottare questa strategia. Le cose cambierebbero se non

avessi questo controllo, dovendomi adattare a quello impostato da altri.

Nel service *chat.service.ts* invece, dovrai creare il metodo che effettuerà l'effettivo

invio al server remoto. Il metodo lo avevamo già creato in precedenza,

e gli

passeremo un oggetto Messaggio con le diverse proprietà già valorizzate, al fine di

ricostruire i parametri da passare e creare una stringa JSON:

```
sendMsgChatObservable(obj): Observable<Messaggio> {
```

```
return this.http.post<Messaggio>(
this.apiPostUrl,
{idtreno: obj.idt,
idutente: obj.idu,
messaggio: obj.testo})
.pipe(
map(risp => risp[‘dati’]),
catchError(this.handleErrors));
}
```

L'API remota, sarà progettata al fine di

ricevere i paramet

ri idtreno, idutente,

messaggio e di restituire un oggetto messaggio comprensivo anche del parametro

idm valorizzato al valore effettivo e non più zero.

Questo è una delle possibili soluzioni, che presenta lo svantaggio che è necessario

attendere la risposta dell'API prima di vedere il messaggio apparire nella lista.

Avresti potuto aggiungere subito il

messaggio all'array e poi effettuare l'invio, ma

dovresti poi cambiare la logica, nel caso di errore, in modo da cancellare il

messaggio o segnalare all'utente che non è stato memorizzato.

Esercitazione facoltativa

Come esercitazione potresti provare a progettare la sezione di login dell"applicazione

MetroChat, aggiungendo il componente di nome *login.component.ts* con due campi

di input: username e password.

Inoltre potresti disabilitare il bottone non appena l'utente clicca sul pulsante d i login,

e abilitarlo alla ricezione di una risposta dell'API remota.

196

13.6 Settare un dato all'interno di un campo di

<input>

Ora che abbiamo visto le diverse tecniche per r ecuperare un dato, vediamo come si

possa *scrivere* un valore all'interno di un campo di input. Questa operazione è

richiesta ad esempio tutte le volte in cui vuoi recuperare dei dati

precedentemente

memorizzati in una sorgente esterna al fine di modificarli.

Chiaramente l'operazione dovrà essere fatta nel corpo della classe . Dovrai pertanto

definire una proprietà, con lo stesso nome assegnato alla direttiva

ngModel, e

valorizzarla con un valore.

La sintassi da usare sarà:

nomecampo_modello = "valore da

impostare"

dove nomecampo_modello è proprio una proprietà della classe , collegata grazie a

ngModel, al relativo campo presente nel template.

Prendendo l'esempio iniziale, se volessi assegnare al campo nome il mio nome

"Davide" dovresti scrivere:

```
import { Component,OnInit } from '@angular/core';

import { NgForm } from '@angular/forms';
```

```
@Component({

selector: 'ca-form-iscriviti',

template: `

<form #mioform="ngForm"
(ngSubmit)="invio(mioform.value)">

<label for="nome">Nome</label>

<input type="text" name="nome"

placeholder="Il tuo nome"
[(ngModel)]="nome_modello">

<label for="email">Email</label>

<input type="email" name="email"
```

```
placeholder="La tua email"
[(ngModel)]="email_modello">

<button type="submit">Invia</button>

</form>

<p>Stai scrivendo nel campo nome:
{{nome_modello}}</p>

<p>Stai scrivendo nel campo email:
{{email_modello}}</p>

`


}) (continua)
```

export class FormIscrivitiComponent implements OnInit {

nome_modello: string=’’;

constructor() {}

ngOnInit() {

this.nome_modello = "Davide";

}

}

form-iscriviti.component.ts

dove ho

definito una proprietà

nome_modello, valorizzata all"interno di

ngOnInit()

13.7 Gestire i dati di un campo di select

I campi di s elect, sono costituiti da un insieme di valori, quindi nell'ipotesi tu voglia

ricreare in Angular in campo come quello qui sotto, che mostra alcune province

italiane con la relativa sigla:

<select name="pv">

<option value="MI">Milano</option>

<option value="BO">Bologna</option>

<option value="NA">Napoli</option>

<option value="CT">Catania</option>

</select>

sono sicuro ti venga in mente la direttiva *ngFor.

Infatti se avessi un array valorizzato in questo modo:

prov = [

```
{valore: 'PD', nome: 'Padova'},

{valore: 'BO', nome: 'Bologna'},

{valore: 'MI', nome: 'Milano'}

];
```

nel template potresti sfruttare *ngFor per mostrare l'elenco:

```
<select name="pv">

<option *ngFor="let pv of prov" value="
{{pv.valore}}">

{{ pv.nome }}
```

(continua)

```
</option>
```

```
</select>
```

Analogamente a quello visto per il campo input, vediamo come si possa sfruttare

ngModel per recuperare il valore scelto dall'utente. Costruiamo allora il modulo,

aggiungendo ngModel in corrispondenza al campo di select.

```
<form #mioform="ngForm"
(ngSubmit)="invio(mioform)">

<label for="pv">Provincia</label>
```

```
<select name="pv"
[(ngModel)]="pv_modello">

<option *ngFor="let pv of prov" value="
{{pv.valore}}">

{{ pv.nome }}

</option>

</select>

<button type="submit">Invia</button>

</form>

<p>Provincia selezionata
{{pv_modello}}
```

Come vedi ho aggiunto la classica notazione "banana -in-a-box", inserendo il nome

pv_modello per distinguerlo dall'attributo name. Avendolo inserito all'interno

delle parentesi quadre e tonde, significa che posso sia leggerne il valore che

mostrarlo nel template in tempo reale, non appena cambia.

All'interno del corpo della classe invece, dovrai sfruttare la stessa identica notaz ione

vista in precedenza per il campo di

input, quindi:

form.controls['pv'].value

o in alternativa, se passassi direttamente mioform.value al metodo invio:

form.pv

dove ti ricordo pv è il nome impostato per l'attributo name del campo. Il codice

completo diventa:

...

@Component({

selector: 'ca-form-iscriviti',

template: `

(continua)

<form #mioform="ngForm"
(ngSubmit)="invio(mioform.value)">

<label for="pv">Provincia</label>

<select name="pv"
[(ngModel)]="pv_modello">

<option *ngFor="let pv of prov" value="
{{pv.valore}}">

{{ pv.nome }}

```
</option>

</select>

<button type="submit">Invia</button>

</form>

<p>Provincia selezionata
{{pv_modello}}

`


})

export class FormIscrivitiComponent {

constructor() {}
```

```
invio(form: any) {

alert("Provincia selezionata: " +
form.pv);

}

}
```

form-iscriviti.component.ts

13.8 Settare un valore all'interno di un campo

select

Ipotizzando che nel database remoto, sia memorizzato il valore “BO” per il campo

provincia, come faccio a settarlo all"interno del campo di select, in modo che venga

visualizzato all"apertura del componente?

Dobbiamo sfruttare il collegamento tra il campo select de l template e la propri età

pv_modello,ottenuta grazie alla direttiva

ngModel

Impostando pertanto il valore alla stringa “BO”, ottengo:

this.pv_modello="BO"

In questo modo, tra la lista dei valori presenti nel campo di select, verrà visualizzato

quello corrispondente a Bologna.

...

@Component({

selector: 'ca-form-iscriviti',

```
template: `

<form #mioform="ngForm"
(ngSubmit)="invio(mioform.value)">

<label for="pv">Provincia</label>

<select name="pv"
[(ngModel)]="pv_modello">
```
(continua)

```
<option *ngFor="let pv of prov" value="
{{pv.valore}}">

{{ pv.nome }}

</option>
```

```
</select>

<button type="submit">Invia</button>

</form>

<p>Provincia selezionata
{{pv_modello}}

`


})

export class FormIscrivitiComponent {

pv_modello: string;

constructor() {
```

```
this.pv_modello = 'BO';

}

invio(form: any) {

alert("Provincia selezionata: " +
form.pv);

}

}
```

form-iscriviti.component.ts

13.9 Gestire i dati di un campo checkbox

Il campo checkbox e il campo radio,

come ben saprai, possono assumere due stati:

selezionato o non selezionato. Se ad esempio volessi dare la possibilità all'utente di

selezionare delle categorie di interesse, dovrei inserire:

<input type="checkbox" name="catA">

<input type="checkbox" name="catB">

Come sempre in Angular è necessario trasformarlo in un FormControl, grazie alla

direttiva ngModel, quindi la prima

modifica da fare sarà aggiungerla per ognuno.

Ipotizzando di inserire sempre il collegament o bidirezionale per vedere gli effetti in

tempo reale nel template:

<input type="checkbox" name="catA" [(ngModel)]="catA_modello>

<input type="checkbox" name="catB" [(ngModel)]="catB_modello>

dove, come già detto più volte, ho differenziato i nomi rispetto all'attributo name, per

capire i diversi ruoli. Se aggiungessi nel template questi due campi:

```
<form #mioform="ngForm"
(ngSubmit)="invio(mioform.value)">

<label for="cat">Categorie</label>
```

(continua)

```
<input type="checkbox" name="catA"
[(ngModel)]="catA_modello>

<input type="checkbox" name="catB"
[(ngModel)]="catB_modello>

<button type="submit">Invia</button>
```

</form>

Categoria A: {{catA_modello}}

Categoria B: {{catB_modello}}

noteresti che al cambiamento dello stato di uno qualsiasi dei campi, verrebbe

visualizzata la stringa "true" o "false", dove con

“true” si intende un campo

selezionato.

Indipendentemente dallo stato, il

recupero del valore "true" o "false" all"interno della classe, avviene con la seguente notazione:

this.campo_modello

dove con campo_modello, intendo il nome scelto per valorizzare ngModel.

Il codice completo diventa:

...

@Component({

selector: 'ca-form-iscriviti',

template: `

```
<form #mioform="ngForm"
(ngSubmit)="invio(mioform.value)">

<label for="cat">Categorie</label>

<input type="checkbox" name="catA"
[(ngModel)]="catA_modello">

<input type="checkbox" name="catB"
[(ngModel)]="catB_modello">

<button type="submit">Invia</button>

</form>

<p>Categoria A: {{catA_modello}}
</p>

<p>Categoria B: {{catB_modello}}
```

```
</p>

`

})

export class FormIscrivitiComponent {

catA_modello:boolean = false;

catB_modello:boolean = false;

constructor() {}

invio(form: any) {

alert("Valore checkbox: " +
this.catA_modello);
```

}

}

form-iscriviti.component.ts

202

Nel caso invece volessi settare un valore, trattandosi di valori booleani, dovre

sti

impostare una propri età con lo stesso nome collegato a ngModel e valorizzarla con

“true” o “false”.

Ad esempio , se volessi impostare la categoria B a

llo stato “ selezionata”, dovre sti

scrivere:

...

@Component({

selector: 'form-iscriviti',

template: `

<form #mioform="ngForm" (ngSubmit)="invio(mioform.value)">

```
<label for="cat">Categorie</label>

<input type="checkbox" name="catA"
[(ngModel)]="catA_modello">

<input type="checkbox" name="catB"
[(ngModel)]="catB_modello">

<button type="submit">Invia</button>

</form>

<p>Categoria A: {{catA_modello}}
</p>

<p>Categoria B: {{catB_modello}}
</p>

`
```

```
})

export class FormIscrivitiComponent {

catA_modello:boolean = false;

catB_modello:boolean = false;

constructor() {

this.catB_modello = true;

}

invio(form: any) {

alert("Valore checkbox" +
this.catA_modello);
```

}

}

form-iscriviti.component.ts

13.10 Gestire i dati di un campo radiobox

Come dicevamo in precedenza, questo campo può assumere due stati. Ad esempio se

avessi bisogno dell'informazione sul sesso di una persona (Ma

schio o Femmina),

potresti memorizzati i dati in un array,

così come fatto per il campo checkbox:

```
sesso = [
{valore: 'M', nome: 'Maschile'},
{valore: 'F', nome: 'Femminile'}
];
```

---

e sfruttare la direttiva *ngFor, per creare i d ue campi di input, con collegata la

direttiva ngModel, con le stesse tecniche viste per i campi precedenti:

```
<div *ngFor="let genere of sesso">

<input type="radio" name="sesso"

value="{{genere.valore}}"

[(ngModel)]="sesso_modello">
{{genere.nome}}

</div>
```

Il recupero del valore, avviene con le stesse tecniche viste per il campo checkbox,

quindi scriveremo:

...

```
@Component({

selector: 'ca-form-iscriviti',

template: `

<form #mioform="ngForm"
(ngSubmit)="invio(mioform.value)">

<label for="cat">Maschio o
Femmina</label>

<div *ngFor="let genere of sesso">

<input type="radio" name="sesso"

value="{{genere.valore}}"

[(ngModel)]="sesso_modello">
```

```
{{genere.nome}}

</div>

<button type="submit">Invia</button>

</form>

<p>Sesso: {{sesso_modello}}</p>

`

})

export class FormIscrivitiComponent {

constructor() {}

invio(form: any) {
```

```
alert("Valore radiobox: " +
this.sesso_modello);

}

}
```

form-iscriviti.component.ts

Per poter impo stare un valore nel campo invece, dovr

ai valorizzare una proprietà

avente lo stesso nome di quello associato al campo tramite ngModel.

Ad esempio se volessi mostrare un valore predefinito pari a “M”, dovresti

scrivere:

this.sesso_modello = 'M';

Il codice completa diventa:

...

@Component({

selector: 'ca-form-iscriviti',

template: `

<form #mioform="ngForm"
(ngSubmit)="invio(mioform.value)">

```
<label for="cat">Maschio o
Femmina</label>

<div *ngFor="let genere of sesso">

<input type="radio" name="sesso"

value="{{genere.valore}}"

[(ngModel)]="sesso_modello">
{{genere.nome}}

</div>

<button type="submit">Invia</button>

</form>

<p>Sesso: {{sesso_modello}}</p>
```

```
`

})

export class FormIscrivitiComponent {

sesso_modello:string;

constructor() {

this.sesso_modello = 'F';

}

invio(form: any) {

alert("Valore radiobox: " +
this.sesso_modello);
```

}

}

form-iscriviti.component.ts

205

Conclusione e Codice Personale

Siamo arrivati alla conclusione di queste 24 ore di corso e devo congratularmi con te

per essere arrivat o fino a qui. Non è stato un percorso agevole, ma spero di essere

riuscito nel mio intento, ossia quello di farti capire

le basi del funzionamento del

framework Angular, nel più breve tempo

possibile.

Ci sarebbero altri mille possibili approfondimenti, quindi ti suggerisco di seguirmi

consultando il sito https://www.creareapp.com oppure https://www.video-corsi.com

Ti invito a scriver e il tuo feedback e a p ubblicarlo nel caso sia positivo (4 e 5 stelle

corrispondono ad un feedback positivo, le altre negativo

), per incentivare altre

persone a seguire il libro . Eventuali tuoi commenti personali , anche negativi se

motivati, e/o richieste specifiche, puoi inviar le direttamente per email all"indirizzo:

assistenza9@video-corsi.com

Rileggi con calma le sezioni più “ostiche” e testa il tuo apprendimento sul campo ,

provando a riscrivere i diversi pezzi dell"applicazione sviluppata nel libro, provando

anche a svolgere le diverse esercitazioni

richieste.

Tutto il codice scritto nel libro lo puoi **scaricare gratuitamente**, registrandoti al link

qui sotto . **NB:** Tieni presente che saranno presenti più versioni del codice, e ch

e

probabilmente alcune righe saranno diverse da quelle presenti nel libro, in seguito ai

frequenti aggiornamenti di Angular:

https://su.video-corsi.com/registrati.php

Dovrai registrarti all"area privata, fornendo il tuo nome, l"indirizzo email che usi su

Amazon e una password di almento 5 caratteri . Per attivare il download del codice

inserisci questo dato personale: **8GWFEKTGH23W94** (tutto in maiuscolo) o quello

presente nell"ultima pagina del libro a fianco di “Printed in”.

Ti invito a segnalarmi privatamente eventuali errori di battitura o sul codice in modo

che possa correggerli anche sul libro. Ogni nuovo aggiornamento introduce

potenzialmente nuovi errori, quindi grazie fin da ora se vorrai scrivermi.

Buona programmazione!

Un abbraccio

Davide Copelli {ing.}

APPENDICE

I test per il recupero e l"invio delle informazioni dell"applicazione MetroChat, posso

essere condotti sfruttando un"API realizzata per l"occasione e raggiunbile con i

comandi indicati qui sotto. Per ogni chiamata, sono registrati l"indirizzo IP, la data e

l"ora, in modo da bloccare eventuali richieste multiple che potrebbero rallentare il

server. Ti invito ad usarla con moderazione.

L"indirizzo principale dell"API è: http://angular.dcopelli.it/

Qui sotto, sono indicati la tipologia di chiamata (GET, POST) e il relativo percorso.

GET /treni/ - Restituisce la lista dei treni in arrivo presso un"ipotetica stazione.

Nella demo verranno restituiti tre treni, con un ritardo alla partenza variabile. Una

volta che l"ultimo treno è partito, dovrai

rieseguire l"app per avere nuovamente la

lista dei treni aggiornata.

(URL di chiamata: http://angular.dcopelli.it/treni/)

Formato della risposta: oggetto JSON con singola chiave indicata qui sotto:

nome

tipo

Significato del valore

dati

Array

Il valore è un array costituito da oggetti JSON

con la lista di chiavi evidenziata qui sotto.

Oggetto JSON (metro):

nome

tipo

Significato del valore

idt

string

Identificativo del treno

linea

string

Nome della linea

direzione

string

Ultima stazione in quella direzione

numchatting string

Numero persone in chat in quel treno

tempo

number

Orario di partenza espresso in secondi dalla

data di Unix.

POST /treni/ - Restituisce i dati di dettaglio del treno

Le intestazioni devono contenere

l"identificativo del treno ***idt*** e l"API restituisce i

dati del treno corrispondente.

(URL di chiamata: http://angular.dcopelli.it/treni/)

207

Formato della risposta: oggetto JSON con singola chiave indicata qui sotto:

nome

tipo

Significato del valore

dati

Array

Il valore di dati è un array costituito da un

singolo oggetto JSON, rappresentativo del treno,

con la seguente sequenza di chiavi.

Oggetto JSON (metro):

nome

tipo

Significato del valore

idt

string

Identificativo del treno

linea

string

Nome della linea

direzione

string

Ultima stazione in quella direzione

numchatting string

Numero persone in chat in quel treno

tempo

number

Orario di partenza espresso in secondi dalla

data di Unix.

POST /chat/ - Restituisce la lista dei messaggi chat di uno specifico treno

Le intestazioni devono contenere un

oggetto JSON con chiave ***idtreno*** e l"API

restituisce la lista dei messaggi chat scambiati in quel treno.

(URL di chiamata: http://angular.dcopelli.it/chat/)

Formato della risposta: oggetto JSON con singola chiave indicata qui sotto:

nome

tipo

Significato del valore

dati

Array

Ogni elemento dell"array è un oggetto JSON con

la lista di chiavi evidenziata qui sotto.

Oggetto JSON (messaggio):

nome

tipo

Significato del valore

idm

string

Identificativo del messaggio

testo

string

Testo del messaggio

idu

string

Identificativo dell"utente

idt

string

Identificativo del treno

iddestinatario number

Identificativo del destinatario del messaggio

privato.

POST /chat/send/ - Permette di memorizzare un messaggio chat.

Le intestazioni devono contenere un

oggetto JSON con chiavi ***idm, idt, idu, testo*** e

208

l"API restituisce il messaggio completo comprensivo dell"identificativo aggiornato

del messaggio.

(URL di chiamata: http://angular.dcopelli.it/chat/send/)

Formato della risposta: oggetto JSON con singola chiave indicata qui sotto:

nome

tipo

Significato del valore

dati

Array

Ogni elemento dell"array è un oggetto JSON con

la lista di chiavi evidenziata qui sotto.

Oggetto JSON (messaggio):

nome

tipo

Significato del valore

idm

string

Identificativo del messaggio

testo

string

Testo del messaggio

idu

string

Identificativo dell"utente

idt

string

Identificativo del treno

iddestinatario number

Identificativo del destinatario del messaggio

privato.

PUT /chat/send/preferiti/ - Abilita/Disabilita l"indicazione associata al simbolo di

“preferiti” per un messaggio chat.

Le intestazioni devono contenere un oggetto JSON con chiavi ***idutente,***

idmessaggio, stato e l"API restituisce una stringa che indica la riuscita o meno

dell"operazione.

(URL di chiamata: http://angular.dcopelli.it/chat/send/prefer

Formato della risposta: stringa con valori “1” o “0”