ANGULARJS

by Google

(mean stack 2)

Introducció i Arquitectura

Raúl García Lepe rglepe@Gmail.com

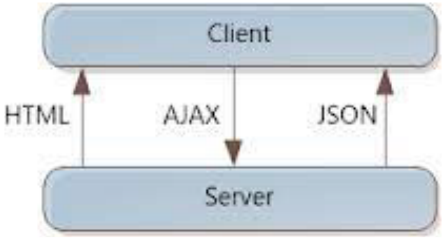
Por qué Mean Stack 2.0

- Només un llenguatge: **JavaScript** per a construir una aplicació complerta:
  - Base de Dades: **MongoDB**
  - Framework web o màquina virtual: **ExpressJS**
  - Framework Front-End: **Angular**
  - Servidor d'aplicacions Web i framework de desenvolupament Back-End: **Node.js**
- Altres avantatges:
  - Suport al patró de disseny Model-Vista-Controlador (MVC)
  - Utilització de JSON per a la transferència de dades
  - La llibreria de mòduls de Node.js és enorme
  - Open Source…

Models d'Aplicacions

- Els nous paradigmes de les Aplicacions Web

  - El navegador és la nova plataforma
  - El núvol permet l'accés a continguts sense importar el dispositiu
  - HTML5 i les seves tecnologies són la base d'aquest "*framework*"
  - El nou model d'aplicacions SPA (*Single Page Applications*), s'adapta millor a aquesta arquitectura
  - S'utilitza un patró de arquitectura tipo MVC, que facilita la separació de responsabilitats
  - Els formats de transferència d'informació tendeixen a optimitzar-se, utilitzant propostes com JSON (fonamentalment)
  - Els canvis a les pàgines *responsive* es produeixen principalment sobre la informació modificada (peticions AJAX).



Models d'Aplicacions

- Nous paradigmes per a Aplicacions Web

  - Experiència d'Usuari (UX) fluida
  - Animacions i transicions que ajuden a interpretar el diàleg amb la IU
  - Noves propostes de disseny que permeten crear llocs amb experiències similars en mòbils i dispositius d'escritori
  - Velocitat superior dels motors de JavaScript (V8/Chrome, Chakra/Edge) permeten aprofitar tot els recursos de hardware del dispositiu
  - Adaptació als dispositius
  - Noves API de JavaScript: Web Workers, Web Sockets, localStorage, sessionStorage, Notifications, File API, Drag&Drop, etc. es poden integrar amb els serveis propis d'AngularJS (1.x) y Angular (2+) per a obtenir solucions més efectives

Razones para utilizar Angular (cualquier versión)

Nou Programador	• Popularitat • Demanda • Suport i recursos • Front-End
Programador expert	• Marc de desenvolupament estructurat i ben provat • Productivitat • Consistència (i suport multiplataforma i multi-navegador)
Cap de Projecte	• Eficiència • Longevitat (Google/Microsoft donaran suport continuat en el temps)

Per què Angular

- Visualització òptima de la informació
- Facilitat de lectura i navegació
- Independència del dispositiu
- Compatibilitat navegadors
- "Feedback" continu (qualsevol acció suposa una resposta)
- Accés a dades de forma simplificada, mitjançant serveis Web
- Possibilitat de realitzar Test Unitaris
- Escalabilitat i manteniment

SINGLE PAGE APLICATIONS (SPAs)

- Sorgeixen per a millorar l'experiència d'usuari disminuint el **temps de recàrrega de la pàgina**

Sense SPAs, varies tècniques ho milloren:

- Minimitzar scripts i CSS
- Combinar imatges amb un únic *sprite*
- Retardar l'execució de JavaScript
- Maneig especial d'arxius estàtics (CDN)
- Caché de recursos…

Però és inevitable:

- Tornar a *parsear* y executar el codi CSS i JavaScript. Descarregar y *parsear* tot el codi HTML
- Encara que no més hagi canviat una petita part
- Reconstruir l'arbre DOM. *Renderitzar* el Interface
- L'usuari veu com la pàgina es construeix

SINGLE PAGE APLICATIONS (SPAs)

Què és una SPA

- Un nou enfoc per a construir aplicacions web
- Tot el codi es carrega durant la primera crida i posteriorment de forma dinàmica, sense recarregar la pàgina
- La navegació es resolc al client
- Les crides al servidor es fan de forma asíncrona. L'interfície es construeix al client

Què NO es una SPA

- Unir totes les pàgines del lloc en una i carregar-la estàticament

SINGLE PAGE APLICATIONS (SPAs)

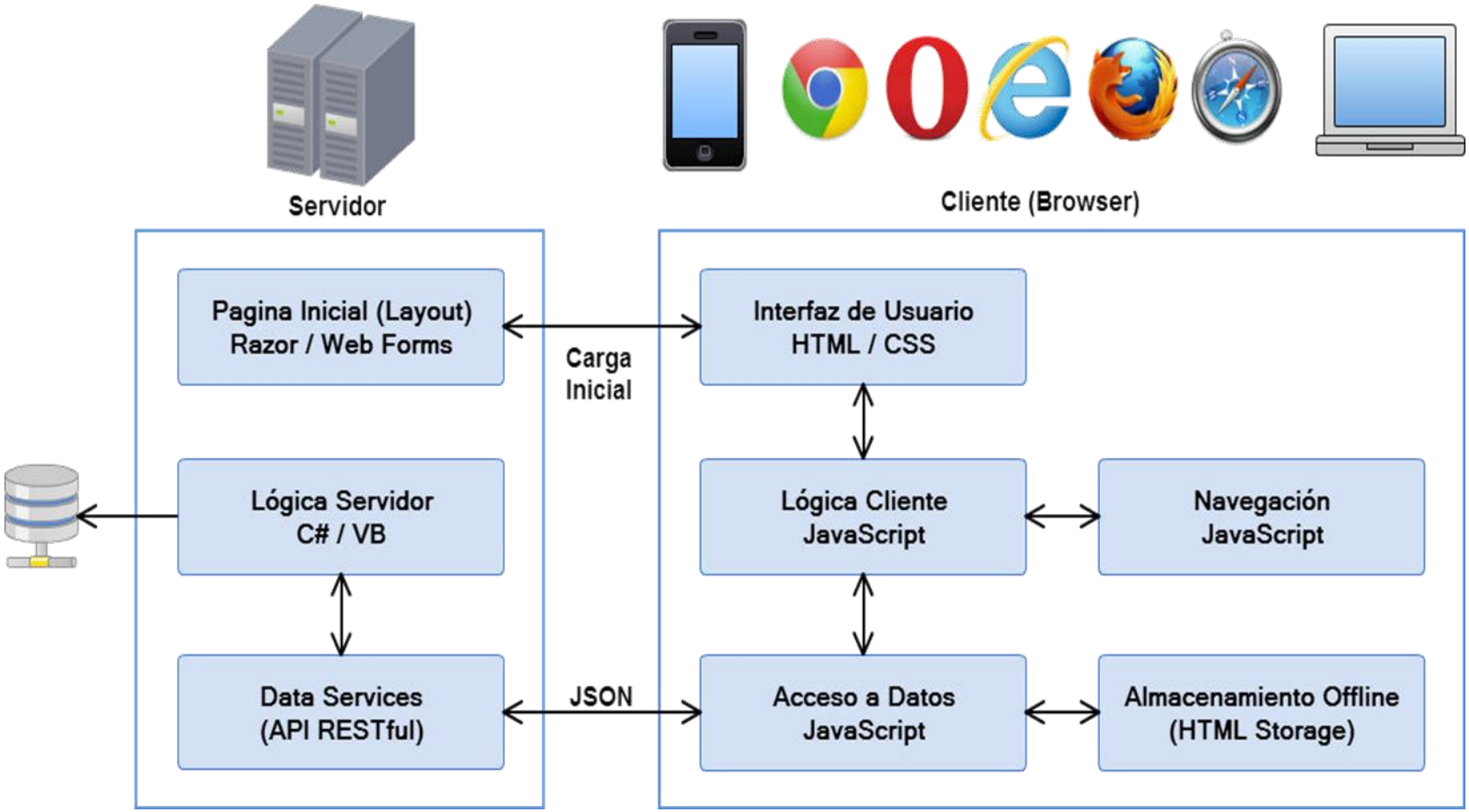
Què pot fer una SPA

- Canvis d'URL i navegació endavant i enrere
- Manipulació de DOM del costat del client
- Esperar a que la vista es carregui abans de mostrar-la
- Emmagatzemar pàgines ja carregades al client

EXEMPLES:

- GMail
- Twitter
- Facebook (semi)

Arquitectura



SINGLE PAGE APLICATIONS (SPAs)

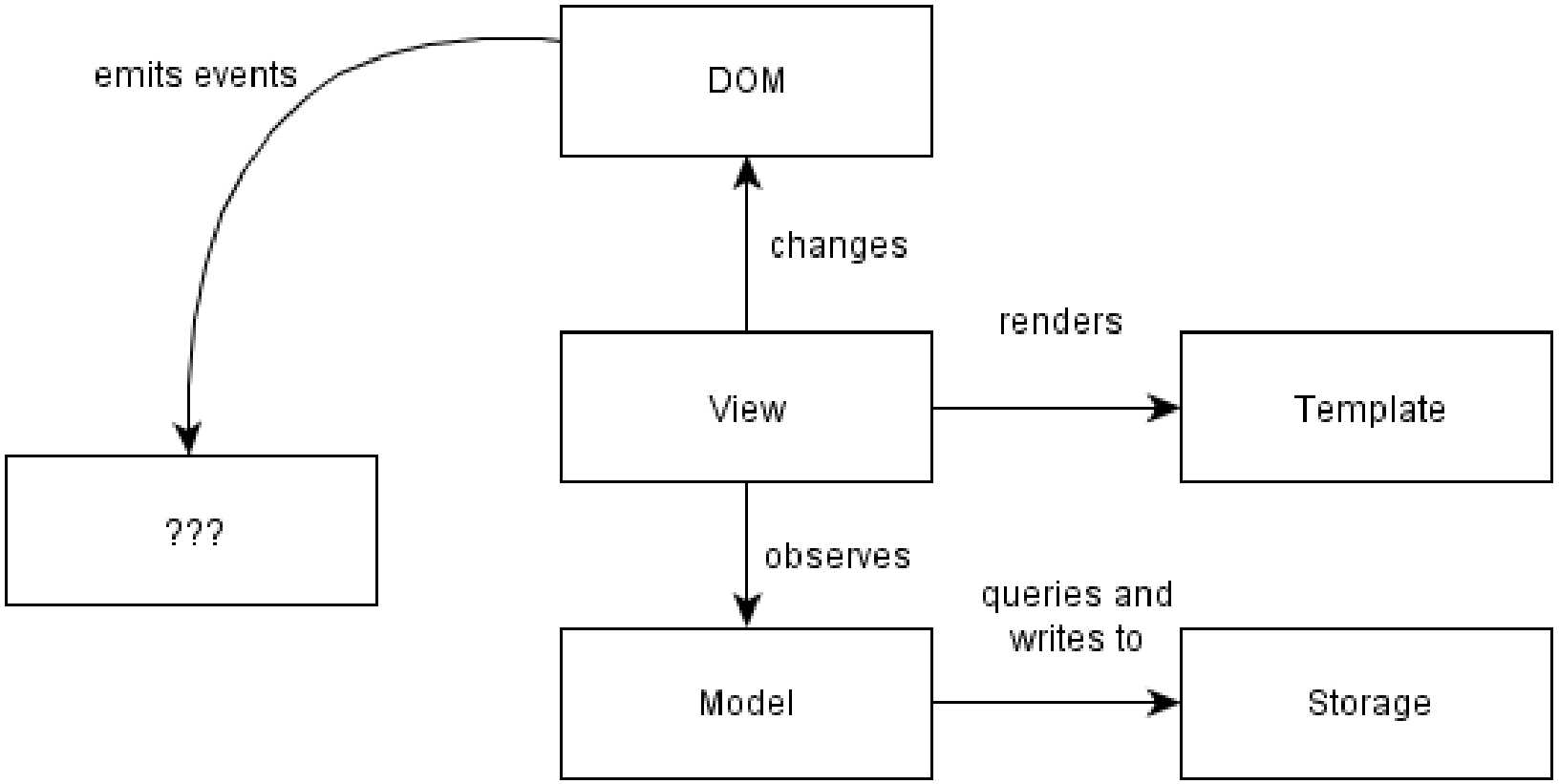
Avantatges:

- Interfície més ràpida. Manteniment més senzill. Distribució de càrrega
- Inici del desenvolupament més àgil
- La interfície és simplement altre client. Bo per a fer testing
- Perfecte per a combinar amb aplicacions mòbils

Desavantatges:

- La primera càrrega pot ser més lenta
- SEO es torna complex
- Requereix habilitat amb JavaScript
- Requereix coneixement addicional de JavaScript
- Trenca amb les convencions d'analítics, ads y widgets

Arquitectura



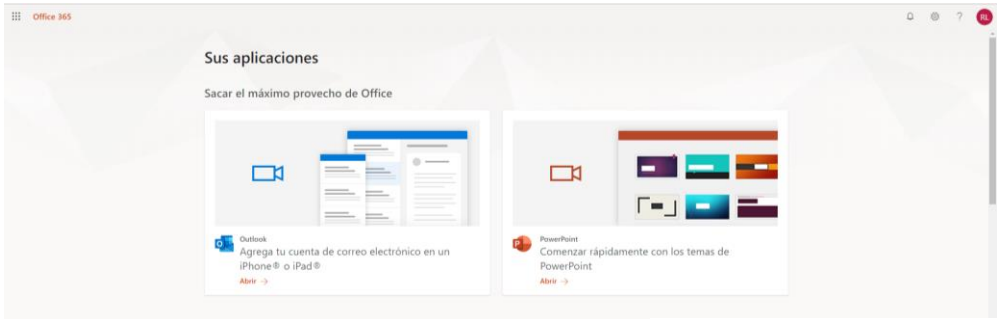
SINGLE PAGE APLICATIONS (SPAs)

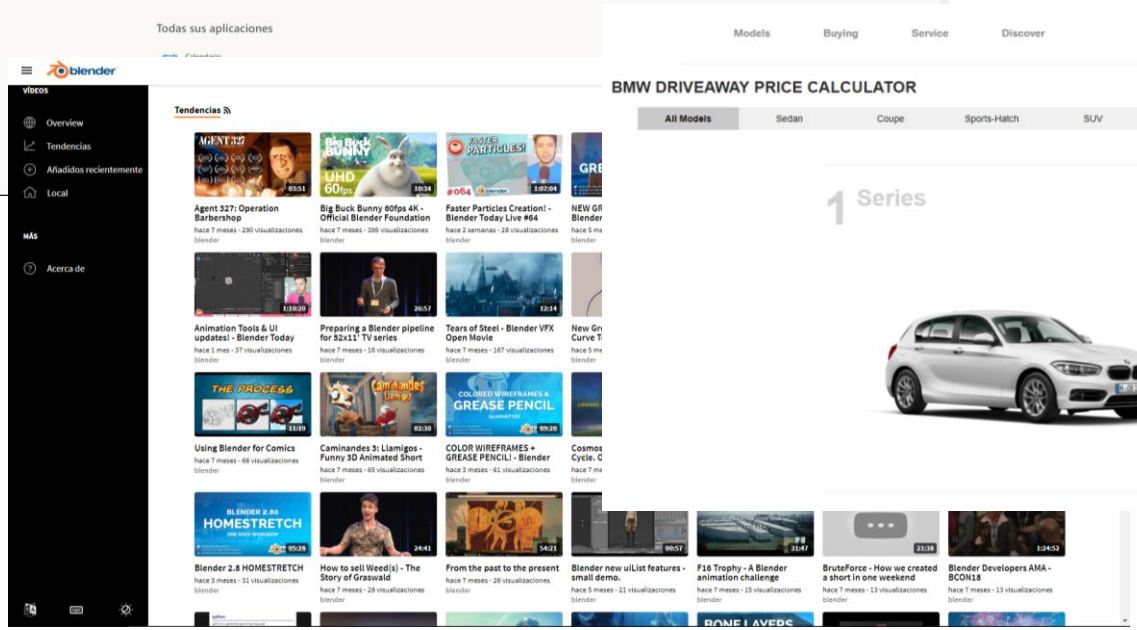
- Es trasllada la lògica de negoci no crítica al client:
  - Abans el nostre codi era 90% servidor (Java, C#) y 10% JS
  - Ara passa a ser a 50 y 50.

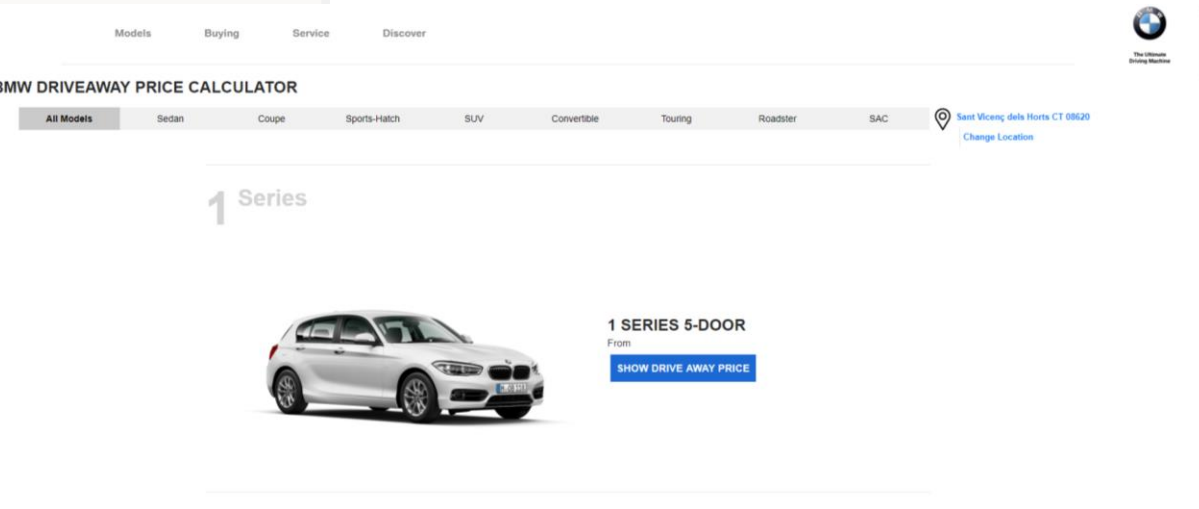
- Es fa precís un lloc diferent d'eines per a mantenir bones pràctiques

EXEMPLES APLICACIONS AMB ANGULAR

- FROM https://www.madewithangular.com/categories/angular/





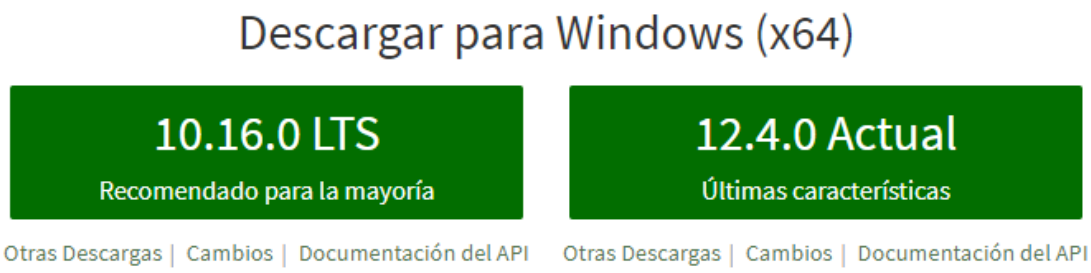


L'entorn de desenvolupament

- Directori de treball: **LOCALHOST**
- Servidor **Apache**
- **Node.JS i NPM**
- **Angular 8 CLI**
- Editor de text pla o IDE: **Visual Source Code**
- Navegador: **Chrome**

Instal·lació de Node.js

- Node és, principalment, una eina de desenvolupament back-end en JavaScript.
- Per a Angular ens interessa perquè instal·la NPM.
- Anem a la web oficial: https://nodejs.org/es/ i descarreguem la darrera versió LTS (Long time Support):



- Per a comprovar que s'ha instal·lat adequadament obrim una consola i escrivim:

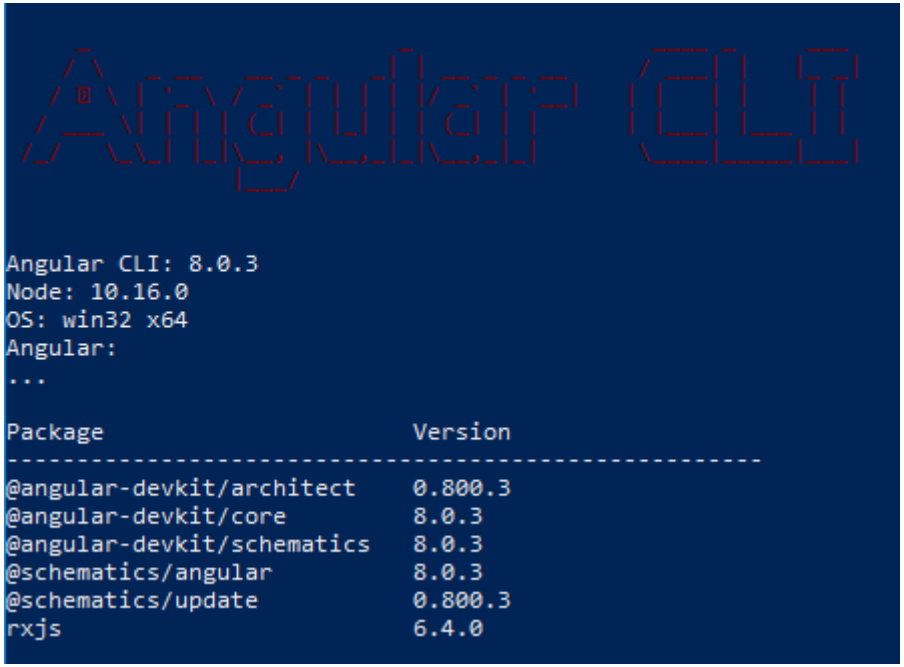
```
PS C:\WINDOWS\system32> node -v
v10.16.0
PS C:\WINDOWS\system32>
```

Instal·lació de Angular CLI

- Usem npm per a instal·lar Angular CLI (Command Line Interface) versión Angular 8

```
PS C:\WINDOWS\system32> npm install -g @angular/cli
```

- Comprovem la versió amb **ng version**:



Instal·lació de Visual Source Code

- Descarreguem i instal·lem de la pàgina oficial:

  https://code.visualstudio.com

- Per a instal·lar plugins o extensions cliquem l'icona  a la pàgina de inici:
  - Extensions interessants:
    - **Angular Essentials**
    - Color Picker
    - TS Lint

Instal·lació terminal cygwin

- Instal·lem el Package **chere** per a integrar el terminal quan configurem el Visual Studio Code

Instal·lació Visual Studio Code

- Modifiquem l'entorn d'usuari amb el nostre terminal cygwin. Per això, editem la configuració terminal per mostrar en Windows i afegim les següents línies:

```
// Cygwin, with chere package installed
"terminal.integrated.shell.windows": "C:\\Cygwin\\bin\\bash.exe",
"terminal.integrated.shellArgs.windows": ["/bin/xhere", "/bin/bash"],
```

Primera aplicació Angular

- Obrim el terminal i creen un subdirectori dins del directori on tindrem els projectes. Al meu cas, he creat el directori c:/xampp/htdocs/angular
- Creem un projecte de nou:

```
rglep@RGL-LAP /cygdrive/c/xampp/htdocs/angular/prueba_angular
$ ng new prueba-angular --prefix pr --minimal true --routing true
```

  - **ng:** ordre de terminal per a les operacions d'Angular.
  - **new :** creem un projecte de nou. El CLI (invocat per ng) s'ocuparà de crear un directori anomenat prueba-angular i col·locarà endins l'estructura bàsica de arxius
  - **--prefix pr** o abreviat **-p pr:** prefix que portaran els elements del projecte. Encara que sigui un paràmetre opcional resultarà imprescindible per a organitzar els elements del programari. Poden ser les inicials del projecte.
  - **-- minimal true** o, abreviat**, -m true:** indica al CLI el desenvolupament d'un projecte amb les funcionalitats bàsiques d'Angular. No portarà test unitaris o d'integració i, a més, incorpora tot el codi HTML, javaScript i CSS al mateix arxiu.
  - **-- routing true:** selecciona el sistema de enrutament d'Angular 8.

Primera aplicació Angular

- Estructura d'arxius del projecte:

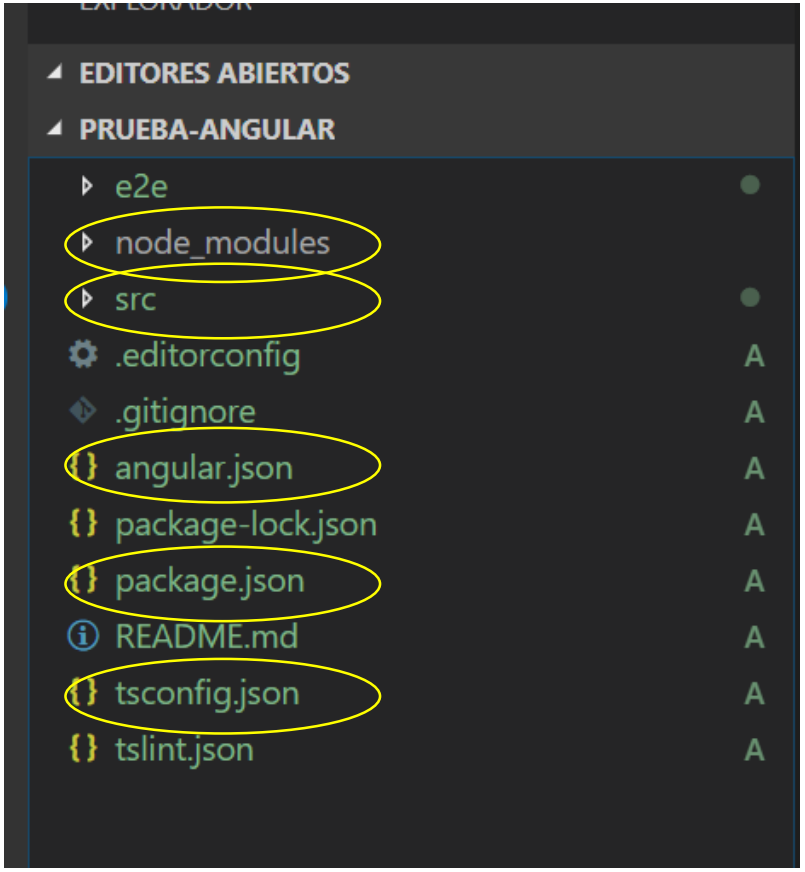
directori on **npm** instal·la totes les dependències d'Angular.

arxius bàsics del projecte:
codi Angular (HTML, *TypeScript, *CSS, etc)

Configuració global del projecte

dependències i alguns detalls del comportament del projecte

configuració de TypeScript



Primera aplicació Angular

- Estructura d'arxius del projecte, directori ***src***:
  - Arxiu **index.html:** `<pr-root></pr-root>` Fa referència a la plantilla javascript de l'arxiu **app.component.ts**

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

@Component({
  selector: 'pr-root',
  template: `
    <!--The content below is only a placeholder and can be replaced.-->
    <div style="text-align:center">
      <h1>
        Welcome to {{title}}!
      </h1>
      <img width="300" src="
    </div>
    <h2>Here are some links to help you start: </h2>
    <ul>
      <li>
        <h2><a target="_blank" rel="noopener" href="https://angular.io/tutorial">
      </li>
      <li>
        <h2><a target="_blank" rel="noopener" href="https://github.com/angular/an
      </li>
      <li>
        <h2><a target="_blank" rel="noopener" href="https://blog.angular.io/">Ang
      </li>
    </ul>
    <router-outlet></router-outlet>
  `,
  styles: []
})
export class AppComponent {
  title = 'pr';
}
```

Encadenament de processos de càrrega

- Tot comença amb la definició dins l'arxiu **angular.json** (angular.cli.json en versions <6) sobre quin és el mòdul de bootstrap i quina és la pàgina d'inici:

```
      "favicon.ico"
    ],
    "index": "index.html",
    "main": "main.ts",
    "polyfills": "polyfills.ts",
    "test": "test.ts",
    "tsconfig": "tsconfig.app.json",
    "testTsconfig": "tsconfig.spec.json",
    "prefix": "a01",
    "styles": [
      "styles.css"
    ],
    "scripts": [],
    "environmentSource": "environments/environment.ts",
    "environments": {
      "dev": "environments/environment.ts",
      "prod": "environments/environment.prod.ts"
    }
```

Encadenament de processos de càrrega

- **Webpack:** crida a les instruccions ng build ng serve
- Aquí, es produeix un procés complex de filtrat i empaquetat de fragments Javascript, HTML i CSS
- **main.ts** càrrega **AppModule** i ho registra amb la instrucció:

```
platform.bootstrapModule (AppModule)
```

- …que llança tota seqüència.

- Quan l'arxiu indicat (index.html o un altre), ha estat "injectat" amb la informació necessària, ja podem obrir la pàgina en qualsevol navegador sobre el port indicat (4200, per defecte).
- **app.module** importa **app.component** (entre altres coses), i és **app.component** el que defineix la interfície d'usuari.
- Aquest funcionament permet tenim més llibertat a l'hora de triar la cadena de funcionament (fins i tot la inicial, associada a la instrucció anterior:

Engegant l'Aplicació

- Podem, abans de fer l'engegada, afegir algun estil CSS o usar altres llibreries com BootStrap.css. En aquest cas, afegim un arxiu del mateix nom "styles.css" al root de l'aplicació.
- Per engegar-la hi ha múltiples opcions, ja que depèn del servidor Web que volem muntar (val qualsevol).
- De manera predeterminada, s'usa un servidor lleuger (lite-server), que s'instal·la dinàmicament des de npm, quan llancem l'aplicació amb el comando npm start.
- Si estem utilitzant Visual Studio Code, podem obrir una finestra terminal en la part inferior de l'editor de codi mitjançant CTRL+ñ o en el menú "Veure/Terminal Integrat".
- Des d'aquí podem llançar qualsevol instrucció, com des de la finestra exterior.

Engegant l'Aplicació

- Teclegem **npm start**.
  - Es llança el mandat ng serve i, després d'uns segons, es duu a terme una transpil·lació de TypeScripts a JavaScripts:

```
rglep@RGL-LAP /cygdrive/c/xampp/htdocs/angular/prueba-angular
$ npm start

> prueba-angular@0.0.0 start C:\xampp\htdocs\angular\prueba-angular
> ng serve

** NG Live Development Server is listening on localhost:4200, open your browser on http://localhos
t:4200/ **
Date: 2018-08-28T15:55:26.581Z
Hash: 047bbd23048c7669e45b
Time: 5412ms
chunk {inline} inline.bundle.js (inline) 3.85 kB [entry] [rendered]
chunk {main} main.bundle.js (main) 19.4 kB [initial] [rendered]
chunk {polyfills} polyfills.bundle.js (polyfills) 555 kB [initial] [rendered]
chunk {styles} styles.bundle.js (styles) 41.5 kB [initial] [rendered]
chunk {vendor} vendor.bundle.js (vendor) 8.46 MB [initial] [rendered]

webpack: Compiled successfully.
```

- Ara tenim preparat el port 4200 per córrer la aplicació

Engegant l'Aplicació: Modificant paràmetres

- Detenim l'àplicació i modifiquem el paràmetre **start** de l'arxiu **package.json:**

```
"start": "ng serve -o --port 4500",
```

  - **--open(-o):** obra el navegador automàticament
  - **--port:** per a canviar el port de escuxa

Engegant l'Aplicació

- El compilador es llança en mode "watch" ("tsc–w") Qualsevol canvi en el codi es reflectirà immediatament en el navegador

- Es pot garantir això, configurant V.S.Code per desar automàticament després d'una curta espera:
  En "Preferències/Configuració d'usuari", afegim l'entrada:

```
"files.autoSave": "afterDelay"
```

AFEGIR LLIBRERIES DE TERCERS

- Afegim llibreries perquè estiguin disponibles per a diferents entorns (desenv, pre, pro):

```
npm install --save jquery
npm install --save moment
npm install --save milligram
npm install --save bootstrap
```

- Les llibreries s'instal·len al directori node_modules. Comprovem al package.json que les llibreries s'han afegit a l'apartat de dependències
- Ara haurem de referenciar-les des de l'arxiu angular.json:

```
"styles": [
        "./node_modules/bootstrap/dist/css/bootstrap.min.css",
        "./node_modules/milligram/dist/milligram.min.css",
        "styles.css"
      ],
"scripts": [
        "./node_modules/jquery/dist/jquery.min.js",
        "./node_modules/bootstrap/dist/js/bootstrap.min.js",
        "./node_modules/moment/min/moment.min.js"
      ],
```

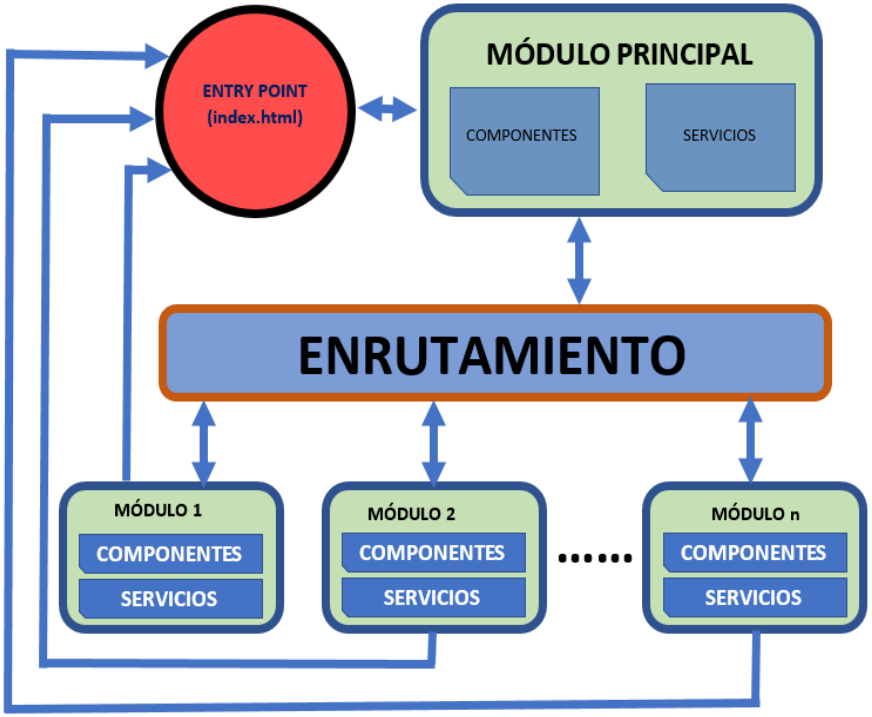
Estructura d'una aplicació Angular 2+

- Les aplicacions Angular 2+ es formen a partir de **components** individuals construïts de forma independent:

- Els components obtenen funcionalitats dels **serveis** disponibles.

Estructura d'una aplicació Angular 2+

- Els components i els serveis s'organitzen en **mòduls** que es comuniquen entre si per un sistema d'enrutament. Bàsicament l'esquema d'una aplicació Angular seria el següent:



Mòduls

- Els mòduls ens permeten fer agrupacions lògiques de components
- Per defecte: **app.module.ts.**
- Els mòduls, com tot els elements d'Angular, són **classes**.
  - A la part superior **s'importen** els elements amb els que es va treballar dins del mòdul:

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

import { AppRoutingModule } from './app-routing.module';

import { AppComponent } from './app.component';
```

  - Després es troba el **decorador** d'Angular que el defineix com a mòdul (@NgModule) i aporta l'operativa a la classe.

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

  - Al finalitzar es fa l'**export** de la classe

```
export class AppModule { }
```

Aplicació complerta

- Nova aplicació sense l'opció minimal = true:

```
ng new angular01 -p a01 --routing true
```

- Es generen nous fitxers:

  - **app-routing.module.ts**. Codificació dels enrutaments
  - **app.component.css**. Permet col·locar aquí els nostres CSS
  - **app.component.html**. Arxiu propi de HTML
  - **app.component.spec.ts**. Arxiu per a test unitaris i d'integració
  - **app.component.ts**. Amb noves propietats en el decorador: **templateURL** i **styleUrls**
  - **app.module.ts**. mòdul AppModule.

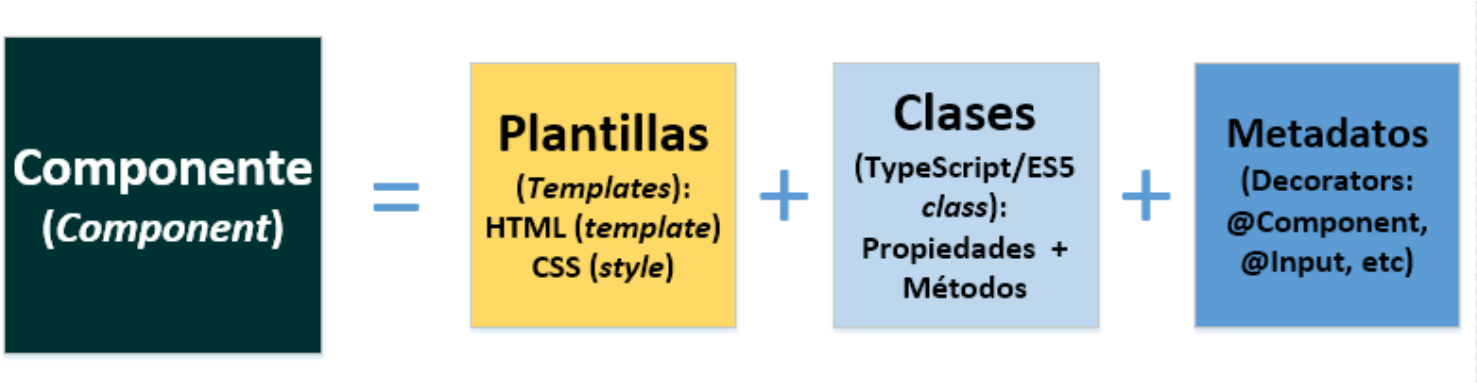
Component inicial

- S'importa la definició del *decorator* **@Component** de "@angular/core".
- Es defineix l'IU del component dins del decorador que porta la definició **templateUrl:** apuntant a un fragment HTML separat en un arxiu (una vista).
- La pròpia definició de la classe, a la qual s'associa el decorador (exportada, per poder ser usada externament).

  (Més endavant portarà la seva pròpia lògica de negoci, però aquí la seva única funció és servir de suport al decorador.)

Components Angular 2+

- Un component està format per tres blocs principals:



- Una **plantilla**, amb la part de HTML (view) i estils CSS
- Una **classe**, amb la part funcional (model) i disposa de propietats i mètodes
- **Metadades**, configuren el comportament de tot dos i la seva relació (controller)

Creació d'un component nou

- S'utilitza l'instrucció **ng** amb el modificador generate i el nom del component.
- El component es crearà dins del mòdul principal.
- Crearem un component que sigui el header a visualitzar dins de totes les vistes de l'aplicació (una pàgina – múltiples vistes – SPA).
- Per a que es pugui utilitzar aquest component fora del mòdul s'ha de crear amb el modificador –export:

```
ng generate component header --export
o, abreviat:
ng g c header --export
```

Creació d'un component nou

- Ara, cal referencial el selector del component nou, que trobem dins del header.component.ts (<a01-header>), al app.component.html quedan la pàgina d'aquesta manera:

```
<a01-header></a01-header>
 <h1>
   Página principal de la aplicación
 </h1>
 <router-outlet></router-outlet>
```

- Per a que es mostrin els canvis nous s'ha de rengegar el servidor perquè ho tenim configurat com *aot (Ahead Of Time)*

Exercici 1

- Afegir estils al component creat:
  - HeaderComponent:

```
header {
  padding: 2rem;
  top: 0;
  left: 0;
  right: 0;
  background-color: #00b4b3;
}

  h1 {
    color: #840101;
  }
```

- Afegir estils al component principal: AppComponent:

```
h1 {
  margin-left: 4rem;
}
```

Exercici 1(cont)

- Afegir estils global a la aplicació:
  - styles.css

```
body{
    margin-top: 0;
    margin-left: 0;
    margin-right: 0;
}
```

Cabecera de la aplicación

Página principal de la aplicación

Exercici 2

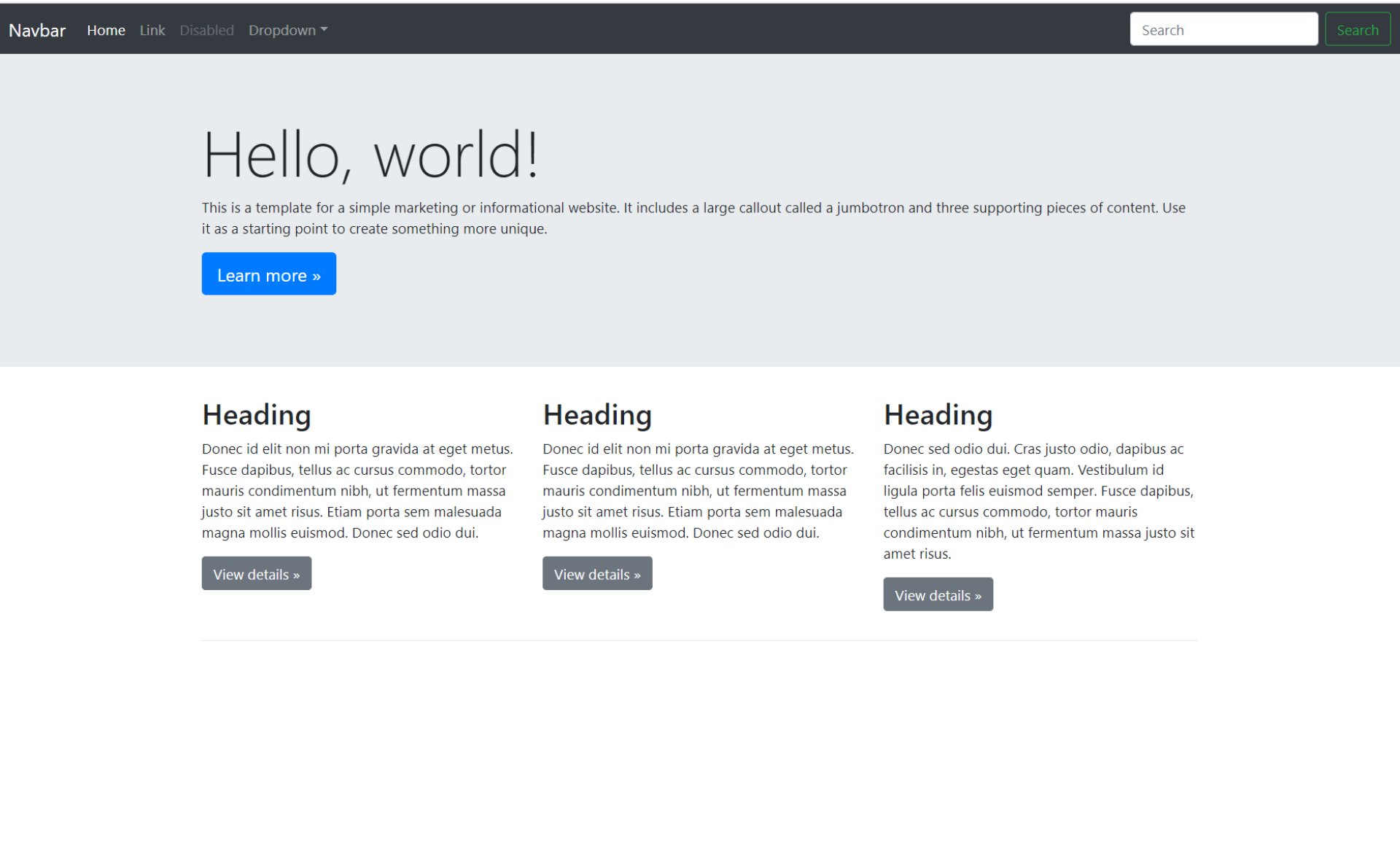
- Afegir llibreries externes a la aplicació:
  - jQuery i BootStrap
  - *tips:*
    - *instal·lar-les amb npm install **nom-llibreria** –save*
    - *modificar l'arxiu .angular-cli.json a les etiquetes: "styles" i "script" per a indicar al CLI on es trobem les llibreries dins de la carpeta node_modules*

Cabecera de la aplicación

Página principal de la aplicación

Exercici 3

- Personalització del *jumbotron* de la aplicació:
  - Buidem la fulla de estils del projecte (styles.css)
  - Modifiquem header.component.css deixant només color blau per al fons de la capçalera i deixem el color vermell fosc del ròtul h1
  - Afegim l'element class = "jumbotron" a la etiqueta <header> del header.component.html
  - Modifiquem la etiqueta <h1> del app.component.html per a incorporar la classe *container-fluid* de Bootstrap


Navbar
Home
Link
Disabled
Dropdown
Search
Search
Hello, world!
This is a template for a simple marketing or informational website. It includes a large callout called a jumbotron and three supporting pieces of content. Use it as a starting point to create something more unique.
Learn more »
Heading
Donec id elit non mi porta gravida at eget metus. Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut fermentum massa justo sit amet risus. Etiam porta sem malesuada magna mollis euismod. Donec sed odio dui.
View details »
Heading
Donec id elit non mi porta gravida at eget metus. Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut fermentum massa justo sit amet risus. Etiam porta sem malesuada magna mollis euismod. Donec sed odio dui.
View details »
Heading
Donec sed odio dui. Cras justo odio, dapibus ac facilisis in, egestas eget quam. Vestibulum id ligula porta felis euismod semper. Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut fermentum massa justo sit amet risus.
View details »

Nomenclatura de components

- El nom no pot coincidir amb noms de components d'Angular per això:
  - Pots afegir un prefix o un sufix al nom, quan es crea amb l'instrucció **new**. Ex.: a01header
- Els noms amb més d'una paraula millor amb un guió, guió baix o notació *camelCase*