Corso Udemy

Inizializzazione

Inserire questi comandi da terminale:

```

```

Commento dopo la creazione del progetto Angular

- *package.json*: file in cui vengono inserite tutte le dipendenze che il progetto Angular dovra' utilizzare. Quando dovremo aggiungere una dipendenza, andra' inserita qui
- cartella *node_modules*: quando avremo aggiunto una dipendenza, la vedremo qui. Contiene tutte le dipendenze che abbiamo specificato in package.json
- cartella *src*: contiene il codice sorgente. Al suo interno, i files piu' importanti sono:
  - *main.ts*: file typescript che costituisce l'entry point della nostra applicazione
  - *index.html*: html entry point. Al suo interno e' presente il *tag app-root*
  - *polyfills.ts*: trasforma il codice ts in codice javascript compatibile con i diversi browsers
  - *styles.css*: foglio di stile
  - *test.ts*: file con con cui poter eseguire gli *unit tests* per la nostra applicazione
  - cartella *app*: contiene tutti i componenti che compongono l'applicazione
  - cartella *assets*: contiene i file come immagini che verranno utilizzate nella pagina web

Componenti Angular - Intro

Le diverse parti che vediamo in una pagina web (vedi header - menu' - footer), tra loro interconnessi. Angular ci permette di creare questi elementi separatamente. I componenti a loro volta sono costituiti da:

- *Template* (componente html)

- *Stile* (componente css)
- *Codice* (componente typescript)

I componenti sono utili per la *riusabilita'* del codice e per semplificare il suo *mantenimento*.

Alla creazione di un'app con Angular, viene creato automaticamente un primo componente, che risiede sotto la cartella *app*. Questo componente e' l'*app.component*, che e' quello che andiamo ad utilizzare in *index.html* (l'*app-root*). Il tag app-root lo deriviamo dall'*app.component.ts*, in cui vediamo un codice come questo:

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

@Component( {
    selector: 'app-root',
    template: './app.component.html',
    styleUrls: ['./app.component.css']
})

export class AppComponent {
    title = 'nomeApp';
}
```

La variabile *title* e' quella che poi viene citata in nell'*app.component.html*. E' possibile aggiungere altre variabili, come nel codice d'esempio qui sotto:

```
// [...]

export class AppComponent {
    title = 'nomeApp';
    greetings: string = 'Benvenuti in nomeApp';
}

// [...]
```

Creazione di un nuovo componente

```
ng generate component <name> [options]
```

Nel nostro esempio:

```
ng generate component welcome --skip-tests --dry-run
```

Per non generare il file con gli unit tests. Con la seconda opzione avremo invece un'anteprima di cio' che si andra' a generare. E' come se si iniziasse una transazione senza dare il commit e tornando indietro con un rollback. Se non notiamo errori nell'esecuzione del comando sopra, ripetiamolo senza l'ultima opzione.

Dentro la cartella *app* viene cosi' creata una nuova sotto-cartella *welcome* che contiene le sotto-componenti (typescript, html, css) del componente appena generato.

Alla creazione del componente, viene anche aggiornato il file *app.module.ts*. E' il file in cui vengono specificati i vari elementi che compongono l'applicazione:

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

import {AppRoutingModule} from './app-routing.module';
import {AppComponent} from './app.component';
import {WelcomeCompotent} from './welcome/welcome.component';

@NgModule ({
    declarations: [
        AppComponent,
        WelcomeComponent
    ],
    imports: [
        BrowserModule,
        AppRoutingModule
    ],
    providers: [],
    bootstrap: [AppComponent]
})
export class AppModule {}
```

Indagando le diverse sotto-componenti appena create, vedremo che:

- *welcome.component.html* viene generato con un semplice tag che dice che il componente funziona
- *welcome.component.ts* e' piu' articolato:

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

@Component ({
    selector: 'app-welcome',
    templateUrl: './welcome.component.html',
```

```
    styleUrls: ['./welcome.component.css']
})
export class WelcomeComponent implements OnInit {

    constructor() {}

    ngOnInit(): void {
    }

}
```

ngOnInit() e' il primo metodo che viene lanciato in automatico alla creazione del nostro componente. In esso si possono quindi inserire degli elementi di inizializzazione.

Altro elemento importante e' il *selettore*. Per poter inizializzare il componente, bisogna dirigersi in *app.component.html* e aggiungere il componente di nostro interesse come se fosse un tag:

```
<div style="text-align:center">
    <h1>
        {{greetings}}
    </h1>
    <app-welcome></app-welcome>
</div>
```

Creazione componente Login

```
ng generate component login --skip-tests
```

Sotto la cartella *app* viene generata la sotto-cartella *login*. Andiamo a modificare il file *login.component.html* in questo modo:

```
<label for="userId">User Id:</label>
<input type="text" id="userId" placeholder="Nome Utente">

<label for="password">Password:</label>
<input type="text" id="password" placeholder="Password">

<button>Connetti</button>
```

Implementate queste modifiche, andiamo a fare in modo di utilizzare il componente login. Per prima cosa si va a vedere quale *selettore* e' stato utilizzato nella definizione del *login.component.ts* (app-login) ed usarlo come tag nell'*app.component.html*, che diventera':

```
<app-login></app-login>
```

Visto che stiamo costruendo l'app da zero, il codice di prima non ci serviva piu', e abbiamo lasciato solo il rimando al componente di login.

Trovando un'analogia con il progetto che vogliamo sviluppare, concettualmente vorrei che chiunque potesse vedere i threads ed i commenti, ma solo gli utenti logati potessero postare e reactare.

Andiamo sul *login.component.ts*:

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

@Component ({
    selector: 'app-login',
    templateUrl: './login.component.html',
    styleUrls: ['./login.component.css']
})
export class LoginComponent implements OnInit {

    userId: String = "Andrea";
    password: String = "";

    constructor() {}

    ngOnInit(): void {
    }

}
```

In questo modo abbiamo inizializzato le due variabili necessarie al login. Affinche' si possa visualizzare questa inizializzazione, dobbiamo tornare sul *login.component.html*:

```
<label for="userId">User Id:</label>
<input type="text" id="userId" placeholder="Nome Utente" value = "{{userId}}">

<label for="password">Password:</label>
<input type="text" id="password" placeholder="Password">
```

```
<button (click) = gestAuth()>Connetti</button>
```

Tramite *interpolazione* ci stiamo riferendo al valore assunto dalla variabile *userId* definita in *login.component.ts*. Abbiamo poi dato vita al bottone *Connetti*, associandolo all'evento *click*. Al click sul bottone scatta il metodo *gestAuth()* (che ci accingiamo a creare). Sara' il metodo con cui gestiamo l'autorizzazione del login.

Torniamo sul *login.component.ts* per aggiungere il metodo di cui sopra:

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

@Component ({
    selector: 'app-login',
    templateUrl: './login.component.html',
    styleUrls: ['./login.component.css']
})
export class LoginComponent implements OnInit {

    userId: string = "Andrea";
    password: string = "";

    constructor() {}

    ngOnInit(): void {
    }

    // metodo senza parametri
    // restituisce semplicemente la stampa dello userId
    // sfruttando una arrow function
    // Vedremo "Andrea" guardando dagli Strumenti per Sviluppatori, sotto la voce
"Console"
    gestAuth = () : void => {
        console.log(this.userId);
    }

}
```

N.B. la console.log restituira' sempre "Andrea", anche se cambiamo il nome dell'utente a schermo.
Al momento e' un processo unidirezionale: dall'html, il cambiamento non passa al typescript.

Cerchiamo di implementare ora un *Data Binding bidirezionale* (leggere e scrivere il valore della variabile).

Modifichiamo il *login.component.html*:

```
<label for="userId">User Id:</label>
<!--
```

```
    il costrutto sintattico [()] si chima "banana in the box"
    al suo interno inseriamo una direttiva di angular, ovvero ngModel
    che valorizziamo con la variabile userId
-->
<input type="text" id="userId" placeholder="Nome Utente" [(ngModel)] = userId>

<label for="password">Password:</label>
<input type="text" id="password" placeholder="Password" [(ngModel)] = pasword>

<button (click) = gestAuth()>Connetti</button>
```

Una modifica di questo tipo non e' sufficiente. Dobbiamo andare a modificare anche l'*app.module.ts*:

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

import {AppRoutingModule} from './app-routing.module';
import {AppComponent} from './app.component';
import {WelcomeCompotent} from './welcome/welcome.component';
// aggiunto
import {FormsModule} from '@angular/forms';

@NgModule ({
    declarations: [
        AppComponent,
        WelcomeComponent,
        LoginComponent
    ],
    imports: [
        BrowserModule,
        AppRoutingModule,
        // aggiunto
        FormsModule
    ],
    providers: [],
    bootstrap: [AppComponent]
})
export class AppModule {}
```

andando ad importare il modulo *FormsModule* che permette per l'appunto il binding bidirezionale.
Ricapitolando: l'interpolazione e' solo di lettura, mentre l'ngModel e' di lettura e scrittura.

Intro all'ngIf

Andiamo ad ampliare il *login.component.ts*:

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

@Component ({
    selector: 'app-login',
    templateUrl: './login.component.html',
    styleUrls: ['./login.component.css']
})
export class LoginComponent implements OnInit {

    userId: string = "Andrea";
    password: string = "";

    // aggiungiamo una variabile sull'autenticazione ed un messaggio di errore
    autenticato: boolean = true;
    consentito: boolean = true;
    errMsg: string = "Spiacente, user id e/o password sono errati. Riprovare";
    okMsg: string = "Accesso effettuato con successo";

    constructor() {}

    ngOnInit(): void {
    }

    gestAuth = () : void => {
        console.log(this.userId);

        // implementiamo qui un primo esempio di controllo sull'autenticazione
        // bisogna mettere il "this" riferendoci alle variabili definite nella
classe
        if (this.userId === "Andrea" && this.password === "pupopeligroso95") {
            this.autenticato = true;
            this.consentito = true;
        }
        else {
            this.autenticato = false;
            this.consentito = false;
        }
    }

}
```

Ora torniamo su *login.component.html*:

```
<!--
    inseriamo il tutto all'interno di un blocco div
-->
<div>
<label for="userId">User Id:</label>
<input type="text" id="userId" placeholder="Nome Utente" [(ngModel)] = userId>
```

```
<label for="password">Password:</label>
<input type="text" id="password" placeholder="Password" [(ngModel)] = pasword>

<button (click) = gestAuth()>Connetti</button>
</div>
<!--
    aggiungiamo un ulteriore blocco div
    dove andiamo ad inserire l'interpolazione della variabile errMsg
    da visualizzare solo se "autenticato" e' pari a false
    tramite *ngIf
-->
<div *ngIf="!autenticato">
    {{errMsg}}
</div>

<div *ngIf="consentito">
    {{okMsg}}
</div>
```

ngIf="autenticato" significa che la parte di codice dentro la div verrebbe mostrata solo in caso un cui autenticato sia pari a true. Noi vogliamo il contrario, ed e' per questo che abbiamo aggiunto il punto esclamativo, con cui otteniamo l'inverso.

N.B. non si puo' sfruttare esclusivamente *autenticato*?

Routing

Il routing e' la capacita' dell'applicazione di poter modificare il contenuto della pagina sulla base della richiesta delle risorse che vengono effettuate dal client. Esempi:

- http://localhost:4200/login
- http://localhost:4200/index

I routing in Angular si implementano in uno specifico file: *app-routing.module.ts*.

```
// [...]

const routes: Routes = [
    // di default e' blank; il codice qui dentro viene aggiunto da noi
    {path:'login', component: LoginComponent}
];

// [...]
```

A questo punto ci dirigiamo nell'*app.component.html*, che al momento risulta cosi':

```
<app-login></app-login>
```

e la modifichiamo in questo modo:

```
<router-outlet></router-outlet>
```

questo tag attiva il routing in questa pagina.

Che cosa succede se andiamo ad accedere ad una risorsa che non e' gestita dal routing? Vedremmo una pagina bianca. Quindi andiamo a modificare di nuovo *app-routing.module.ts*:

```
// [...]

const routes: Routes = [
    {path:'', component: LoginComponent}
    {path:'login', component: LoginComponent}
];

// [...]
```

In questo modo la nostra *landing page* verra' rimandata al LoginComponent.

Per gestire le eccezioni, la cosa migliore da fare e' creare un nuovo componente, *error*:

```
ng generate component error
```

Ora possiamo andare a specificare cosa succede quando si accede ad una pagina che non esiste. Torniamo su *app-routing.module.ts*:

```
// [...]

const routes: Routes = [
    {path:'', component: LoginComponent}
    {path:'login', component: LoginComponent}
    // aggiungiamo gestione delle eccezioni:
```

```
    {path:'**', component: ErrorComponent}
    // dovra' sempre essere l'ultimo path dell'elenco
];

// [...]
```

Code-Injection

Andiamo sul *login-component.ts*:

```
import { Router } from '@angular/router';

// [...]

// all'interno dei parametri del costruttore aggiungiamo
costructor(private route: Router) {}

// [...]
```

Abbiamo fatto *code injection* avendo la possibilita' di accedere a tutte le proprieta' di questo oggetto di tipo Router.

A questo punto sfruttiamo questa possibilita' per far si' che all'autenticazione corretta l'utente sia direzionato alla pagina di benvenuto.

Rimanendo sempre nel *login-component.ts*, andiamo a modiicare un'ulteriore parte di codice:

```
// [...]

if (this.user === 'Andrea' && this.password === 'pupopeligroso') {
    // aggiungiamo cio' di cui abbiamo appena discusso:
    this.route.navigate(['welcome']);
    // [...]
}
```

Con la riga di codice appena aggiunta, stiamo accedendo al metodo *navigate* dell'oggetto *Route*, che ci porta ad uno dei *path* definiti in *app-routing.module.ts*.

Creazione di una tabella dati (introduzione NgFor)

Creiamo un nuovo componente:

```
ng negerate component articoli
```

Ci dirigiamo nell'*articoli.component.ts*:

```
// [...]

export class ArticoliComponent implements OnInit {

    // aggiungiamo una variabile di tipo array:
    articoli = [
        // inseriamo una serie di dati come
        // {...}
    ]

    constructor() { }

    ngOnInit() {}
}
```

A questo punto andiamo a modificare *articoli.component.html*:

```
<h1>Articoli disponibili</h1>

    <table id="articoli" border="1">
        <caption>Alimentari</caption>
        <thead>
            <tr>
                <th>CodArt</th>
                <th>DescArt</th>
                <th>Qnty</th>
                <th>Price</th>
            </tr>
        </thead>
        <tbody>
            <tr *ngFor="let articolo of articoli">
                <td>{{articolo.codArt}}</td>
                <td>{{articolo.descArt}}</td>
                <td>{{articolo.qnty}}</td>
                <td>{{articolo.price}}</td>
            </tr>
        </tbody>
    </table>
```

Chiaramente le proprieta' qui sopra saranno quelle che abbiamo definito all'interno dei dati dentro l'array *articoli*.

Fatto cio', dobbiamo andare ad aggiornare il *routing*, quindi andiamo in *app-routing.module.ts*:

```
// [...]

const routes: Routes = [
    {path:'', component: LoginComponent}
    {path:'login', component: LoginComponent}
    {path:'articles', component: ArticoliComponent}
    {path:'**', component: ErrorComponent}

];

// [...]
```

A questo punto potrebbe essere interessante fare in modo che dalla pagina di welcome, tramite *routerLink*, si possa accedere alla pagina degli articoli. Prendiamo il *welcome.component.html*:

```
<p>Saluti, clicca a <a routerLink="/articoli"><a> per vedere la lista degli
articoli disponibili<p>
```

Tipi e Pipes

L'approccio che abbiamo descritto sopra funziona, chiaramente, ma non e' formalmente corretto.

Il modo giusto di procedere e':

- creare una cartella (allo stesso livello delle componenti), chiamata *models*
- all'interno di questa cartella creare un file *Articoli.ts* in cui indicheremo tutte le proprieta' che deve avere la classe Articoli:

```
export interface IArticoli {
    // queste sono le proprieta' con cui ci eravamo gia' interfacciati
     codArt: string
     descArt: string
     qnty: number
     price: number
     // aggiungiamo altre due proprieta':
     active: boolean
     data: Date
```

```
}
```

Ora si tratta di andare a modificare *articoli.component.ts*:

```
// [...]

export class ArticoliComponent implements OnInit {

    // specifichiamo il tipo IArticoli, che e' un array. Quindi gli aggiungimo
    // le parentesi quadre
    articoli: IArticoli[] = [
        // inseriamo una serie di dati come
        // {...}
        // RICORDANDOCI DI AGGIUNGERE LE PROPRIETA' ACTIVE E DATA CHE
        // ABBIAMO INSERITO SOPRA
    ]

    constructor() { }

    ngOnInit() {}
}
```

Vista l'aggiunta delle due nuove proprieta', dobbiamo andare a modificare anche *articoli.component.html*:

```
<h1>Articoli disponibili</h1>

    <table id="articoli" border="1">
        <caption>Alimentari</caption>
        <thead>
            <tr>
                <th>CodArt</th>
                <th>DescArt</th>
                <th>Qnty</th>
                <th>Price</th>
                <th>Active</th>
                <th>Date</th>
            </tr>
        </thead>
        <tbody>
            <tr *ngFor="let articolo of articoli">
                <td>{{articolo.codArt}}</td>
                <td>{{articolo.descArt}}</td>
                <td>{{articolo.qnty}}</td>
                <td>{{articolo.price}}</td>
                <td>{{articolo.active}}</td>
```

```
                <td>{{articolo.date}}</td>
            </tr>
        </tbody>
    </table>
```

Unico appunto: la data salta fuori in un formato che fa cagare. Bisogna riformattarla e per far questo usiamo il *pipe (|)*, quindi:

```
<h1>Articoli disponibili</h1>

    <table id="articoli" border="1">
        <caption>Alimentari</caption>
        <thead>
            <tr>
                <th>CodArt</th>
                <th>DescArt</th>
                <th>Qnty</th>
                <th>Price</th>
                <th>Active</th>
                <th>Date</th>
            </tr>
        </thead>
        <tbody>
            <tr *ngFor="let articolo of articoli">
                <td>{{articolo.codArt}}</td>
                <td>{{articolo.descArt}}</td>
                <td>{{articolo.qnty}}</td>
                <td>{{articolo.price | currency : 'EUR'}}</td>
                <td>{{articolo.active}}</td>
                <td>{{articolo.date | date: 'dd-MM-YYYY'}}</td>
            </tr>
        </tbody>
    </table>
```

Abbiamo modificato anche la formattazione del prezzo.

Bootstrap

E' una libreria che rende tutto piu' figo esteticamente.

```
npm install bootstrap@5.3.0-alpha1
```

```
gem install bootstrap -v 5.3.0-alpha1
```

Ora bisogna abilitare il bootstrap sul nostro progetto. Il sistema piu' semplice per farlo e' modificare il file *styles.css* (che e' il foglio di stile generale dell'intera applicazione):

```
@import "~bootstrap/dist/css/bootstrap.css";
```

Ora possiamo usare il bootstrap. Facciamo una prova andando a modificare un bottone che avevamo creato in precedenza, dal file *login.component.html*:

```
<!--
    [...]
-->
<div>
<label for="userId">User Id:</label>
<input type="text" id="userId" placeholder="Nome Utente" [(ngModel)] = userId>

<label for="password">Password:</label>
<input type="text" id="password" placeholder="Password" [(ngModel)] = pasword>

<button class="btn btn-primary" (click) = gestAuth()>Connetti</button>
</div>
<div *ngIf="!autenticato">
    {{errMsg}}
</div>
```

Abbiamo aggiunto una classe specifica del bootstrap per rendere piu' gradevole l'estetica del bottone "Connetti".

Moduli

In Angular le applicazioni sono modulari, sono divise in contenitori che prendono il nome di *NgModules*, all'interno dei quali sono presenti i *componenti*, i *servizi* ed altri elementi di codice tra loro interrelati. I moduli possono importare altri moduli ed esportare funzionalita' predisposte in essi.

In ogni applicazione e' presente almeno un modulo, ovvero l'*AppModule (app.module.ts)*, da cui si avvia l'app.

Di grande importanza all'interno di questo file e' il *bootstrap*, che indica quale componente viene lanciato all'avvio del modulo.

Il flusso dell'app e':

- l'applicazione si avvia grazie al file *main.ts*

- dentro questo file avviene l'import dell'*AppModule*
- questo modulo viene utilizzato come argomento del *bootstrap* di *main.ts*
- di conseguenza si avvia l'*AppModule*
- l'*AppModule* lancia l'*AppComponent*, contenuto come argomento del *bootstrap*
- l'*AppComponent* e' composto dal *selettore app-root*
- questo selettore e' utilizzato nell'*index.html*
- l'*AppComponent* chiama anche l'*app.component.html*

Ricordiamoci che queste sono applicazioni *SPA(single page applications)*, con un'unica pagina html, che viene modificata dinamicamente grazie alle funzionalita' di Angular.

Generazione di un nuovo modulo

Creiamo il modulo *Core*:

```
ng generate module core
```

Se apriamo il file generato (*core.modules.ts*), vedremo:

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

@NgModule({
    declarations: [],
    imports: [
        CommonModule
    ]
})
export class CoreModule {}
```

E' come se fosse un template vuoto, quindi provvediamo a personalizzarlo. L'intenzione e' quella di creare un header delle pagine. Quindi cominciamo con la creazione di un nuovo componente, che non vogliamo inserire nel modulo primario, bensi' nel nostro nuovo modulo:

```
ng generate component header --path=src/app/core --module=core --export
```

Con l'ultima opzione stiamo specificando che questo componente verra' esportato, e quindi potra' essere visualizzato anche da componenti esterni.

Discorso analogo con il componente footer:

```
ng generate component footer --path=src/app/core --module=core --export
```

Alla luce della creazione di questi due nuovi componenti, e' stato aggiornato il file *core.modules.ts*.

Andiamo a modificare il *footer.component.html* cosi':

```
<nav class="navbar fixed-bottom navbar-light bg-light">
    <div class="container-fluid">
        <p>Matto sul terrazzo</p>
    </div>
</nav>
```

Aggiorniamo poi l'header. Il piano e' quello di inserire una *barra di navigazione*. Il socio del corso e' andato sul sito di bootstrap e ha cercato le navbar.

N.B. GUARDA BENE QUEL SITO PERCHE' E' PRESENTE UNA PARTE DEDICATA AI MENU' A DROPDOWN CHE POSSONO AVERE VOCI COME:

- ACCEDI
- REGISTRATI
- LOGOUT

Ad ogni modo, in *app.modules.ts* dobbiamo andare ad aggiungere il riferimento al modulo *core*:

```
// [...]

import { CoreModule}  from './core/core.module'

// [...]

@NgModule({
    // [...]
    imports: [
        // [...]
        CoreModule
    ]
})
```

Possiamo ora andare sull'*app.module.html*, aggiungendo i riferimenti all'header e al footer:

```
<app-header></app-header>

<router-outlet></router-outlet>

<app-footer></app-footer>
```

Funziona tutto tranne il menu' a dropdown di cui parlavo prima.

Dobbiamo fare una modifica al file *angular.json*. Ci serve inserire uno script in javascript nella sezione *scripts* di questo file:

```
// [...]
"scripts": [
   "node_modules/bootstrap/dist/js/bootstrap.min.js"
],

// [...]
```

Giusto per ragioni estetiche, l'istruttore e' andato a modificare il foglio di stile dell'header component, quindi *header.component.css*. RECUPERA NOTE.

Interazione tra componenti

Prima di addentrarci nell'argomento specifico, creiamo un nuovo componente, chiamato *Jumbotron*:

```
ng generate component jumbotron --path=src/app/core --module=core --export
```

E modifichiamo il suo html (*jumbotron.component.html*):

```
<div class="jumbotron jumbotron-billboard">
    <div class="img"></div>
        <div class="container">
            <div class="row">
                <div class="col-lg-12">
                    <h2>{{Titolo}}</h2>
                    <p>{{Sottotitolo}}</p>
                    <ng-container *ngIf="Show">
                        <a href="" id="SignUp" class="btn btn btn-primary btn-
lg">Accedi</a>
```

```
                        <a href="" id="SignIn" class="btn btn-success btn-
lg">Registrati</a>
                    </ng-container>
                </div>
            </div>
        </div>
</div>
```

Andiamo ad aggiungere *Titolo* e *Sottotitolo* tra le variabili in *jumbotron.component.ts*:

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

@Component({
    // [...]
})
export class JumbotronComponent implements OnInit {

    @Input() Titolo: string = '';
    @Input() Sottotitolo: string = '';
    @Input() Show: boolean = true;

    // [...]

}
```

Va poi ad aggiungere del css nel componente.

Aggiorniamo anche il *login.component.html*:

```
<app-jumbotron [Titolo]="'Ciao Pazzeska'" [Sottotitolo]="'Diventa anche tu una
ragazza di Porta Venezia'"></app-jumbotron>
<div>
<label for="userId">User Id:</label>
<input type="text" id="userId" placeholder="Nome Utente" [(ngModel)] = userId>

<label for="password">Password:</label>
<input type="text" id="password" placeholder="Password" [(ngModel)] = pasword>

<button class="btn btn-primary" (click) = gestAuth()>Connetti</button>
</div>
<div *ngIf="!autenticato">
    {{errMsg}}
</div>
```

Aver hardcodato le variabili all'interno dell'html non e' una cosa bellissima, quindi andiamo a modificare il *login.component.ts*:

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

@Component ({
    selector: 'app-login',
    templateUrl: './login.component.html',
    styleUrls: ['./login.component.css']
})
export class LoginComponent implements OnInit {

    userId: string = "Andrea";
    password: string = "";


    autenticato: boolean = true;
    consentito: boolean = true;
    errMsg: string = "Spiacente, user id e/o password sono errati. Riprovare";

    //aggiungiamo le nuove variabili:
    titolo: string = "Ciao Pazzeska";
    sottotitolo: string = "Diventa anche tu una ragazza di Porta Venezia";

    constructor() {}

    ngOnInit(): void {
    }

    gestAuth = () : void => {
        console.log(this.userId);


        if (this.userId === "Andrea" && this.password === "pupopeligroso95") {
            this.autenticato = true;
            this.consentito = true;
        }
        else {
            this.autenticato = false;
            this.consentito = false;
        }
    }

}
```

Ora che abbiamo inserito le variabili, possiamo tornare a modificare il *login.component.html*:

```
<app-jumbotron [Titolo]="titolo" [Sottotitolo]="sottotitolo"></app-jumbotron>
<div>
<label for="userId">User Id:</label>
<input type="text" id="userId" placeholder="Nome Utente" [(ngModel)] = userId>

<label for="password">Password:</label>
<input type="text" id="password" placeholder="Password" [(ngModel)] = pasword>

<button class="btn btn-primary" (click) = gestAuth()>Connetti</button>
</div>
<div *ngIf="!autenticato">
    {{errMsg}}
</div>
```