



Tutorial de una aplicación con login – Manejo de sesiones en Angular

PEV: Javier G. Palacios
ARGENTINA PROGRAMA

Prerrequisitos:

- Node.js

Revisa que node.js esté instalado en tu computador.

Si no está instalado descargar de <https://nodejs.org/en/download/>

Revisa la versión **node.js**:

```
node -v
```

npm:

```
npm -v
```

Angular CLI (es la herramienta con la que vamos a poder generar aplicaciones donde nos permite crear, depurar y publicar)

Instalar Angular CLI:

```
npm install -g @angular/cli
```

Deberías tener la última versión de Angular CLI.

Visual Studio Code

Se puede descargar de la siguiente página: <https://code.visualstudio.com/>

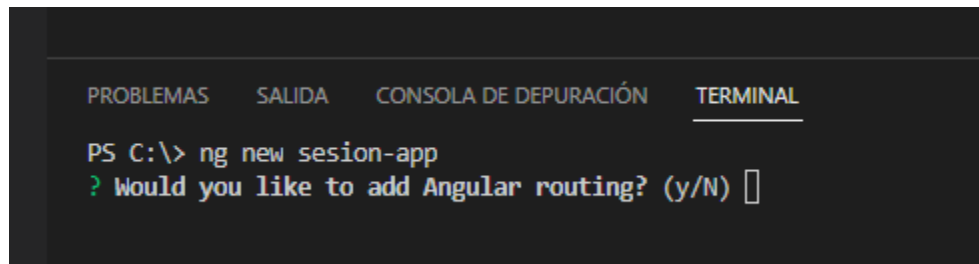
Creando nuestra primera aplicación

Usaremos angular-cli para crear y generar nuestros componentes. Generará servicios, enrutadores, componentes y directivas.

Para crear un nuevo proyecto Angular con Angular-cli, solo ejecuta:

```
ng new sesion-app
```

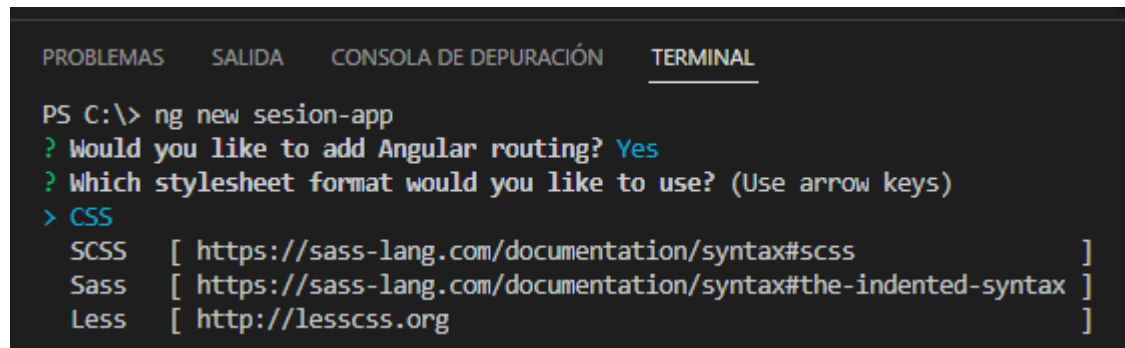
El proyecto se generará automáticamente.



```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS C:\> ng new sesion-app
? Would you like to add Angular routing? (y/N) 
```

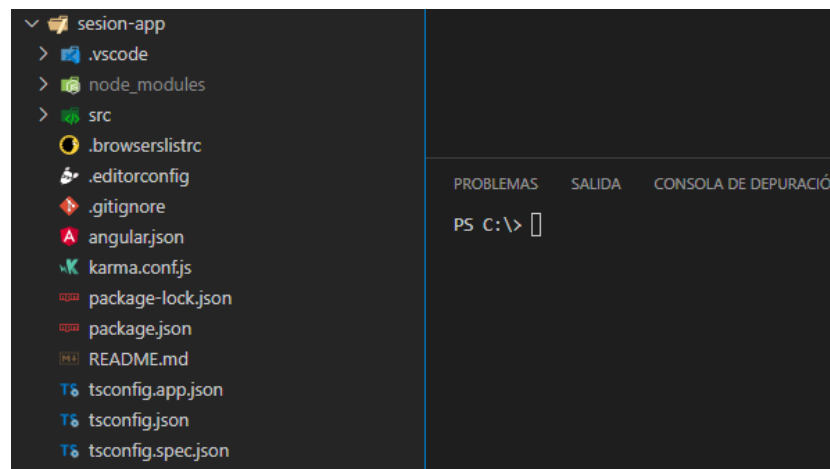
Ingresamos “y”, para agregar el componente que nos permite manejar las rutas en angular.



```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS C:\> ng new sesion-app
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? (Use arrow keys)
> CSS
  SCSS  [ https://sass-lang.com/documentation/syntax#scss ]
  Sass  [ https://sass-lang.com/documentation/syntax#the-indented-syntax ]
  Less  [ http://lesscss.org ]
```

Luego abrimos la carpeta creada con nuestra aplicación en visual studio code.

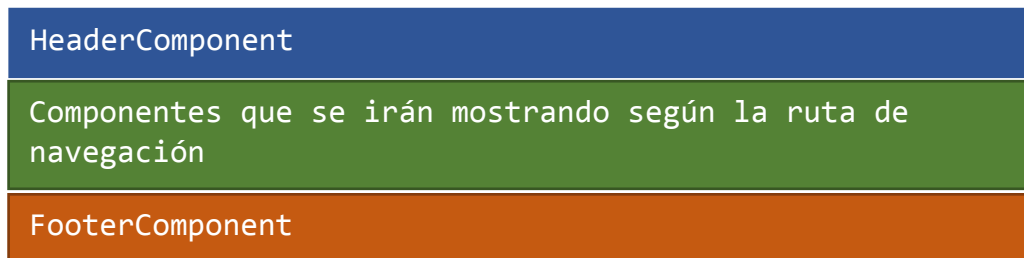


Para tener en cuenta, nuestra aplicación estará dividida en los siguientes componentes:

- Header
- Login: será un componente **público** para cualquier persona
- Home: será un componente **público** para cualquier persona
- Dashboard: será un componente **privado** que se mostrará luego de iniciar sesión.
- Footer

De los cuales el Header y el Footer component, siempre se mostrarán y cuando el usuario inicie sesión mediante el componente Login podrá acceder a Dashboard y según como cambien las rutas en la barra de direcciones del navegador, se irán mostrando los otros componentes en pantalla.

Ósea en pantalla se verá así:



Antes de continuar instalaremos Bootstrap para los estilos de nuestra aplicación:

Usaremos el siguiente comando (actualmente la última versión es la 5.2.1):

```
npm install bootstrap@5.2.1
```

```
PS C:\sesion-app> npm install bootstrap@5.2.1

added 2 packages, removed 1 package, and audited 919 packages in 5s

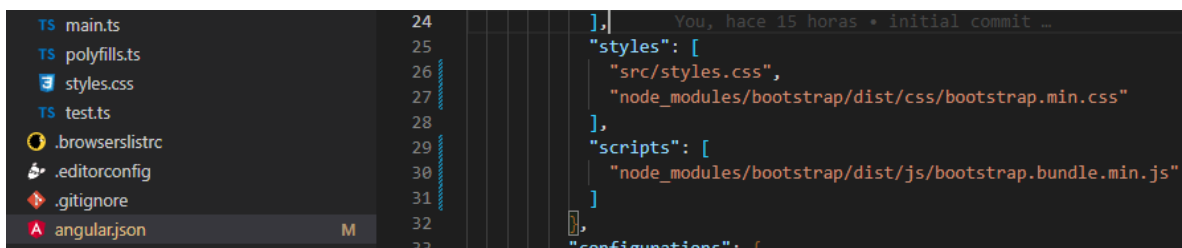
124 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS C:\sesion-app> |
```

Antes de poder aplicar estilos, debemos agregar en el archivo **angular.json** en la sección “styles”, el archivo css de bootstrap, y en scripts el archivo js de bootstrap:

```
"styles": [  
  "src/styles.css",  
  "node_modules/bootstrap/dist/css/bootstrap.min.css"  
],  
"scripts": [  
  "node_modules/bootstrap/dist/js/bootstrap.bundle.min.js"  
]
```

Debe quedar así:



A continuación, crearemos los cinco componentes nombrados anteriormente y lo haremos dentro de la carpeta components:

```
ng generate component components/header
```

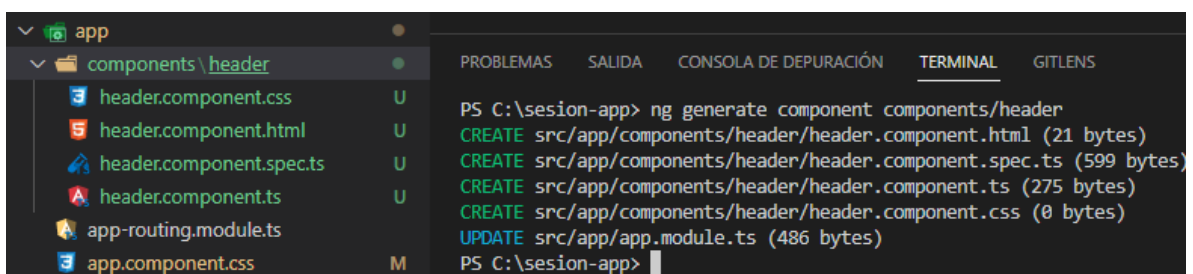
```
ng generate component components/login
```

```
ng generate component components/home
```

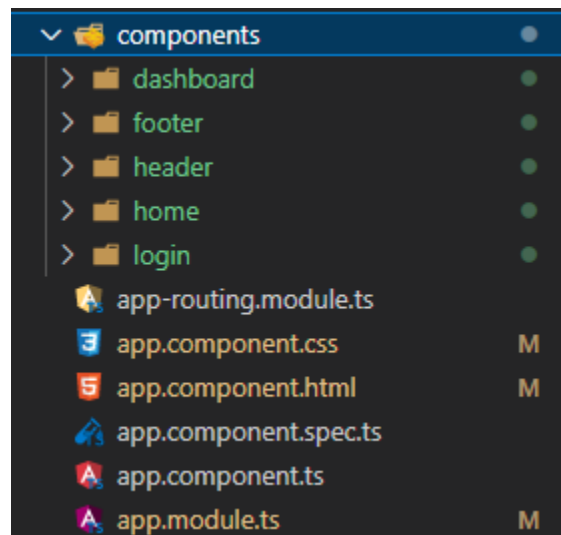
```
ng generate component components/dashboard
```

```
ng generate component components/footer
```

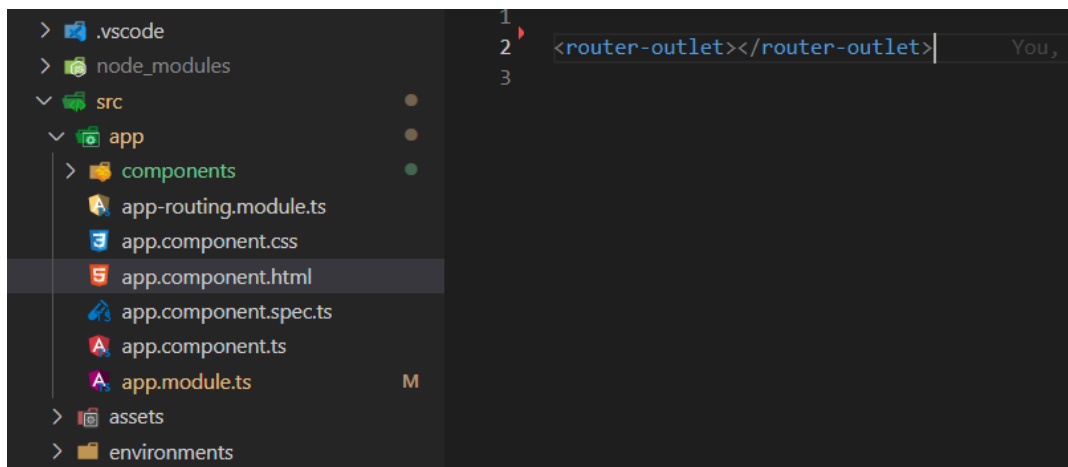
Para el primer componente:



Quedando así nuestra aplicación con los cinco componentes creados:

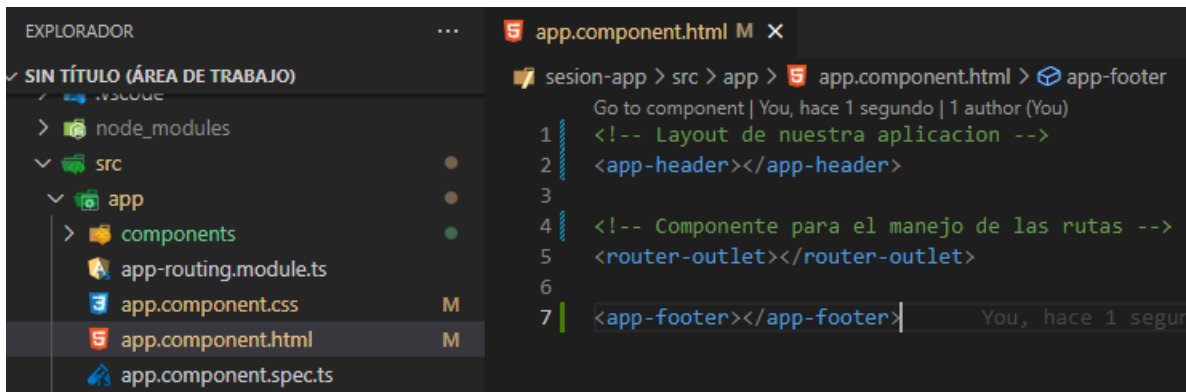


Lo siguiente va a ser limpiar el archivo **app.component.html**, eliminando todo el código en ese archivo, menos las etiquetas del router componet de angular, entonces nos queda así:



El componente `<router-outlet></router-outlet>` es el que va realizar el cambio de componentes en pantalla según la ruta que tengamos en la barra de direcciones del navegador web del usuario.

Ahora, vamos a armar la estructura o diseño (en ingles se lo suele llamar layout) de nuestra aplicación según como vimos anteriormente en la página 4.



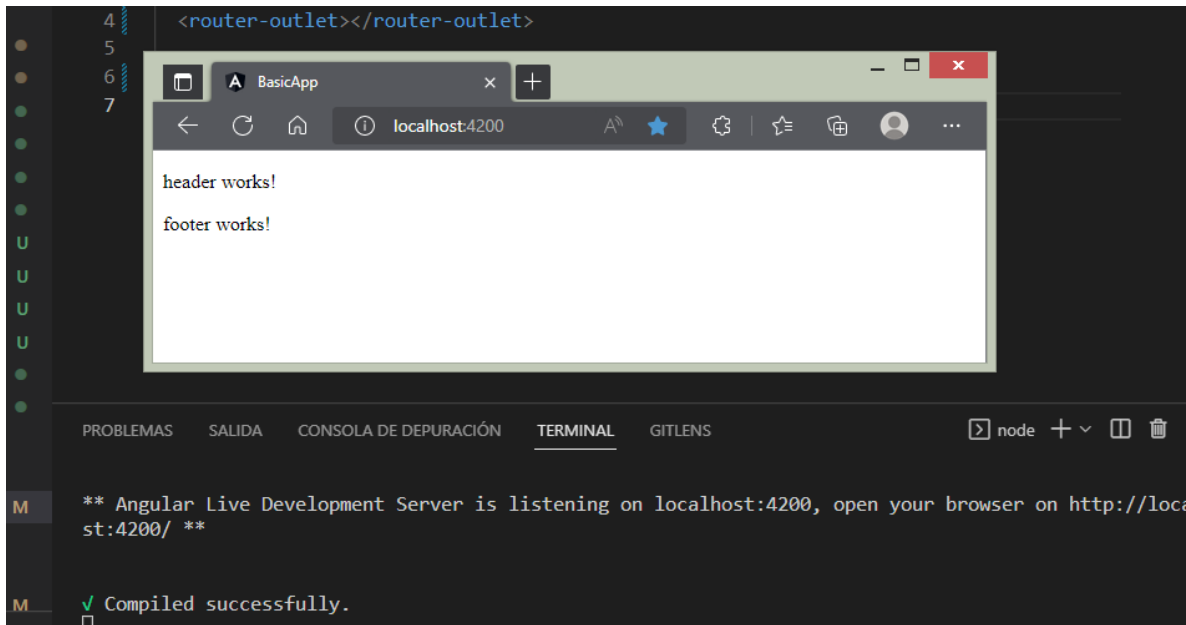
The screenshot shows the Visual Studio Code interface. On the left, the 'EXPLORADOR' (Explorer) sidebar displays the project structure: 'node_modules', 'src', and 'app'. Under 'app', there is a 'components' folder and files 'app-routing.module.ts', 'app.component.css', 'app.component.html', and 'app.component.spec.ts'. The 'app.component.html' file is selected. The main editor area shows the content of 'app.component.html' with the following code:

```
1  Go to component | You, hace 1 segundo | 1 author (You)
2  <!-- Layout de nuestra aplicacion -->
3  <app-header></app-header>
4
5  <!-- Componente para el manejo de las rutas -->
6  <router-outlet></router-outlet>
7  <app-footer></app-footer>
```

Si en este punto ejecutamos el siguiente comando:

```
ng serve
```

Veremos en la dirección: <http://localhost:4200/> en nuestro navegador web la siguiente pantalla.

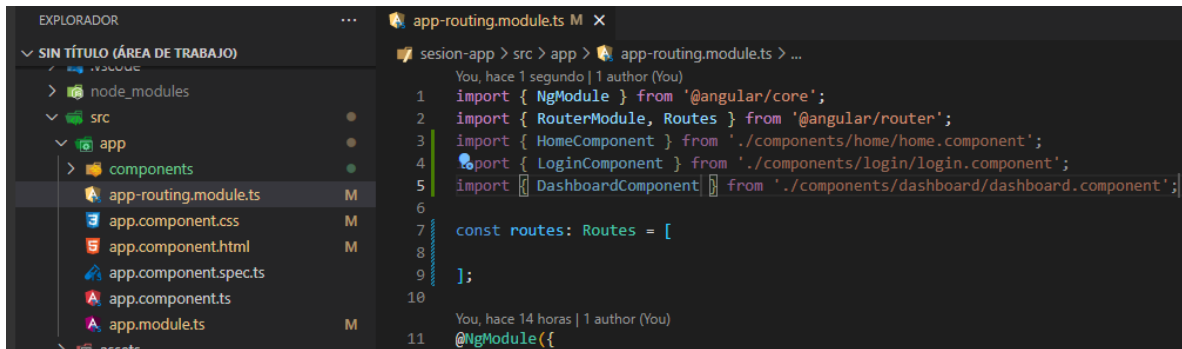


El siguiente paso es ir al archivo **app-routing.module.ts**, allí adentro en donde dice:

```
const routes: Routes = [];
```

agregaremos las rutas a los componentes que se mostraran en pantalla.

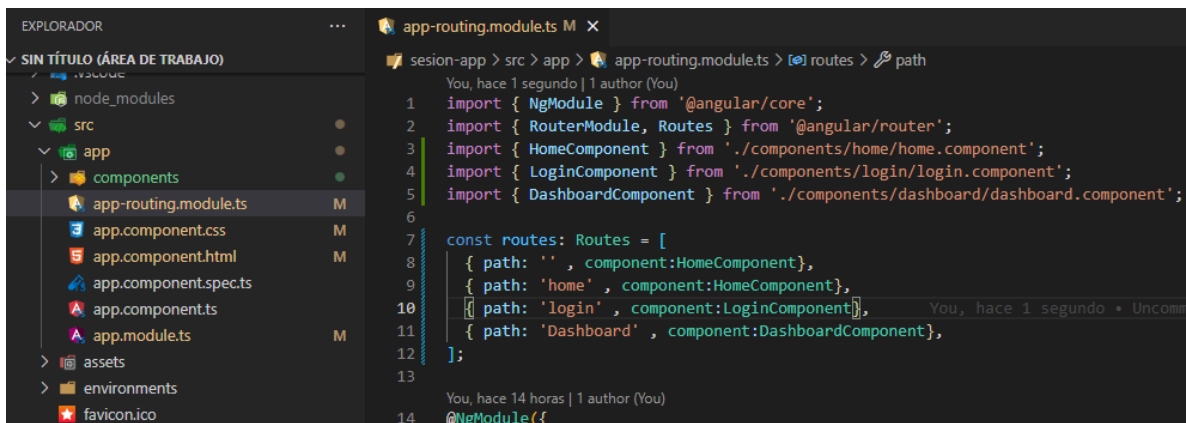
Primero debemos importarlos:



```
1 import { NgModule } from '@angular/core';
2 import { RouterModule, Routes } from '@angular/router';
3 import { HomeComponent } from '../components/home/home.component';
4 import { LoginComponent } from '../components/login/login.component';
5 import { DashboardComponent } from '../components/dashboard/dashboard.component';
6
7 const routes: Routes = [
8
9 ];
10
11 @NgModule({
```

Luego agregamos las rutas, una ruta por cada componente que queremos mostrar, teniendo en cuenta que:

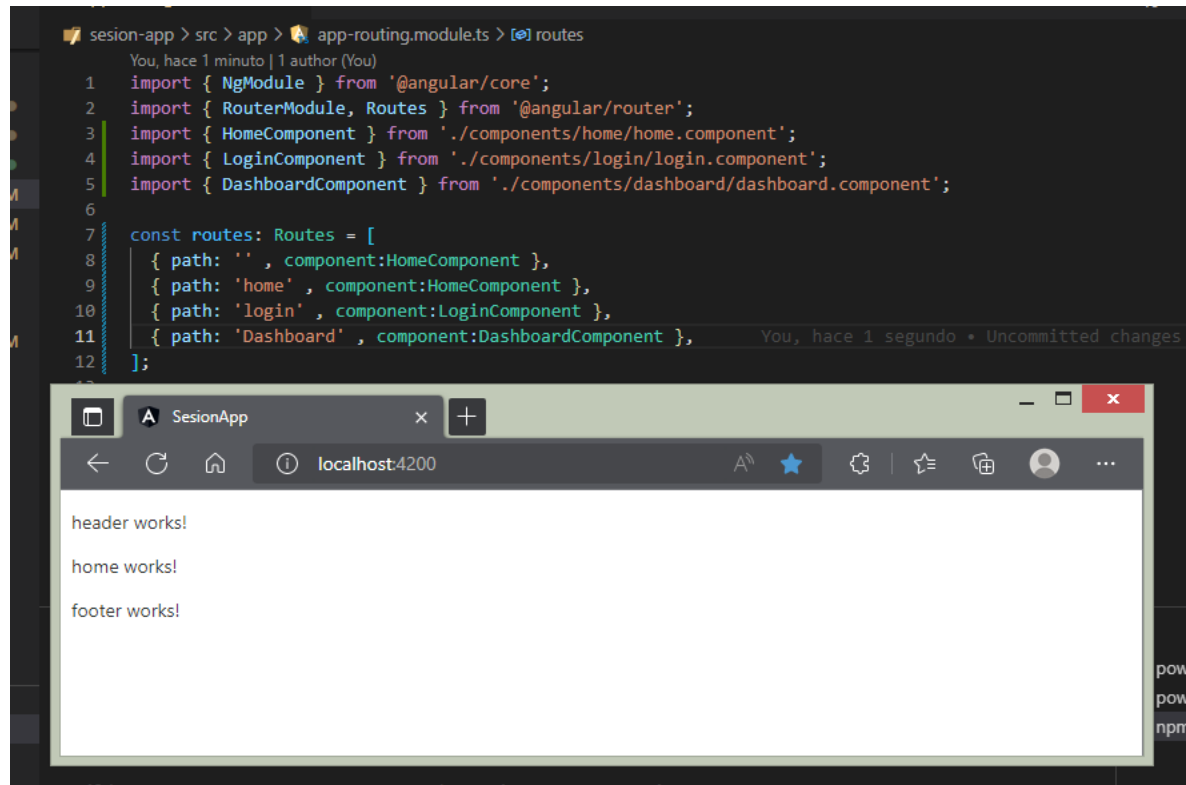
- **path**: define la ruta virtual de nuestra aplicación.
- **component**: define el componente que le dice al enrutador que componente corresponde al seleccionar dicha ruta.



```
1 import { NgModule } from '@angular/core';
2 import { RouterModule, Routes } from '@angular/router';
3 import { HomeComponent } from '../components/home/home.component';
4 import { LoginComponent } from '../components/login/login.component';
5 import { DashboardComponent } from '../components/dashboard/dashboard.component';
6
7 const routes: Routes = [
8   { path: '', component: HomeComponent },
9   { path: 'home', component: HomeComponent },
10  { path: 'login', component: LoginComponent },
11  { path: 'Dashboard', component: DashboardComponent },
12 ];
13
14 @NgModule({
```

Bien, ahora que ya tenemos las rutas y sus componentes asociados registrados en el módulo **app-routing.module.ts**, podemos probarlas en el navegador web.

Como el HomeComponent es el componente por defecto, ósea el que se muestra sin haber ingresado ninguna ruta en la barra de direcciones, angular procede a cargarlo y mostrarlo en pantalla, pero como también está registrada la ruta 'home', si la ingresamos se muestra la misma pantalla:



Corrección:

```
{ path: 'dashboard' , component:DashboardComponent },
```

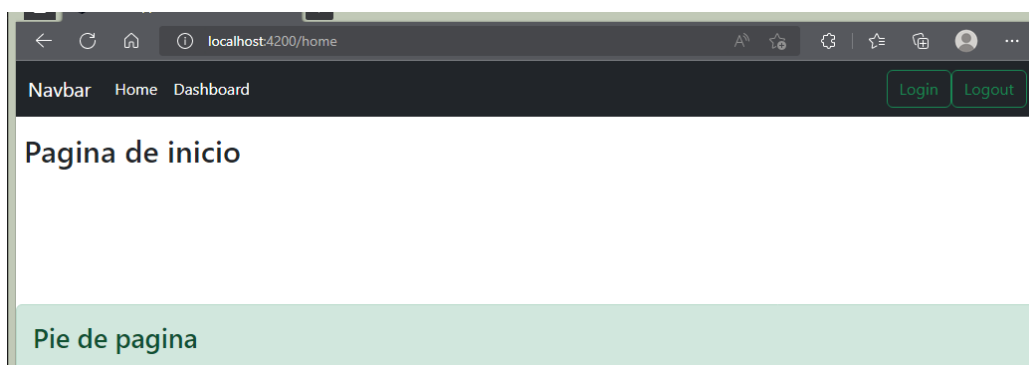
Va con minúscula dashboard.

Luego agregaremos un menú y unos estilos así:

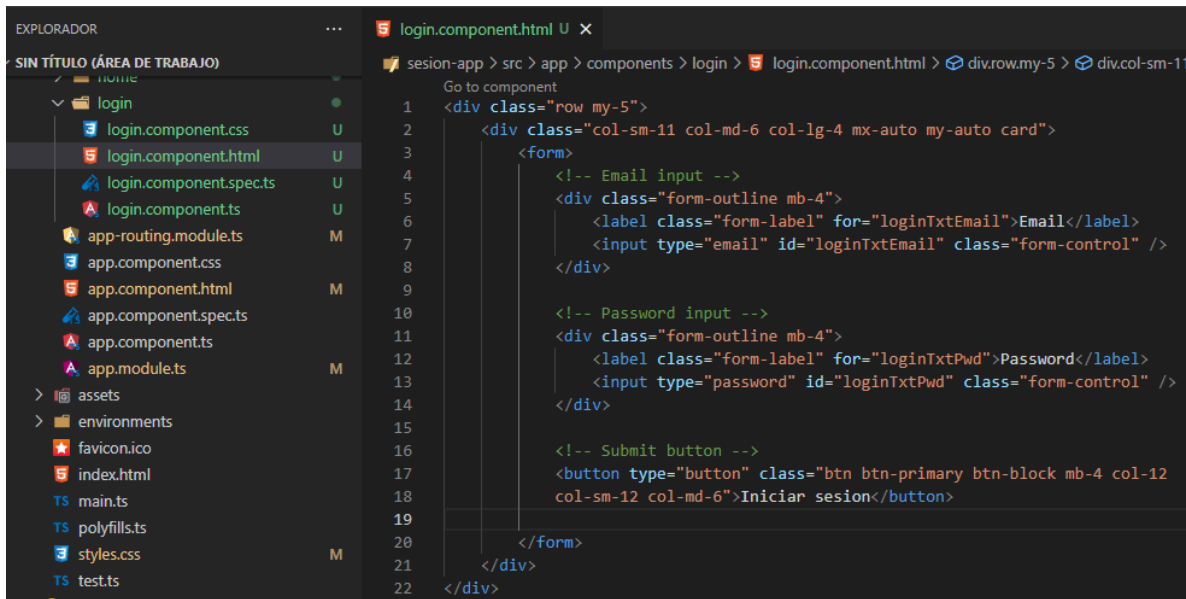
```
header.component.html U x
session-app > src > app > components > header > header.component.html > header > nav.navbar.navbar-expand-lg.navbar-dark.bg-dark > div.container-fluid
Go to component
1 <header>
2   <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
3     <div class="container-fluid">
4       <a class="navbar-brand" href="#">Navbar</a>
5       <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent"
6         aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
7         <span class="navbar-toggler-icon"></span>
8       </button>
9       <div class="collapse navbar-collapse" id="navbarSupportedContent">
10        <ul class="navbar-nav me-auto mb-2 mb-lg-0">
11          <li class="nav-item">
12            <a class="nav-link active" aria-current="page" href="#">Home</a>
13          </li>
14
15          <li class="nav-item">
16            <a class="nav-link active" aria-current="page" href="#">Dashboard</a>
17          </li>
18        </ul>
19        <form class="d-flex">
20          <button class="btn btn-outline-success" type="button">Login</button>
21          <button class="btn btn-outline-success" type="button">Logout</button>
22        </form>
23      </div>
24    </div>
25  </nav>
26
27
28 </header>
```

```
Archivo Editar Seleccion Ver Ir Ejecutar Terminal Ayuda
footer.component.html U x
session-app > src > app > components > footer > footer.component.html
Go to component
1 <footer class="fixed-bottom">
2
3   <div class="alert alert-success">
4     <h3>Pie de pagina</h3>
5   </div>
6
7 </footer>
```

Bien nos queda así:



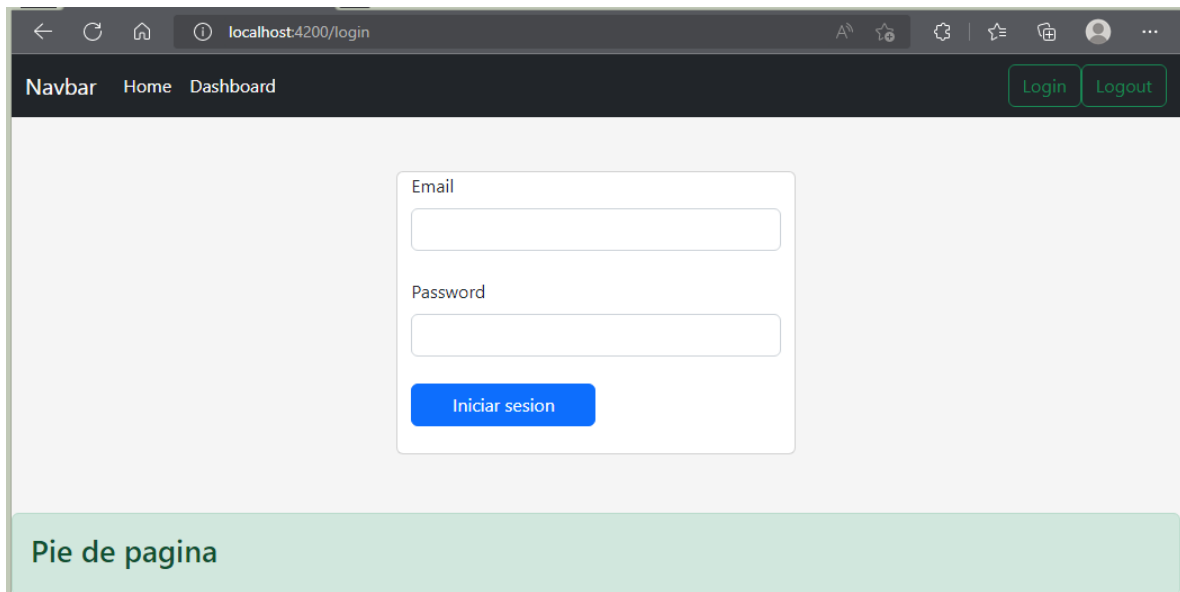
Construiremos un formulario de login bien simple:



```
1 <div class="row my-5">
2   <div class="col-sm-11 col-md-6 col-lg-4 mx-auto my-auto card">
3     <form>
4       <!-- Email input -->
5       <div class="form-outline mb-4">
6         <label class="form-label" for="loginTxtEmail">Email</label>
7         <input type="email" id="loginTxtEmail" class="form-control" />
8       </div>
9
10      <!-- Password input -->
11      <div class="form-outline mb-4">
12        <label class="form-label" for="loginTxtPwd">Password</label>
13        <input type="password" id="loginTxtPwd" class="form-control" />
14      </div>
15
16      <!-- Submit button -->
17      <button type="button" class="btn btn-primary btn-block mb-4 col-12
18        col-sm-12 col-md-6">Iniciar sesion</button>
19    </form>
20  </div>
21 </div>
```

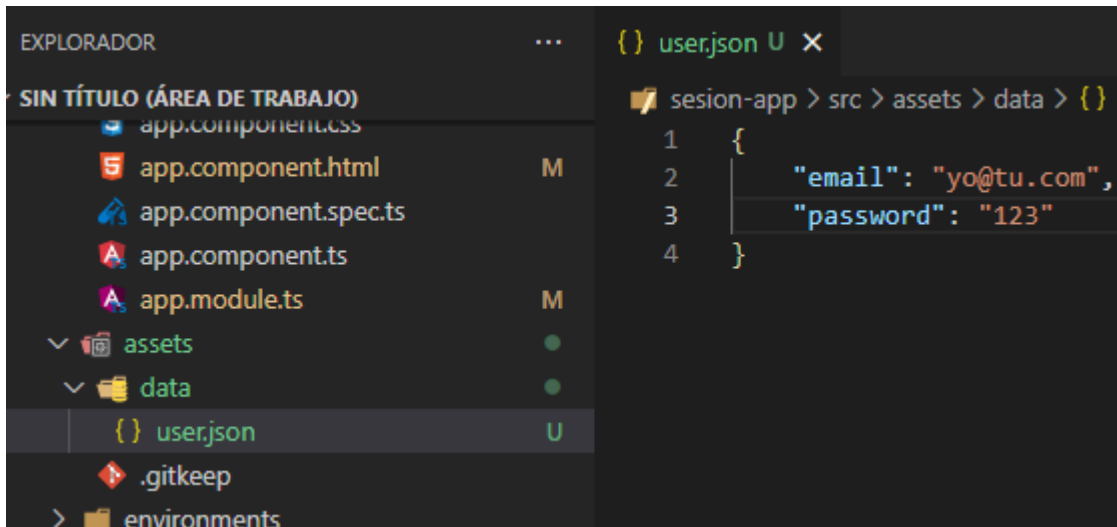
A los dos inputs los he llamado así, al de Email(loginTxtEmail) y al del Password(loginTxtPwd), luego los usaremos para recibir esos dos datos y pasarlos al controlador de login y de allí al servicio de autorización.

En pantalla hasta aquí se ve así:



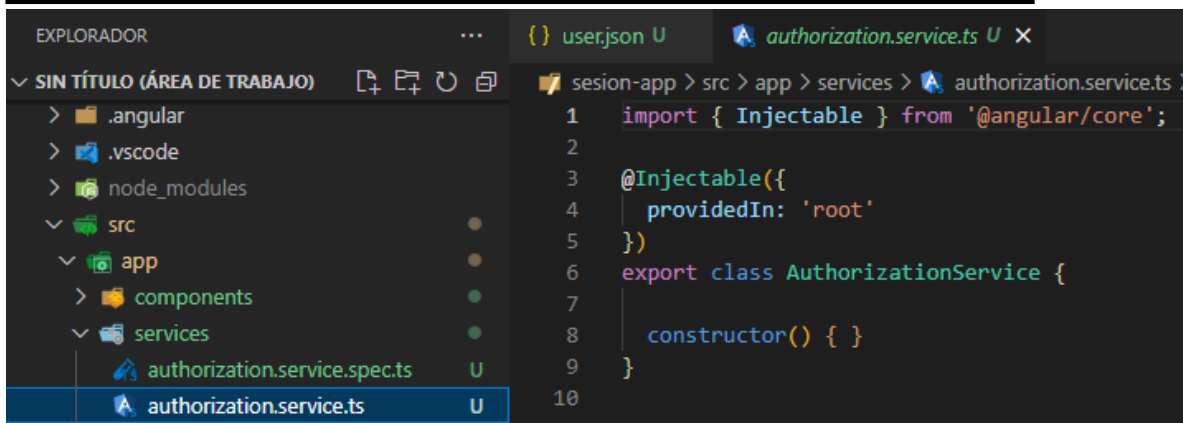
Ahora procederemos a crear toda la funcionalidad relacionada a la lógica de programación del login.

Empezaremos creando la carpeta **data** dentro de **assets**, luego dentro de data el archivo **user.json** para simular nuestra base de datos.

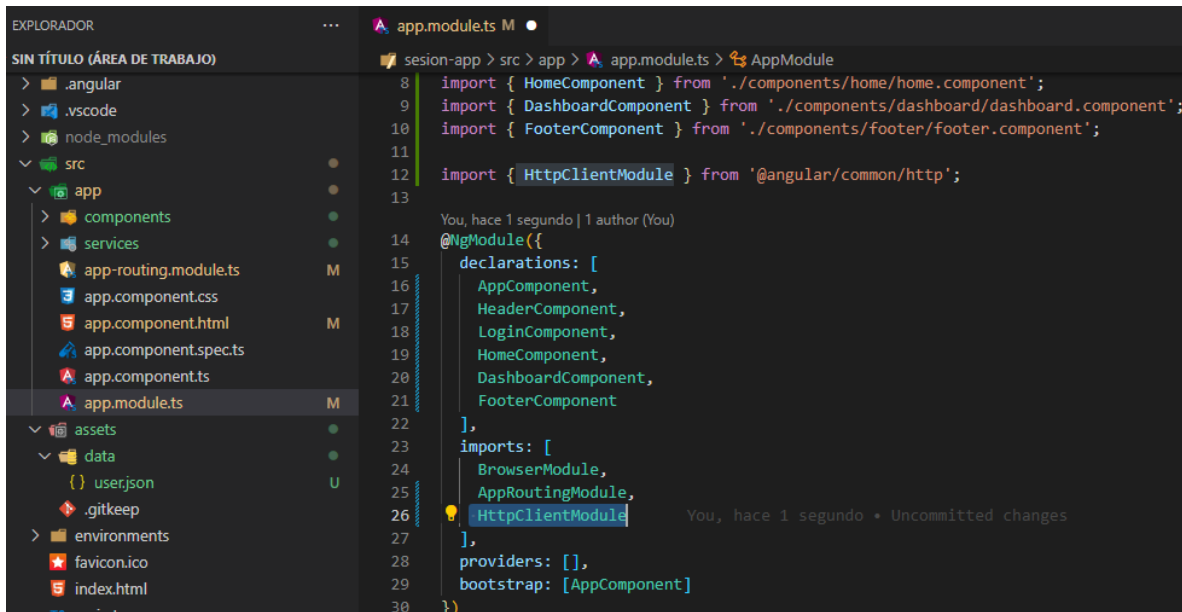


Después creamos el servicio:

```
ng generate service services/authorization
```



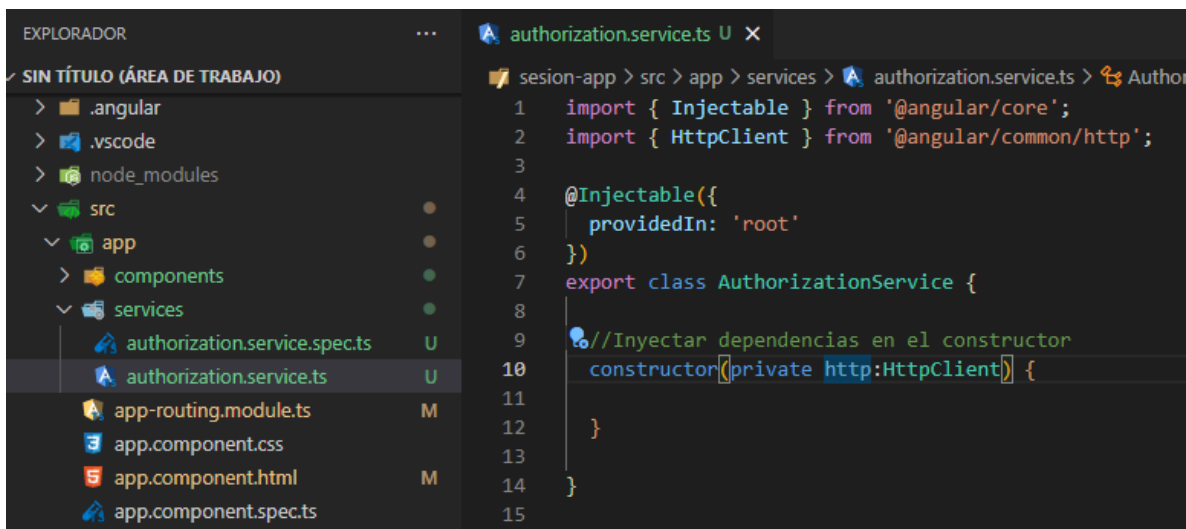
Ahora importamos el módulo de angular `HttpClientModule`, sirve para poder realizar llamadas a una API Rest.



```
EXPLORADOR
SIN TÍTULO (ÁREA DE TRABAJO)
> .angular
> .vscode
> node_modules
  src
    app
      components
      services
        app-routing.module.ts
        app.component.css
        app.component.html
        app.component.spec.ts
        app.component.ts
        app.module.ts
      assets
      data
        user.json
      .gitkeep
      environments
      favicon.ico
      index.html
      main.ts

session-app > src > app > app.module.ts > AppModule
8 import { HomeComponent } from './components/home/home.component';
9 import { DashboardComponent } from './components/dashboard/dashboard.component';
10 import { FooterComponent } from './components/footer/footer.component';
11
12 import { HttpClientModule } from '@angular/common/http';
13
14 You, hace 1 segundo | 1 author (You)
15 @NgModule({
16   declarations: [
17     AppComponent,
18     HeaderComponent,
19     LoginComponent,
20     HomeComponent,
21     DashboardComponent,
22     FooterComponent
23   ],
24   imports: [
25     BrowserModule,
26     AppRoutingModule,
27     HttpClientModule
28   ],
29   providers: [],
30   bootstrap: [AppComponent]
31 })
```

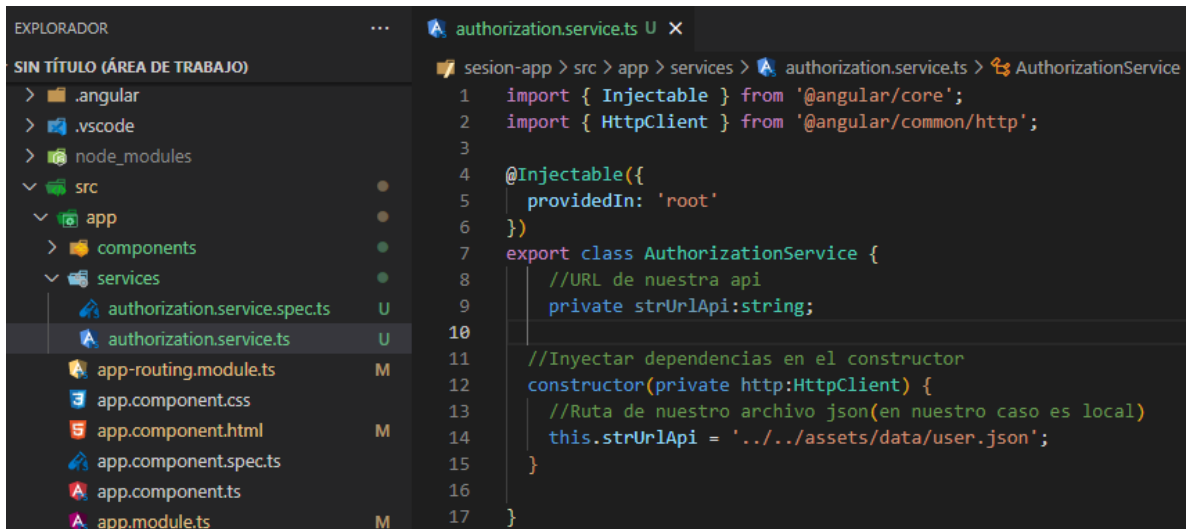
Procedemos a inyectar la dependencia en el constructor:



```
EXPLORADOR
SIN TÍTULO (ÁREA DE TRABAJO)
> .angular
> .vscode
> node_modules
  src
    app
      components
      services
        authorization.service.spec.ts
        authorization.service.ts
        app-routing.module.ts
        app.component.css
        app.component.html
        app.component.spec.ts

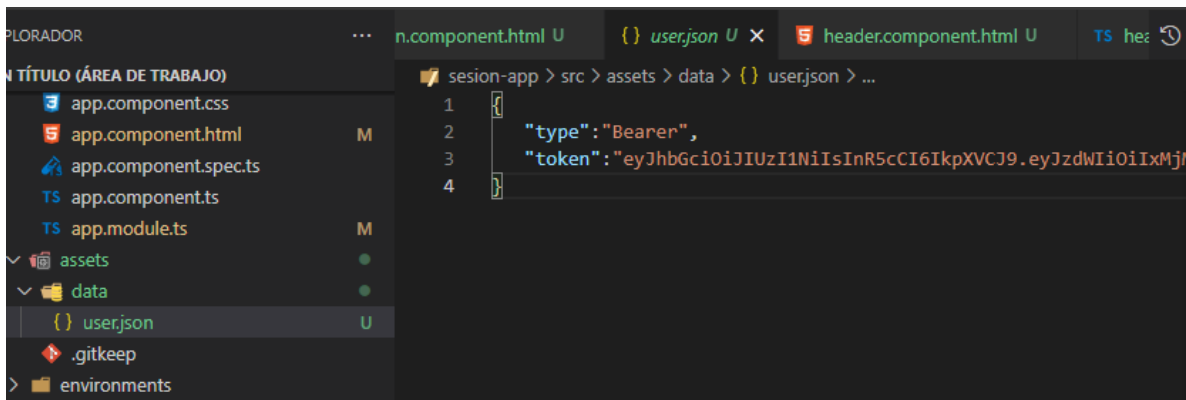
session-app > src > app > services > authorization.service.ts > AuthorizationService
1 import { Injectable } from '@angular/core';
2 import { HttpClient } from '@angular/common/http';
3
4 @Injectable({
5   providedIn: 'root'
6 })
7 export class AuthorizationService {
8
9   //Inyectar dependencias en el constructor
10  constructor(private http:HttpClient) {
11
12  }
13
14 }
15
```

Creemos un atributo para guardar la ruta al archivo user.json, el cual tendrá el token de autorización, esto lo hacemos así por ahora para simular la conexión a una API Rest:



```
1 import { Injectable } from '@angular/core';
2 import { HttpClient } from '@angular/common/http';
3
4 @Injectable({
5   providedIn: 'root'
6 })
7 export class AuthorizationService {
8   //URL de nuestra api
9   private strUrlApi:string;
10
11   //Inyectar dependencias en el constructor
12   constructor(private http:HttpClient) {
13     //Ruta de nuestro archivo json(en nuestro caso es local)
14     this.strUrlApi = '../assets/data/user.json';
15   }
16
17 }
```

Nota: con ../ subimos de nivel una carpeta, con dos ../ subimos dos niveles y así sucesivamente. Luego bajamos con / a la carpeta que queremos encontrar el archivo.



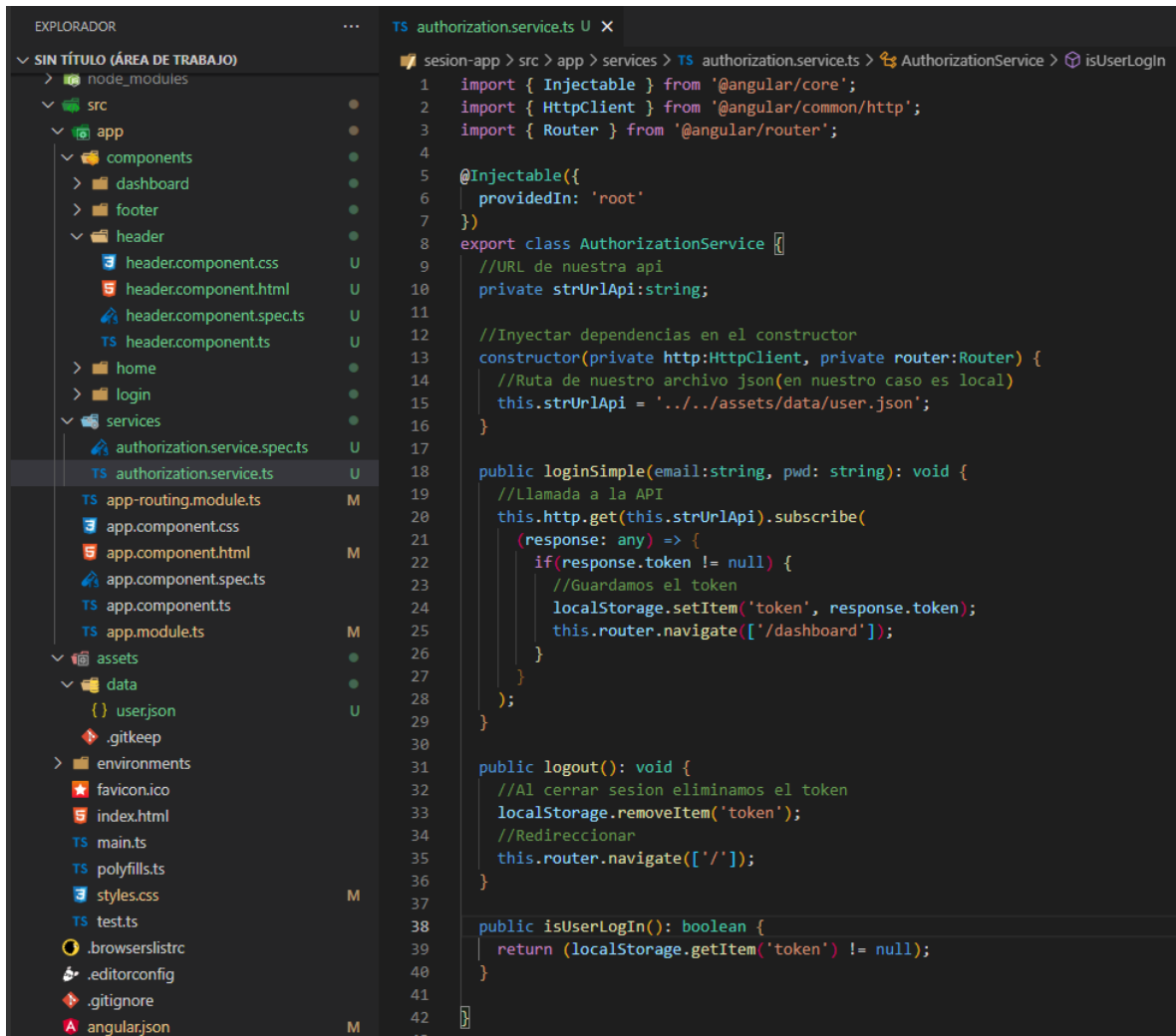
```
1 {
2   "type": "Bearer",
3   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwiaXNjaWkiOiJkbG91IiwiaWF0IjoxNTE2MzIyODQyLCJpc29udGkiOiJkbG91In0.eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwiaXNjaWkiOiJkbG91IiwiaWF0IjoxNTE2MzIyODQyLCJpc29udGkiOiJkbG91In0"
4 }
```

Usaremos un token de ejemplo, copiado de <https://jwt.io/>

```
{
  "type": "Bearer",
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwiaXNjaWkiOiJkbG91IiwiaWF0IjoxNTE2MzIyODQyLCJpc29udGkiOiJkbG91In0.eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwiaXNjaWkiOiJkbG91IiwiaWF0IjoxNTE2MzIyODQyLCJpc29udGkiOiJkbG91In0"
}
```

Bearer: es un estándar.

Luego vamos a nuestro servicio `AuthorizationService` y agregamos un método de login, uno de logout y uno para preguntar si el usuario esta logueado(`isUserLogin()`), el cual devuelve `true` si se creó el token de autorización en el `localStorage`:

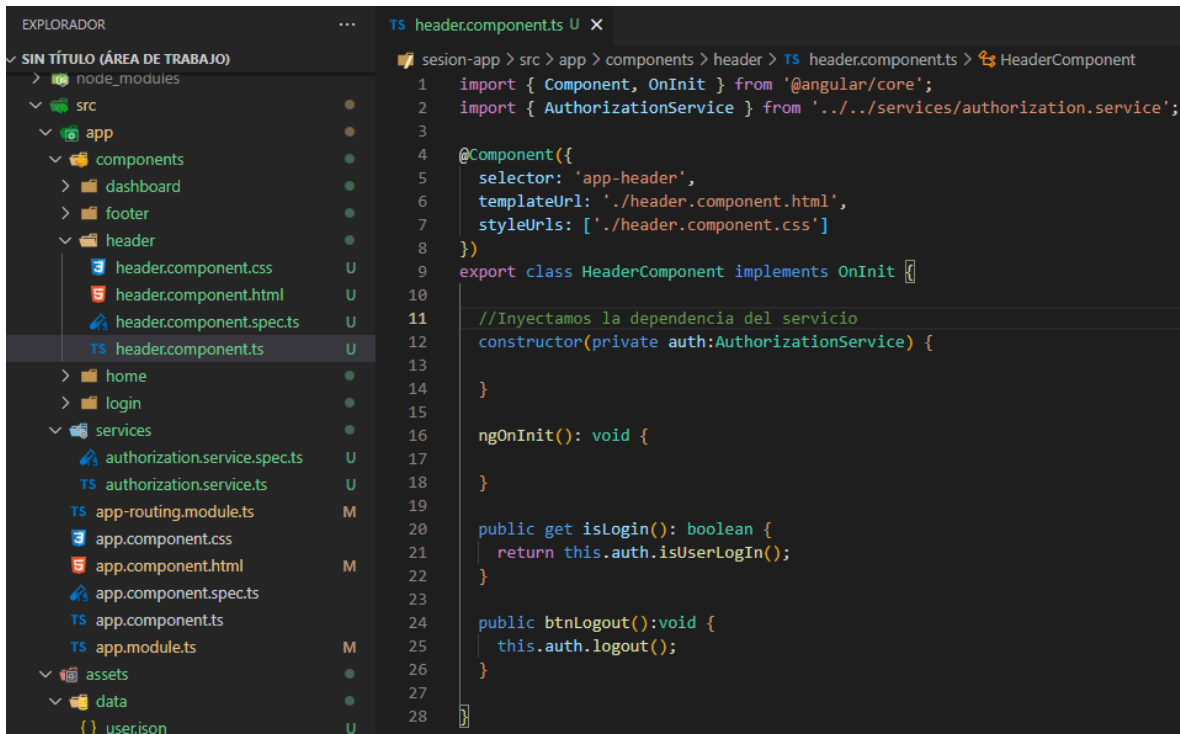


The screenshot shows an IDE with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with folders like `src`, `assets`, and `data`. The `src` folder is expanded, showing subfolders like `components`, `header`, `home`, `login`, and `services`. The `services` folder is selected, and the `authorization.service.ts` file is open in the editor. The code in the editor defines the `AuthorizationService` class, which is injectable and provides a root provider. It includes a constructor that injects `HttpClient` and `Router`, and sets the `strUrlApi` to `../../assets/data/user.json`. The `loginSimple` method sends a POST request to the API, checks for a token in the response, and if found, stores it in `localStorage` and navigates to the dashboard. The `logout` method removes the token from `localStorage` and navigates back to the root. The `isUserLogin` method checks if a token is present in `localStorage` and returns a boolean.

```
1 import { Injectable } from '@angular/core';
2 import { HttpClient } from '@angular/common/http';
3 import { Router } from '@angular/router';
4
5 @Injectable({
6   providedIn: 'root'
7 })
8 export class AuthorizationService {
9   //URL de nuestra api
10  private strUrlApi:string;
11
12  //Inyectar dependencias en el constructor
13  constructor(private http:HttpClient, private router:Router) {
14    //Ruta de nuestro archivo json(en nuestro caso es local)
15    this.strUrlApi = '../../assets/data/user.json';
16  }
17
18  public loginSimple(email:string, pwd: string): void {
19    //Llamada a la API
20    this.http.get(this.strUrlApi).subscribe(
21      (response: any) => {
22        if(response.token != null) {
23          //Guardamos el token
24          localStorage.setItem('token', response.token);
25          this.router.navigate(['/dashboard']);
26        }
27      }
28    );
29  }
30
31  public logout(): void {
32    //Al cerrar sesion eliminamos el token
33    localStorage.removeItem('token');
34    //Redireccionar
35    this.router.navigate(['/']);
36  }
37
38  public isUserLogin(): boolean {
39    return (localStorage.getItem('token') != null);
40  }
41
42 }
```

Nota: por ahora es un login simple, mas adelante en este mismo tutorial mejorare el servicio de login.

Ahora en nuestro en el controlador de nuestro header component:



The screenshot shows the Visual Studio Code interface. On the left, the 'EXPLORADOR' (File Explorer) pane displays the project structure. The 'src' folder is expanded, showing 'app' > 'components' > 'header'. The 'header' folder contains 'header.component.css', 'header.component.html', 'header.component.spec.ts', and 'header.component.ts'. The 'header.component.ts' file is selected. On the right, the editor shows the content of 'header.component.ts'. The code defines the 'HeaderComponent' class, which implements 'OnInit'. It imports 'Component' and 'OnInit' from '@angular/core' and 'AuthorizationService' from '../services/authorization.service'. The component has a selector of 'app-header', a templateUrl of './header.component.html', and styleUrls of ['./header.component.css']. The constructor injects 'auth: AuthorizationService'. The 'ngOnInit()' method is empty. There are two public methods: 'isLogin()' which returns 'this.auth.isUserLoggedIn()', and 'btnLogout()' which calls 'this.auth.logout()'.

```
1 import { Component, OnInit } from '@angular/core';
2 import { AuthorizationService } from '../services/authorization.service';
3
4 @Component({
5   selector: 'app-header',
6   templateUrl: './header.component.html',
7   styleUrls: ['./header.component.css']
8 })
9 export class HeaderComponent implements OnInit {
10
11   //Inyectamos la dependencia del servicio
12   constructor(private auth:AuthorizationService) {
13
14   }
15
16   ngOnInit(): void {
17
18   }
19
20   public get isLogin(): boolean {
21     return this.auth.isUserLoggedIn();
22   }
23
24   public btnLogout():void {
25     this.auth.logout();
26   }
27
28 }
```

Agregamos la propiedad `isLogin`, y el evento para el botón de cerrar sesión (Logout).

Finalmente, en la vista del componente header, agregamos el `*ngIf` para el enlace, entonces si `isLogin=true` se muestra el enlace a Dashboard, si no se oculta:

```
<a class="nav-link active" aria-current="page" routerLink="/dashboard"
*ngIf="isLogin">Dashboard</a>
```

Y en el form, agregamos también el evento click:

```
<form class="d-flex">
  <button class="btn btn-outline-success" type="button" *ngIf="!isLogin"
routerLink="/login">Login</button>
  <button class="btn btn-outline-success" type="button" *ngIf="isLogin"
(click)="btnLogout()">Logout</button>
</form>
```

Nos queda así:

```
header.component.html U X
session-app > src > app > components > header > header.component.html > header > nav.navbar.navbar-expand-lg.navbar-dark.bg-dark
Go to component
1 <header>
2   <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
3     <div class="container-fluid">
4       <a class="navbar-brand" href="/">Navbar</a>
5       <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent"
6         aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
7         <span class="navbar-toggler-icon"></span>
8       </button>
9       <div class="collapse navbar-collapse" id="navbarSupportedContent">
10        <ul class="navbar-nav me-auto mb-2 mb-lg-0">
11          <li class="nav-item">
12            <a class="nav-link active" aria-current="page" routerLink="/home">Home</a>
13          </li>
14
15          <li class="nav-item">
16            <a class="nav-link active" aria-current="page" routerLink="/dashboard" *ngIf="isLoggedIn">Dashboard</a>
17          </li>
18        </ul>
19        <form class="d-flex">
20          <button class="btn btn-outline-success" type="button" *ngIf="!isLoggedIn" routerLink="/login">Login</button>
21          <button class="btn btn-outline-success" type="button" *ngIf="isLoggedIn" (click)="btnLogout()">Logout</button>
22        </form>
23      </div>
24    </div>
25  </nav>
26
27
28 </header>
```

Para el formulario de login hacemos así, en la vista(aquí usamos en el input estos dos atributos para comunicarle los datos al controlador `name="loginTxtEmail"` `[(ngModel)]="loginTxtEmail"`):

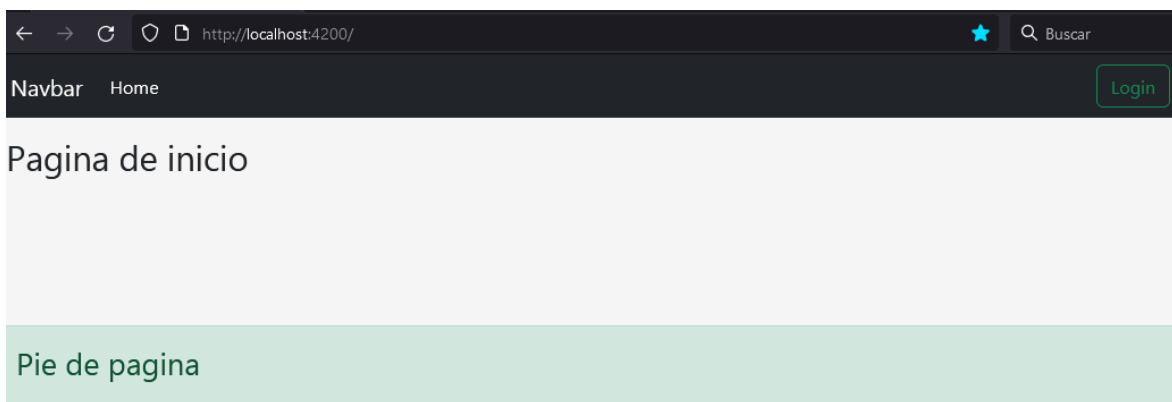
```
login.component.html U X
session-app > src > app > components > login > login.component.html > div.row.my-5 > div.col-sm-11.col-md-6.col-lg-4.mx-auto.my-auto.card > form
Go to component
1 <div class="row my-5">
2   <div class="col-sm-11 col-md-6 col-lg-4 mx-auto my-auto card">
3     <form method="post">
4       <!-- Email input -->
5       <div class="form-outline mb-4">
6         <label class="form-label" for="loginTxtEmail">Email</label>
7         <input type="email" id="loginTxtEmail" class="form-control" name="loginTxtEmail" [(ngModel)]="loginTxtEmail"/>
8       </div>
9
10      <!-- Password input -->
11      <div class="form-outline mb-4">
12        <label class="form-label" for="loginTxtPwd">Password</label>
13        <input type="password" id="loginTxtPwd" class="form-control" name="loginTxtPwd" [(ngModel)]="loginTxtPwd"/>
14      </div>
15
16      <!-- Submit button -->
17      <button type="submit" class="btn btn-primary btn-block mb-4 col-12
18        col-sm-12 col-md-12" (click)="btnLogin()">Iniciar sesion</button>
19    </form>
20  </div>
21 </div>
22 </div>
```

En el controlador de login:

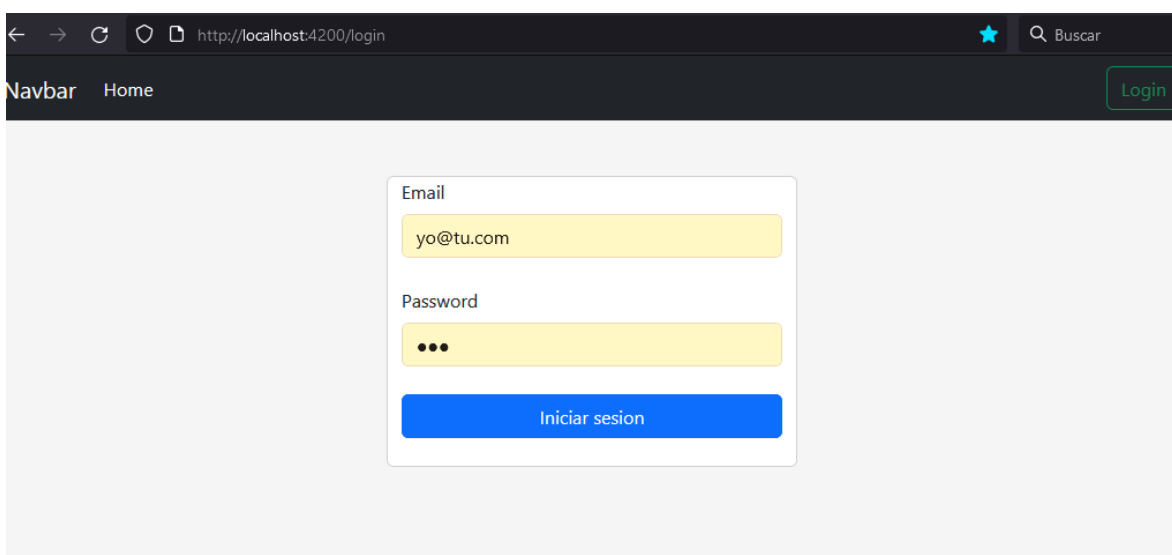
```
TS login.component.ts U X
sesion-app > src > app > components > login > TS login.component.ts > LoginComponent

4  @Component({
5    selector: 'app-login',
6    templateUrl: './login.component.html',
7    styleUrls: ['./login.component.css']
8  })
9  export class LoginComponent implements OnInit {
10
11    public loginTxtEmail:string;
12    public loginTxtPwd:string;
13
14    constructor(private auth:AuthorizationService) {}
15    //Inicializar atributos
16    this.loginTxtEmail = "";
17    this.loginTxtPwd = "";
18  }
19
20  ngOnInit(): void {
21
22  }
23
24  public btnLogin(): void {
25    //Consultamos la API
26    this.auth.loginSimple(this.loginTxtEmail, this.loginTxtPwd);
27  }
28
29 }
```

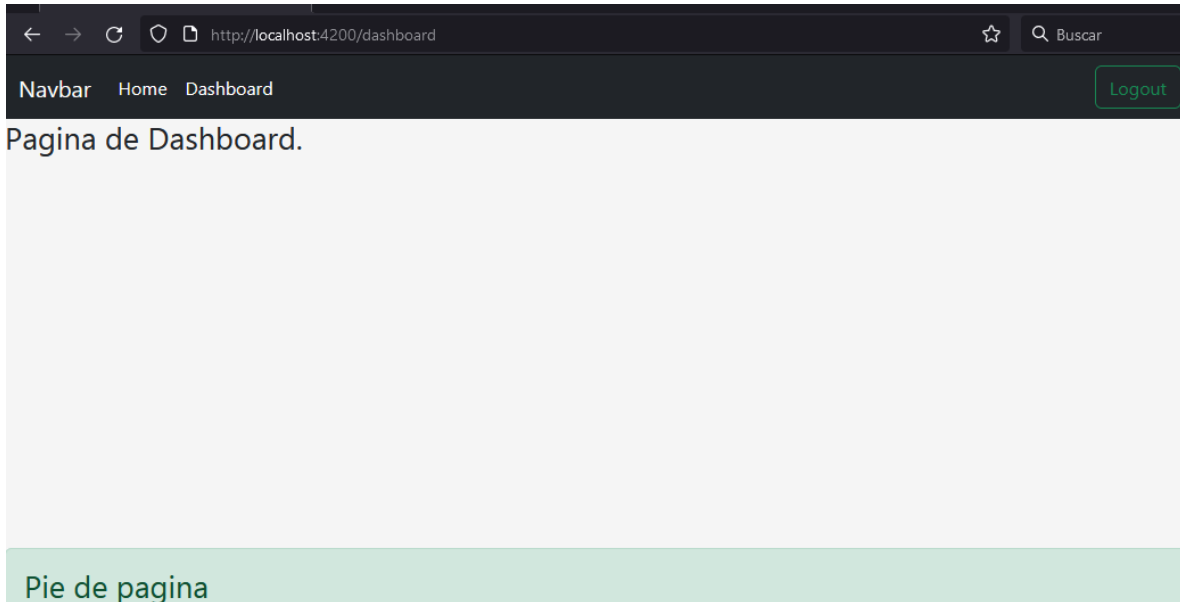
Entonces al inicio se verá así:



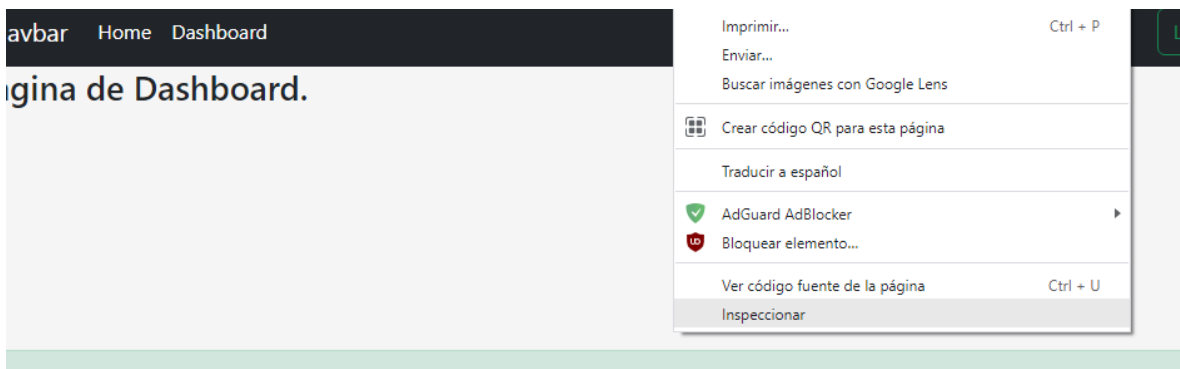
Al hacer click en el botón de login será vera así:



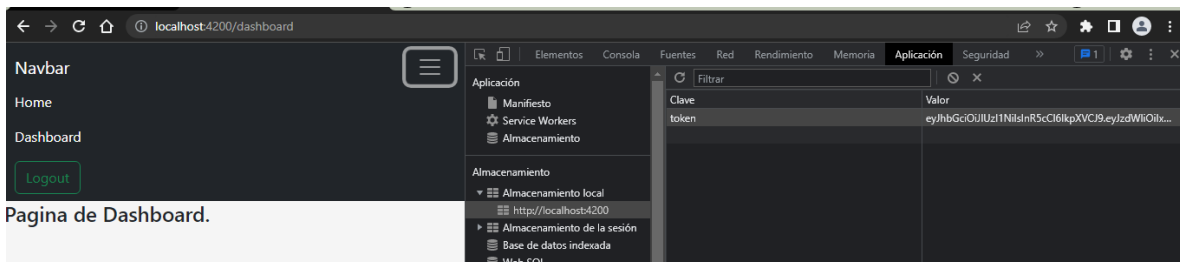
Al iniciar sesión:



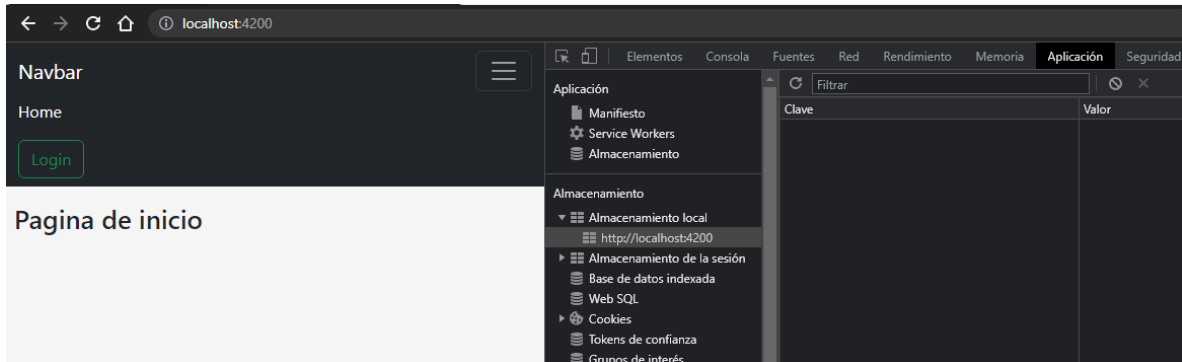
Aquí ya se nos creo el token de autorización, lo podemos ver con el inspector, en Chrome:



Y luego en Aplicación, allí está el token de autorización:



Al hacer click en Logout se borra el token y todo vuelve como estaba al cargar la página por primera vez:



Ultimo punto a tener cuenta es [proteger las rutas privadas](#) para que un usuario no ingrese por ejemplo a Dashboard ingresando directamente la ruta en la barra de direcciones.

Esto se hace utilizando Angular Route Guard que es una interfaz que puede ser implementada para decidir si una ruta puede ser activada.

Hay 5 tipos de guards(guardias) en Angular: CanActivate, CanActivateChild, CanDeactivate, Resolve y CanLoad. Veamos el guardia CanActivate, que es uno de los guardias más utilizados y que nos permitirá entender mejor cómo utilizar los guardias en Angular.

Entonces ahora creamos una guardia:

```
PS C:\sesion-app> ng g guard services/authorization
? Which interfaces would you like to implement? (Press <space> to select, <a> to toggle all, <i> to invert selection, and <enter> to proceed)
>(*) CanActivate
  ( ) CanActivateChild
  ( ) CanDeactivate
  ( ) CanLoad
```

Le damos enter:

```
PS C:\sesion-app> ng g guard services/authorization
? Which interfaces would you like to implement? CanActivate
CREATE src/app/services/authorization.guard.spec.ts (376 bytes)
CREATE src/app/services/authorization.guard.ts (466 bytes)
PS C:\sesion-app>
```

Ahora dentro de la guardia creada, y dentro de canActivate:

```
authorization.service.ts U    TS authorization.guard.ts U X
session-app > src > app > services > TS authorization.guard.ts > AuthorizationGuard > canActivate
import { Injectable } from '@angular/core';
import { ActivatedRouteSnapshot, CanActivate, RouterStateSnapshot, UrlTree } from '@angular/router';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class AuthorizationGuard implements CanActivate {
  canActivate(
    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot): Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree {

    return true;
  }
}
```

Agregamos las importaciones para las dependencias y preguntamos si el usuario este logueado:

```
TS authorization.service.ts U    TS authorization.guard.ts U X
session-app > src > app > services > TS authorization.guard.ts > AuthorizationGuard > canActivate
1 import { Injectable } from '@angular/core';
2 import { ActivatedRouteSnapshot, CanActivate, RouterStateSnapshot, UrlTree, Router } from '@angular/router';
3 import { Observable } from 'rxjs';
4 import { AuthorizationService } from '../authorization.service';
5
6 @Injectable({
7   providedIn: 'root'
8 })
9 export class AuthorizationGuard implements CanActivate {
10
11   //Inyectamos las dependencias necesarias
12   constructor(private authService: AuthorizationService, private router: Router) {
13
14   }
15
16   canActivate(
17     route: ActivatedRouteSnapshot,
18     state: RouterStateSnapshot): Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree {
19
20     //Preguntamos si el usuario esta logueado
21     if (!this.authService.isUserLoggedIn()) {
22       //Si no esta logueado, ir a la pagina de login
23       this.router.navigate(['/login']);
24       return false;
25     }
26
27     return true;
28   }
}
```

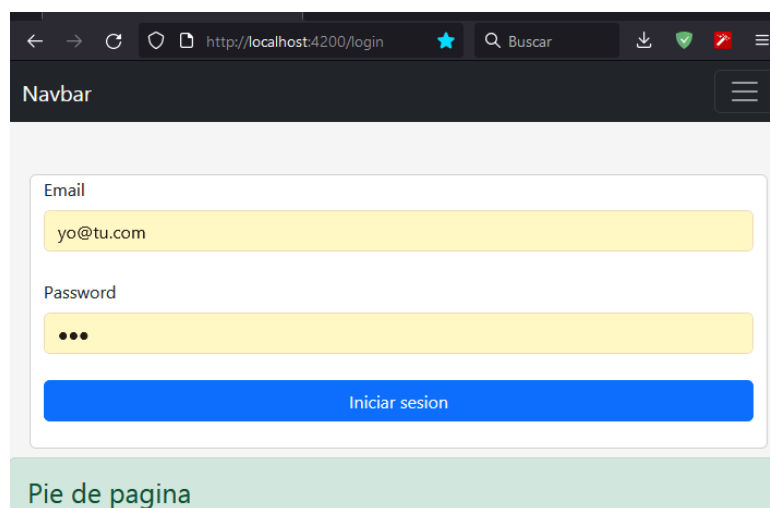
Finalmente vamos al `app-routing.module.ts` y agregamos el `canActivate` a la ruta que queremos proteger:

```

TS authorization.service.ts U    TS authorization.guard.ts U    TS app-routing.module.ts M X
sesion-app > src > app > TS app-routing.module.ts > routes
You, hace 1 minuto | 1 author (You)
1  import { NgModule } from '@angular/core';
2  import { RouterModule, Routes } from '@angular/router';
3  import { HomeComponent } from './components/home/home.component';
4  import { LoginComponent } from './components/login/login.component';
5  import { DashboardComponent } from './components/dashboard/dashboard.component';
6
7  //Primero importar la guardia
8  import { AuthorizationGuard } from './services/authorization.guard';
9
10 const routes: Routes = [
11   { path: '', component: HomeComponent },
12   { path: 'home', component: HomeComponent },
13   { path: 'login', component: LoginComponent },
14
15   //Luego proteger la ruta
16   { path: 'dashboard', component: DashboardComponent, canActivate: [AuthorizationGuard] },
17 ];
18
19 @NgModule({
20   imports: [RouterModule.forRoot(routes)],
21   exports: [RouterModule]
22 })
23 export class AppRoutingModule { }
    
```

Ahora podemos para probar la guardia, si al estar en inicio y sin estar logueado, escribimos en la barra de direcciones <http://localhost:4200/dashboard>

Se redijera la página hacia la página de Login.



Segunda parte – Mejoramiento del servicio de login

Ahora agregare el envío del email y la contraseña a una API Rest, para su correspondiente validación y en caso de ser correctos, la API nos devolverá un token de autorización:

```
TS authorization.service.ts U X
session-app > src > app > services > TS authorization.service.ts > AuthorizationService > log

10 private strUrlApi:string;
11 private strRemoteUrlApi:string;
12
13 //Inyectar dependencias en el constructor
14 constructor(private http:HttpClient, private router:Router) {
15   //Ruta de nuestro archivo json(en nuestro caso es local)
16   this.strUrlApi = '../assets/data/user.json';
17
18   //Ruta a nuestra API
19   this.strRemoteUrlApi = 'http://localhost:9000/api/authenticate';
20 }
21
22 /**
23  * Login real para conectarse a una API local o remota
24  * @param email
25  * @param pwd
26  */
27 public login(email:string, pwd: string): void {
28   //Cuerpo del metodo POST con los parametros del email y el pwd
29   const user = {
30     email: email,
31     password: pwd
32   };
33
34   //Convertir un objeto de javascript a JSON
35   const body = JSON.stringify(user);
36
37   //Llamada a la API
38   this.http.post(this.strRemoteUrlApi, body).subscribe(
39     (response: any) => {
40       if(response.token != null) {
41         //Guardamos el token
42         localStorage.setItem('token', response.token);
43         //Redireccionar
44         this.router.navigate(['/dashboard']);
45       } else {
46         //Mensaje solo para propositos de depuracion
47         console.log("Usuario y/o contraseña no valida.")
48       }
49     }
50   );
51 }
52
53 /**
54  * Login de ejemplo, se puede utilizar mientras no se haya
55  * construido la API Rest en el Back end
56  * @param email
57  * @param pwd
58  */
59 public loginSimple(email:string, pwd: string): void {
60   //Llamada a la API
```

Tercera parte (Nivel avanzado) – Encriptación del token y demás datos de nuestra app

Como vimos anteriormente, cualquier persona que tenga conocimientos de programación web, puede inspeccionar nuestra web con las herramientas de cualquier navegador actual.

Por lo que podría copiar el token de autorización y acceder a los datos de la API Rest, por ello una buena practica es encriptar los datos guardados en el localStorage, usando algún algoritmo de encriptación.

En este ejemplo se usará la librería crypto-js, que se encuentra en

- <https://www.npmjs.com/package/crypto-js>

luego usamos los siguientes comandos para descargarla:

```
npm install crypto-js
```

```
npm i --save-dev @types/crypto-js
```

después de instalar esas dependencias crearemos un nuevo servicio:

```
ng g service services/localStorage
```

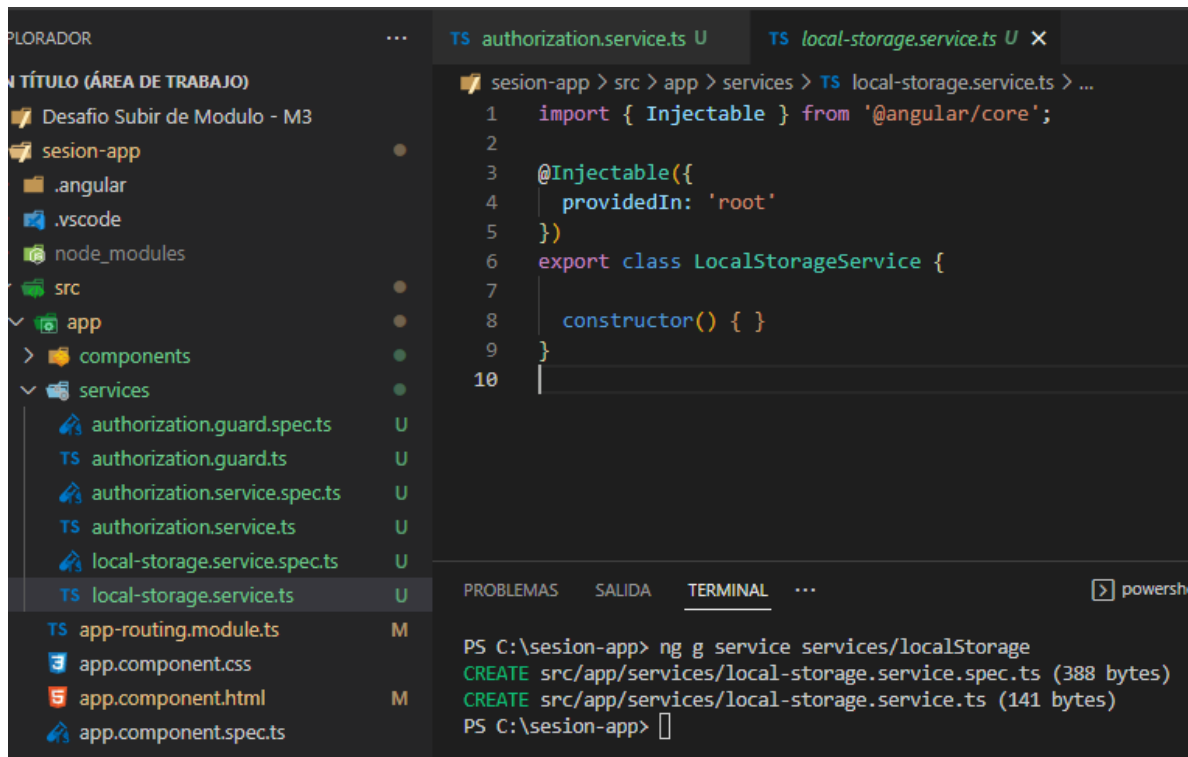
Nota: para evitar

WARNING CommonJS or AMD dependencies can cause optimization

Agregar en angular.json:

```
"allowedCommonJsDependencies": ["crypto-js"],
```

```
    "prefix": "app",
    "architect": {
      "build": {
        "builder": "@angular-devkit/build-angular:browser",
        "options": {
          "allowedCommonJsDependencies": ["crypto-js"],
          "outputPath": "dist/sesion-app",
          "index": "src/index.html",
          "main": "src/main.ts",
```



The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays the project structure for 'sesion-app', including folders like '.angular', '.vscode', 'node_modules', 'src', and 'app'. The 'app' folder is expanded, showing 'components' and 'services'. The 'services' folder is selected, and a list of files is shown, including 'authorization.guard.spec.ts', 'authorization.guard.ts', 'authorization.service.spec.ts', 'authorization.service.ts', 'local-storage.service.spec.ts', 'local-storage.service.ts', 'app-routing.module.ts', 'app.component.css', 'app.component.html', and 'app.component.spec.ts'. The 'local-storage.service.ts' file is highlighted. The main editor area shows the content of 'local-storage.service.ts', which includes an import statement for 'Injectable' from '@angular/core', an '@Injectable' decorator with 'providedIn: 'root'', and an 'export class LocalStorageService' with an empty constructor. The bottom panel shows the 'TERMINAL' tab with the command 'PS C:\sesion-app> ng g service services/localStorage' and the output 'CREATE src/app/services/local-storage.service.spec.ts (388 bytes)' and 'CREATE src/app/services/local-storage.service.ts (141 bytes)'. The terminal prompt is 'PS C:\sesion-app> '.

```
session-app > src > app > services > TS local-storage.service.ts > ...
1  import { Injectable } from '@angular/core';
2
3  @Injectable({
4    providedIn: 'root'
5  })
6  export class LocalStorageService {
7
8    constructor() { }
9  }
10
```

PROBLEMAS SALIDA TERMINAL ... powershell

PS C:\sesion-app> ng g service services/localStorage
CREATE src/app/services/local-storage.service.spec.ts (388 bytes)
CREATE src/app/services/local-storage.service.ts (141 bytes)
PS C:\sesion-app>

Y crearemos cuatros métodos en este servicio:

```
TS authorization.service.ts U TS local-storage.service.ts U X
session-app > src > app > services > TS local-storage.service.ts > LocalStorageService > getData
1 import { Injectable } from '@angular/core';
2
3 @Injectable({
4   providedIn: 'root'
5 })
6 export class LocalStorageService {
7
8   constructor() {
9
10  }
11
12  //Metodo para guardar un dato en localStorage
13  public saveData(key: string, value: string): void {
14    window.localStorage.setItem(key, value);
15  }
16
17  //Metodo para recuperar un dato en localStorage
18  public getData(key: string): any {
19    return window.localStorage.getItem(key);
20  }
21
22  //Metodo para eliminar un dato en localStorage
23  public removeData(key: string): void {
24    window.localStorage.removeItem(key);
25  }
26
27  //Metodo para eliminar todos los datos en localStorage
28  public clearData(): void {
29    window.localStorage.clear();
30  }
31
32 }
```

Ahora importamos CryptoJS:

```
import * as CryptoJS from 'crypto-js';
```

Agregamos nuestra clave secreta de ejemplo (generar una nueva cuando se use en un proyecto nuevo):

```
$ authorization.service.ts U TS local-storage.service.ts U X
sesion-app > src > app > services > TS local-storage.service.ts > ...
1  import { Injectable } from '@angular/core';
2  import * as CryptoJS from 'crypto-js';
3
4  @Injectable({
5    providedIn: 'root'
6  })
7  export class LocalStorageService {
8
9    private secretKey: string;
10
11    constructor() {
12      //Clave generada en https://generate-random.org/
13      this.secretKey = "5bnU8ssMga@BK6b2-5f4-BxyMuq@UKRzC";
14    }
15
16    //Metodo que permite encriptar una cadena de texto
17    private encrypt(txt: string): string {
18      return CryptoJS.AES.encrypt(txt, this.secretKey).toString();
19    }
20
21    //Metodo que permite desencriptar una cadena de texto
22    private decrypt(txtToDecrypt: string) {
23      return CryptoJS.AES.decrypt(txtToDecrypt, this.secretKey).toString(CryptoJS.enc.Utf8);
24    }
25
26    //Metodo para guardar un dato en localStorage
27    public saveData(key: string, value: string): void {
28      window.localStorage.setItem(key, value);
29    }
30  }
```

Y agregamos los métodos encrypt y decrypt.

Ahora haremos una mejora de los métodos de `saveData()` y `getData()`, agregándole la encriptación:

```
TS authorization.service.ts U TS local-storage.service.ts U X
sesion-app > src > app > services > TS local-storage.service.ts > LocalStorageService
14 }
15
16 //Metodo que permite encriptar una cadena de texto
17 private encrypt(txt: string): string {
18   return CryptoJS.AES.encrypt(txt, this.secretKey).toString();
19 }
20
21 //Metodo que permite desencriptar una cadena de texto
22 private decrypt(txtToDecrypt: string) {
23   return CryptoJS.AES.decrypt(txtToDecrypt, this.secretKey).toString(CryptoJS.enc.Utf8);
24 }
25
26 //Metodo para guardar un dato en localStorage
27 public saveData(key: string, value: string): void {
28   window.localStorage.setItem(key, this.encrypt(value));
29 }
30
31 //Metodo para recuperar un dato en localStorage
32 public getData(key: string): any {
33   let data = window.localStorage.getItem(key) || "";
34   return this.decrypt(data);
35 }
```

Luego en el `AuthorizationService`, importamos e inyectamos la dependencia:

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Router } from '@angular/router';
//Importar
import { LocalStorageService } from '../local-storage.service';

@Injectable({
  providedIn: 'root'
})
export class AuthorizationService {
  //URL de nuestra api
  private strUrlApi:string;
  private strRemoteUrlApi:string;

  //Inyectar dependencias en el constructor
  constructor(private http:HttpClient, private router:Router, private localStorage: LocalStorageService) {
    //Ruta de nuestro archivo json(en nuestro caso es local)
  }
}
```

Después reemplazamos `localStorage.setItem` por `this.localStorage.saveData`:

```
password: pwd
};

//Convertir un objeto de javascript a JSON
const body = JSON.stringify(user);

//Llamada a la API
this.http.post(this.strRemoteUrlApi, body).subscribe(
  (response: any) => {
    if(response.token != null) {
      //Guardamos el token
      //localStorage.setItem('token', response.token);
      this.localStorage.saveData('token', response.token);

      //Redireccionar
      this.router.navigate(['/dashboard']);
    } else {
      //Mensaje solo para propósitos de depuración
      console.log("Usuario y/o contraseña no válida.");
    }
  }
);
}
```

Lo mismos aquí:

```
/**
 * Login de ejemplo, se puede utilizar mientras no se haya
 * construido la API Rest en el Back end
 * @param email
 * @param pwd
 */
public loginSimple(email:string, pwd: string): void {
  //Llamada a la API
  this.http.get(this.strUrlApi).subscribe(
    (response: any) => {
      if(response.token != null) {
        //Guardamos el token
        //localStorage.setItem('token', response.token);
        this.localStorage.saveData('token', response.token);
        //Redireccionar
        this.router.navigate(['/dashboard']);
      }
    }
  );
}
```

Finalmente, también, se podría encriptar la palabra token, y también se podría desencriptar el token y extraer los datos del usuario.

En fin, aquí termina este mini tutorial para no hacerlo más largo.

En el componente Home hay un ejemplo de como quedan los textos encriptados:

```
session-app > src > app > components > home > TS home.component.ts > HomeComponent
6   templateUrl: './home.component.html',
7   styleUrls: ['./home.component.css']
8 })
9 export class HomeComponent implements OnInit {
10
11   public textOriginal: string;
12   public textEncrypt: string;
13   public textDecrypt: string;
14
15   constructor(private localStorage: LocalStorageService) {}
16
17   this.textOriginal = "#ArgentinaPrograma";
18   this.textEncrypt = "";
19   this.textDecrypt = "";
20
21
22   ngOnInit(): void {
23     this.textEncrypt = this.localStorage.encrypt(this.textOriginal);
24     this.textDecrypt = this.localStorage.decrypt(this.textEncrypt);
25   }
26
27 }
28
```



```
authorization.service.ts U    TS local-storage.service.ts U    TS home.component.ts U
session-app > src > app > components > home > home.component.html > div
Go to component
<div class="ms-2 me-2 mt-3 mb-3">

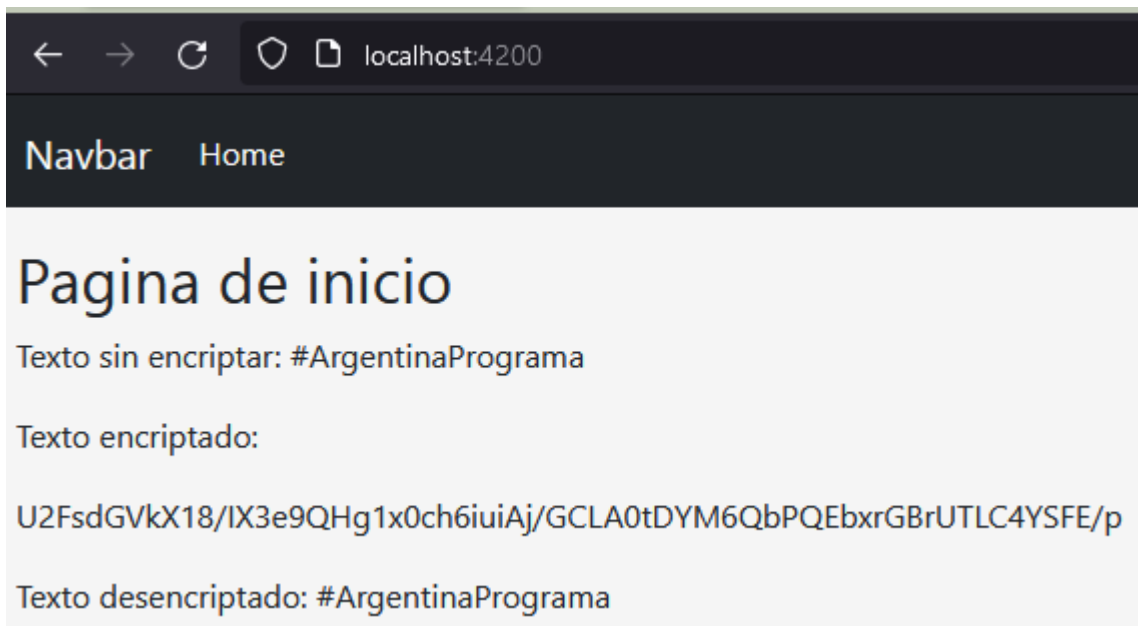
  <h2>Pagina de inicio</h2>

  <p> Texto sin encriptar: {{ textOriginal }} </p>

  <p> Texto encriptado: </p>
  <p class="txt-encrypt"> {{ textEncrypt }} </p>

  <p> Texto desencriptado: {{ textDecrypt }} </p>

</div>
```



- Fin -