

PRÁCTICA 1.

La finalidad de esta práctica es introducirse en la programación de dispositivos móviles (smartphones, tablets, etc.), para la obtención de aplicaciones multiplataforma. En concreto, se trabajará sobre el framework de desarrollo de aplicaciones multiplataforma Ionic (<https://ionicframework.com>). Este framework como veremos, nos permitirá desarrollar una misma aplicación y que esta puede ser utilizada en las diferentes plataformas disponibles en el mercado (Android, iOS).

Se utilizarán el entorno de desarrollo Visual Code, así como las diferentes herramientas que nos facilitarán el trabajo de compilación, depuración, pruebas y despliegue, mediante el framework Ionic.

Comenzaremos desarrollando una serie de ejemplos guiados, los cuales nos introducirán en el desarrollo de aplicaciones multiplataforma, mediante IONIC. Una vez realizados estos ejercicios guiados, se planteará un ejercicio a partir del cual se calificará esta práctica.

Las diferentes herramientas a emplear podemos encontrarlas para los sistemas operativos más habituales (Windows, Linux, macOS). Los ejercicios guiados que se detallan a continuación se basan en una instalación bajo Windows 10/11 64 bits.

Introducción a Ionic Framework

Dentro de los frameworks multiplataforma nos encontramos con diversos frameworks, siendo **Ionic** uno de los más populares y con mayor comunidad.

Para desarrollar con Ionic vamos a necesitar una serie de herramientas. Lo primero será un editor de código, **Visual Studio Code** es una herramienta ligera y potente, en la cual podemos añadir extensiones para facilitar el trabajo.

Ionic es una herramienta en la que se hace un uso intensivo de comandos de consola y por ello haremos uso de la línea de comandos que ofrece Visual Studio Code, evitando alternar tareas constantemente para acceder a la ventana de comandos. Además, nos permite abrir varias instancias de la misma que, como veremos más adelante, será fundamental.

También necesitaremos los **SDK** de aquellas plataformas para las que vayamos a desarrollar. Si desarrollamos para **Android** necesitaremos instalar su SDK, siendo la forma más sencilla instalar Android Studio.

Ionic internamente utiliza **Cordova**, como librería intermedia de código abierto que nos permite hacer aplicaciones móviles basadas en HTML5, CSS3 y Javascript. Por lo tanto será necesario tener un conocimiento básico en esos tres lenguajes. También podemos emplear **Capacitor** el nuevo puente nativo para crear aplicaciones nativas.

Entre el lenguaje propio de Ionic y Cordova existe una capa intermedia. Esta capa intermedia estará formada por un framework que actualmente puede ser Angular, React o Vue. En nuestro caso optaremos por **Angular**, un framework de desarrollo web creado por Google y basado en Javascript que permite el desarrollo de aplicaciones web mediante MVC, muy resumidamente nos permite que si en nuestro modelo tenemos una propiedad y la mostramos en la vista mediante una directiva **ng-model**, todos los cambios que hagamos en nuestro controlador sobre esa propiedad se reflejarán inmediatamente en la vista y todos los cambios que hagamos en la vista (por ejemplo en una casilla de entrada de texto) se reflejarán en la propiedad del modelo.

Para comenzar con Ionic, no necesitaremos obligatoriamente conocer Angular, para desarrollos más avanzados si es interesante estar familiarizado.

Tanto Angular como Ionic emplean **Typescript** un superconjunto de javascript, el cual incluye todas las funcionalidades de JS más las funcionalidades de TS. Es un lenguaje estricto en la forma de declarar y utilizar sus variables con el objetivo de crear un código más inteligible.

En la web de typescript podemos encontrar toda la documentación:

<https://www.typescriptlang.org>

Interesante este tutorial para los programadores que vienen de JS:

<https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html>

Y la herramienta para convertir de TypeScript a Javascript:

<https://www.typescriptlang.org/play>

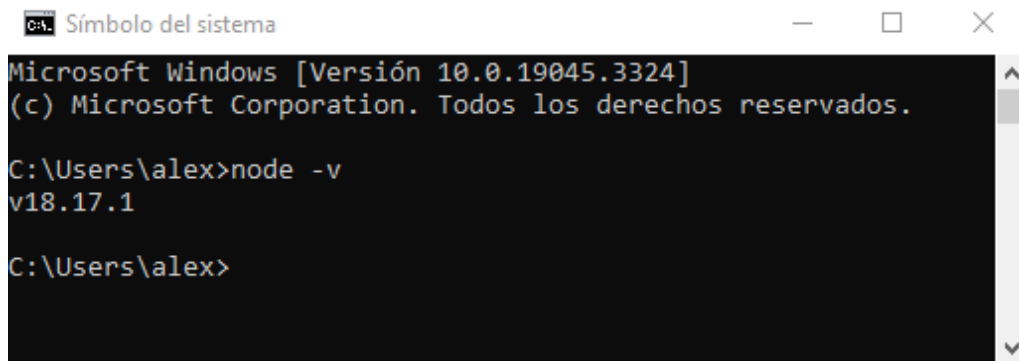
Ejercicio 0. Instalación software necesario

Para poder desarrollar aplicaciones mediante IONIC, necesitaremos el siguiente software:

1. **nodejs LTS** , el cual descargaremos de la siguiente URL:

<https://nodejs.org/es/>

Una vez descargado procederemos a la instalación y finalmente para comprobar el correcto funcionamiento, abriremos una Ventana de comandos (**CMD**) y escribiremos `node -v`:



```
Símbolo del sistema
Microsoft Windows [Versión 10.0.19045.3324]
(c) Microsoft Corporation. Todos los derechos reservados.

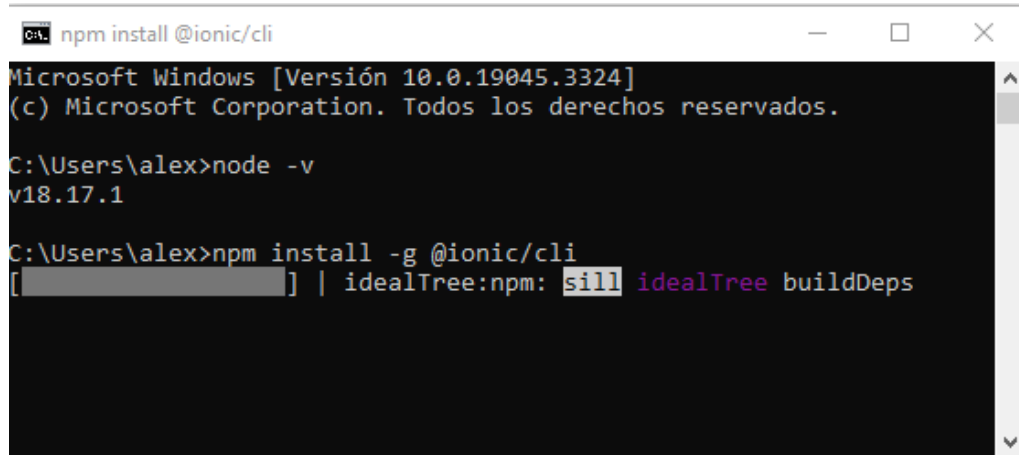
C:\Users\alex>node -v
v18.17.1

C:\Users\alex>
```

2. IONIC, <https://ionicframework.com/>

Para la instalación, emplearemos el gestor NPM (instalado previamente con nodejs). Abriremos una Ventana de comandos (CMD) y escribiremos:

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



```
npm install @ionic/cli
Microsoft Windows [Versión 10.0.19045.3324]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\alex>node -v
v18.17.1

C:\Users\alex>npm install -g @ionic/cli
[redacted] | idealTree:npm: sill idealTree buildDeps
```

El proceso detallado Podemos consultar en la web de ionic:

<https://ionicframework.com/docs/intro/cli>

Una vez instalado IONIC, Podemos comprobar la instalación, con la creación de un Proyecto de ejemplo:

Desde Ventana de comandos, escribimos:

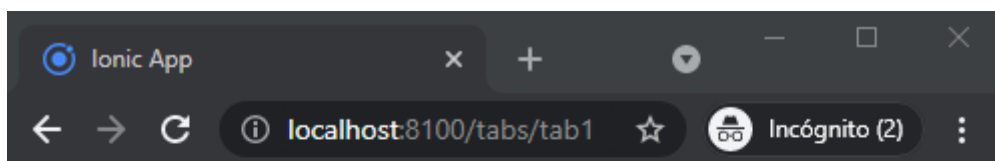
```
ionic start DISM0
```

Esto nos despliega un asistente, en el cual marcamos las opciones (tabs, angular, ngModules), finalmente, tras descargar los paquetes necesarios e indicar que NO queremos crear una cuenta de Ionic, se nos ha generado el proyecto. Para ejecutar la aplicación:

```
cd .\DISM0
```

```
ionic serve
```

Automáticamente se desplegará el navegador con la URL <http://localhost:8100>, con el proyecto creado por defecto:



Tab 1

Tab 1 page
Explore UI Components

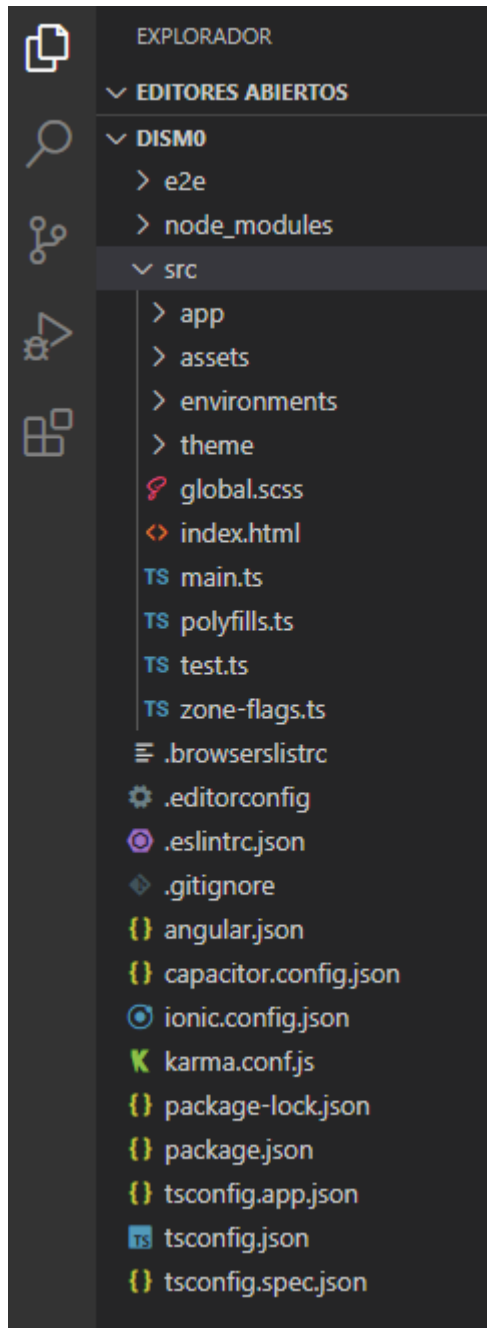
Tab 1

Tab 2

Tab 3

Estructura del proyecto

Al crear un proyecto con ionic se crea una carpeta con el nombre del proyecto y dentro de ella una estructura de archivos y directorios que contiene todos los elementos del proyecto.



e2e: Aquí se encuentra el código para escribir tests.

node_modules: Se genera automáticamente al instalar las dependencias npm con “npm install”. Este comando explora el archivo package.json para todos los paquetes que necesitan ser instalados.

src: Esta es la carpeta más importante y donde realizaremos la mayor parte de nuestro trabajo. Aquí es donde están los archivos con el contenido de nuestra aplicación, donde definimos las pantallas, el estilo y el comportamiento que tendrá nuestra aplicación.

Dentro de **src** tenemos las siguientes subcarpetas:

La carpeta **app**, que es donde se ubicarán las páginas que creemos para la aplicación, los servicios y toda la lógica de programación.

La carpeta **assets** donde almacenaremos aquellos recursos que necesitemos para nuestra aplicación, como imágenes etc.

La carpeta **environments** contiene archivos donde podemos definir variables de entorno.

La carpeta **theme** contiene el archivo variables.scss donde se definen las variables css de ionic. Por defecto vienen definidos los colores para los temas, podemos personalizarlos.

El archivo **global.scss**: En este archivo podemos establecer css para utilizar globalmente en cualquier lugar de nuestra aplicación.

index.html: Es el punto de entrada de nuestra aplicación, por regla general no tocaremos nada.

main.ts: Es el punto de entrada del módulo principal de nuestra aplicación. No necesitaremos cambiar nada aquí.

Karma.js y test.ts: Se utilizan para realizar tests unitarios.

tsconfig.app.json, tsconfig.spec.json: Contienen configuraciones typescript para los tests.

.editorconfig y .gitignore son dos archivos ocultos, así que dependiendo de tu sistema puede que no los veas, están relacionados con la configuración del editor de código y Git

angular.json: Archivo de configuración de la app.

ionic.config.json: Contiene información básica sobre la configuración nuestro proyecto.

package.json: Contiene paquetes y dependencias de nodeJS.

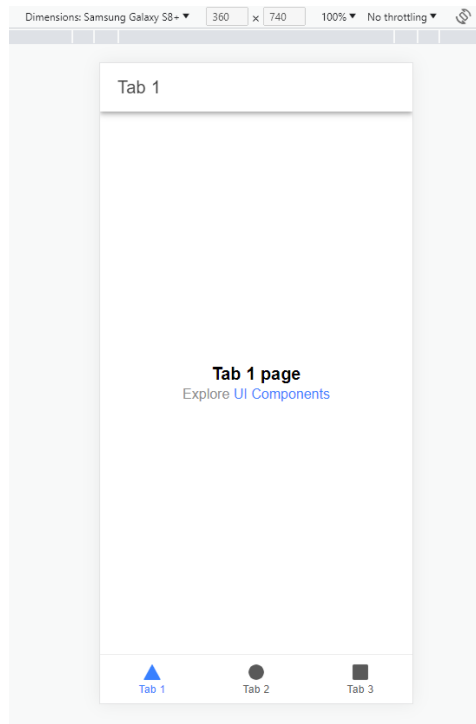
tsconfig.json: Son archivos que contienen información necesaria a la hora de compilar TypeScript, no necesitamos editar estos archivos.

Depuración

Una opción muy interesante para la depuración es ejecutar la aplicación:

```
ionic serve
```

Y al emplear el navegador Chrome, (Botón derecho ratón-Inspeccionar Ctrl+May+I):



3. **Visual Code**, emplearemos este IDE para desarrollar las aplicaciones de Ionic. Lo descargaremos de:

<https://code.visualstudio.com/>

y procederemos a su instalación.

Las siguientes herramientas serán necesarias solo si se desea realizar el despliegue de la aplicación y distribución en Android:

4. **Android Studio**, será necesario para poder implementar la aplicación y distribuirla.
<https://developer.android.com/studio>
5. **JDK mínimo versión 8**, Java Developer Kit, imprescindible para poder trabajar con Android Studio.
<https://www.oracle.com/es/java/technologies/downloads/>

Ionic CLI

Ionic CLI es el intérprete por línea de comandos de Ionic, contiene una serie de herramientas muy útiles que nos permitirán realizar de una forma sencilla diversas tareas habituales en el desarrollo y producción de aplicaciones.

Toda la documentación y usos de esta herramienta:

<https://ionicframework.com/docs/cli>

Conocer la versión instalada de ionic:

```
ionic --version
```

Crear un nuevo proyecto:

<https://ionicframework.com/docs/cli/commands/start>

Ejecutar un proyecto creado:

<https://ionicframework.com/docs/cli/commands/serve>

Uno de los usos más habituales es la generación de páginas, componentes, servicios:

<https://ionicframework.com/docs/cli/commands/generate>

Construir aplicaciones para despliegue

<https://ionicframework.com/docs/cli/commands/build>

Generar SSL Key y certificados:

<https://ionicframework.com/docs/cli/commands/ssl-generate>

Ejercicio 1. Desarrollo Aplicación TABS

A continuación, vamos a desarrollar una aplicación básica sobre ionic, mediante la plantilla tabs y veremos la estructura que presenta esta.

Desde Ventana de comandos, escribimos

```
ionic start
```

Esto nos despliega un asistente, en el cual seleccionaremos:

Angular como framework

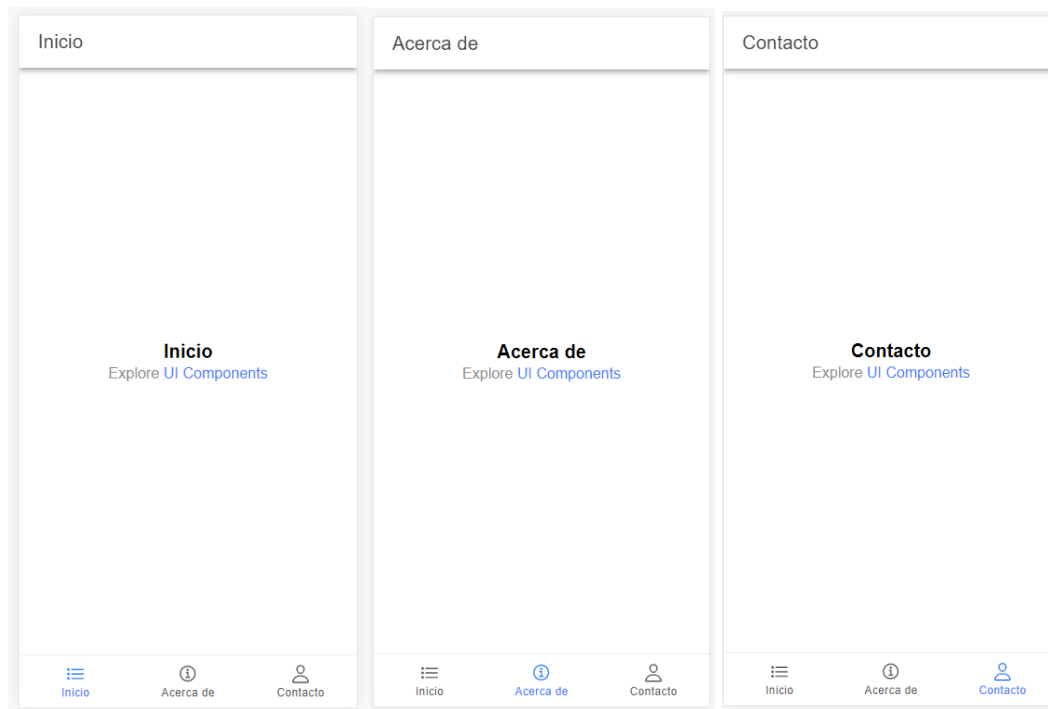
Nombre del proyecto **DISM1**

Y Plantilla **tabs**

Abrimos con Visual Code y nos vamos **tab1.page.html** y dejamos la estructura de esta forma:

```
src > app > tab1 > <> tab1.page.html > ...
1  <ion-header [translucent]="true">
2    <ion-toolbar>
3      <ion-title>
4        Inicio
5      </ion-title>
6    </ion-toolbar>
7  </ion-header>
8
9  <ion-content [fullscreen]="true">
10    <ion-header collapse="condense">
11      <ion-toolbar>
12        <ion-title size="large">Tab Inicio</ion-title>
13      </ion-toolbar>
14    </ion-header>
15    <app-explore-container name="Inicio"></app-explore-container>
16  </ion-content>
17
```

Modificamos tab2 y tab3 de forma similar para que queden como en la imagen:



El listado de iconos disponibles lo podéis consultar en la documentación oficial de ionic desde el siguiente enlace:

<https://ionicframework.com/docs/ionicons>

Ahora vamos a ver como añadir una nueva página para ello haremos uso del comando:

```
ionic generate page
```

Si estamos en visual studio code, pulsando Ctrl+May+Ñ , podemos abrir un terminal e introducir el comando para generar en esta.

```
PROBLEMS  OUTPUT  TERMINAL  CONSOLA DE DEPURACIÓN

Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscore6

PS D:\ionic\DISM0> ionic generate page
```

Indicamos como nombre del nuevo tab: **formulario**

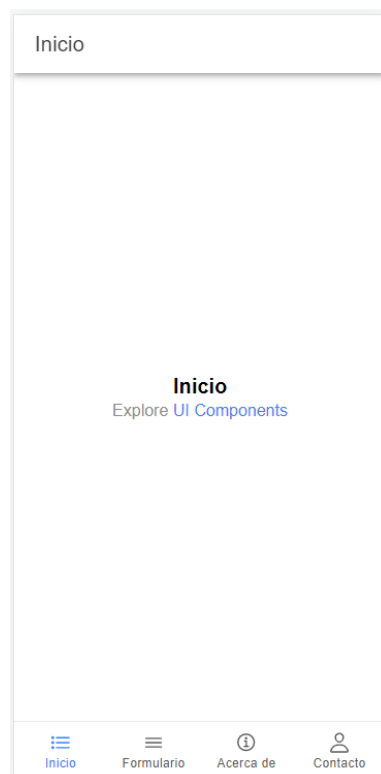
Si nos fijamos, en la estructura del proyecto en `src\app\` nos ha creado una nueva carpeta con el nombre formulario y con todo lo necesario. Ahora nos toca indicarle a **tabs-routing.module.ts** y **tabs.page.html** , la existencia de esa nueva página para que nos la muestre:

```
<ion-tab-button tab="tab3">
  <ion-icon name="body-outline"></ion-icon>
  <ion-label>Contacto</ion-label>
</ion-tab-button>

<ion-tab-button tab="formulario">
  <ion-icon name="layers-outline"></ion-icon>
  <ion-label>Formulario</ion-label>
</ion-tab-button>
```

```
path: 'tab3',
loadChildren: () => import('../tab3/tab3.module').then(m => m.Tab3PageModule)
},
{
  path: 'formulario',
loadChildren: () => import('../formulario/formulario.module').then(m => m.FormularioPageModule)
},
{
  path: '',
  redirectTo: '/tabs/tab1',
```

Quedando la vista de la App, con la nueva página:



A continuación, vamos a hacer uso de los estilos, para ello Ionic hace uso de **Sass** (<https://sass-lang.com/>). **Sass** es un preprocesador CSS. Un preprocesador CSS es una herramienta que nos permite generar, de manera automática, hojas de estilo, añadiéndoles características que no

tiene CSS, y que son propias de los lenguajes de programación, como pueden ser variables, funciones, selectores anidados, herencia, ...

Tutorial básico: <https://sass-lang.com/guide>







Ionic está pensado para usar Sass, de tal manera que tenemos un tema establecido, el cual podemos personalizarlo sin añadir estilos CSS. Para ello debemos de acceder a **theme/variables.css**, ahí podremos personalizar nuestro estilo.

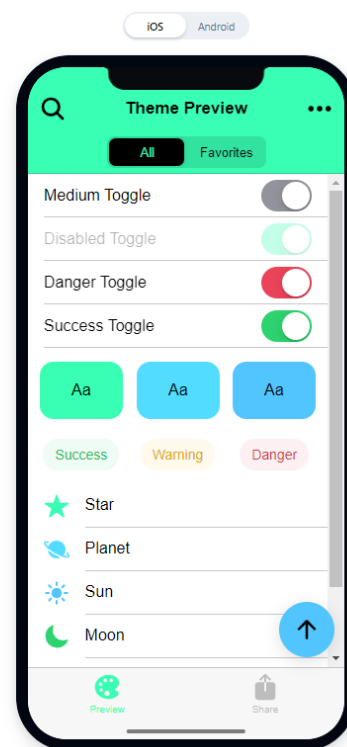
Para facilitar personalizar los temas, podemos emplear el generador de ionic:

<https://ionicframework.com/docs/theming/color-generator>

Color Generator

Create custom color palettes for your app's UI. Update a color's hex values, check the demo app on the right to confirm, then copy and paste the generated code directly into your Ionic project.

	Primary	<input type="text" value="#38ffb3"/>
	Secondary	<input type="text" value="#52dcff"/>
	Tertiary	<input type="text" value="#52c5ff"/>
	Success	<input type="text" value="#2dd36f"/>
	Warning	<input type="text" value="#ffc409"/>
	Danger	<input type="text" value="#eb445a"/>



Botones (ion-button)

En la documentación de Ionic podemos observar todas las opciones de los botones (<https://ionicframework.com/docs/api/button>).

Vamos a crear en **inicio.page.html**, diferentes estilos de botones basados en la documentación de ionic:

```

<ion-button>Default</ion-button> <ion-button [disabled]="true">Disabled</ion-button>
<ion-button expand="block">Block</ion-button> <ion-button expand="full">Full</ion-button>
<ion-button>Default</ion-button> <ion-button shape="round">Round</ion-button>
<ion-button>Default</ion-button>
<ion-button fill="clear">Clear</ion-button>
<ion-button fill="outline">Outline</ion-button>
<ion-button fill="solid">Solid</ion-button>
<ion-button size="small">Small</ion-button>
<ion-button size="default">Default</ion-button>
<ion-button size="large">Large</ion-button>
<ion-button>
  <ion-icon slot="start" name="star"></ion-icon>
  Left Icon
</ion-button>

<ion-button>
  Right Icon
  <ion-icon slot="end" name="star"></ion-icon>
</ion-button>

<ion-button>
  <ion-icon slot="icon-only" name="star"></ion-icon>
</ion-button>
<ion-button>Default</ion-button>
<ion-button color="primary">Primary</ion-button>
<ion-button color="secondary">Secondary</ion-button>
<ion-button color="tertiary">Tertiary</ion-button>
<ion-button color="success">Success</ion-button>
<ion-button color="warning">Warning</ion-button>
<ion-button color="danger">Danger</ion-button>
<ion-button color="light">Light</ion-button>
<ion-button color="medium">Medium</ion-button>
<ion-button color="dark">Dark</ion-button>

</ion-content>

```



Listas (ion-list)

Las listas son elementos muy comunes en las aplicaciones y pueden alojar cualquier tipo de componentes (botones, toogles, iconos, imágenes, etc).

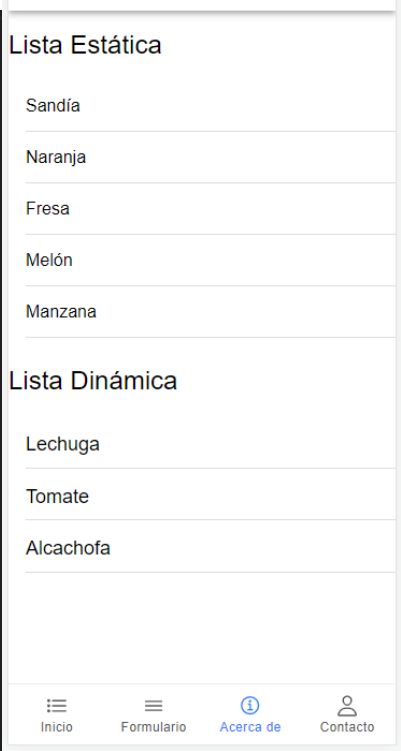
Vamos a crear una lista de elementos estáticos y dinámicos en **tab2.page.html**:

Es muy interesante poder cargar las listas desde un array, modificamos **tab2.page.ts** y **tab2.page.html**:

```

<h2>Lista Estática</h2>
<ion-list>
  <ion-item>
    <ion-label>Sandía</ion-label>
  </ion-item>
  <ion-item>
    <ion-label>Naranja</ion-label>
  </ion-item>
  <ion-item>
    <ion-label>Fresa</ion-label>
  </ion-item>
  <ion-item>
    <ion-label>Melón</ion-label>
  </ion-item>
  <ion-item>
    <ion-label>Manzana</ion-label>
  </ion-item>
</ion-list>
<h2>Lista Dinámica</h2>
<ion-list>
  <ion-item *ngFor="let item of lista"><h5>{{item.name}}</h5></ion-item>
</ion-list>
</ion-content>

```



```

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

@Component({
  selector: 'app-tab2',
  templateUrl: 'tab2.page.html',
  styleUrls: ['tab2.page.scss']
})
export class Tab2Page {

  constructor() {}

  lista:Array<any>=[{name:"Lechuga"},{name:"Tomate"},{name:"Alcachofa"}];
}

```

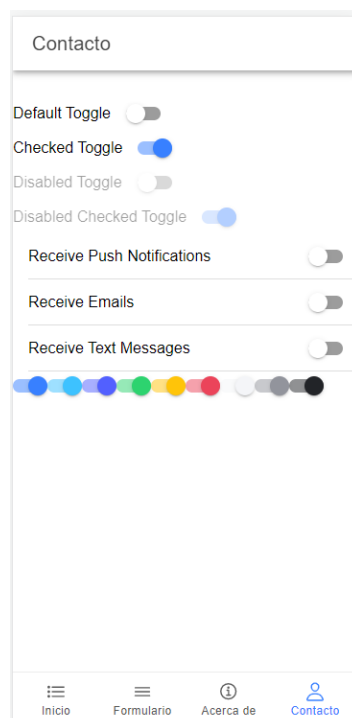
Botones Check (ion-toggle)

Al igual que con los botones, ionic presenta los ion-toggle (<https://ionicframework.com/docs/api/toggle>).

A continuación, dentro de **formulario.page.html**, introducimos las siguientes etiquetas:

```
<ion-toggle>Default Toggle</ion-toggle><br /><br />
<ion-toggle [checked]="true">Checked Toggle</ion-toggle><br /><br />
<ion-toggle [disabled]="true">Disabled Toggle</ion-toggle><br /><br />
<ion-toggle [checked]="true" [disabled]="true">Disabled Checked Toggle</ion-toggle>

<ion-list>
  <ion-item>
    <ion-toggle>Receive Push Notifications</ion-toggle>
  </ion-item>
  <ion-item>
    <ion-toggle>Receive Emails</ion-toggle>
  </ion-item>
  <ion-item>
    <ion-toggle>Receive Text Messages</ion-toggle>
  </ion-item>
</ion-list>
<ion-toggle aria-label="Primary toggle" color="primary" [checked]="true"></ion-toggle>
<ion-toggle aria-label="Secondary toggle" color="secondary" [checked]="true"></ion-toggle>
<ion-toggle aria-label="Tertiary toggle" color="tertiary" [checked]="true"></ion-toggle>
<ion-toggle aria-label="Success toggle" color="success" [checked]="true"></ion-toggle>
<ion-toggle aria-label="Warning toggle" color="warning" [checked]="true"></ion-toggle>
<ion-toggle aria-label="Danger toggle" color="danger" [checked]="true"></ion-toggle>
<ion-toggle aria-label="Light toggle" color="light" [checked]="true"></ion-toggle>
<ion-toggle aria-label="Medium toggle" color="medium" [checked]="true"></ion-toggle>
<ion-toggle aria-label="Dark toggle" color="dark" [checked]="true"></ion-toggle>
```



Ion-card

Las tarjetas son una pieza estándar de la interfaz de usuario que sirve como punto de entrada a información más detallada. Una tarjeta puede ser un solo componente, pero a menudo se compone de un encabezado, título, subtítulo y contenido.

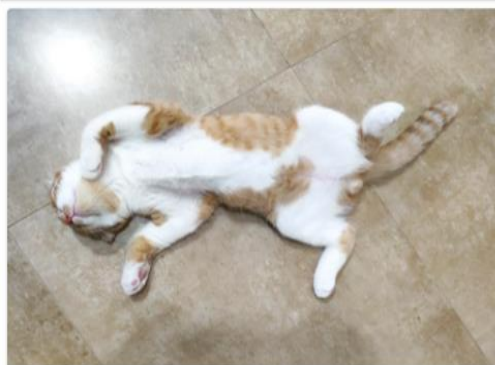
<https://ionicframework.com/docs/api/card>

Modificar **tab2.page.html**. Añadir una imagen en **assets/img**:

```
<ion-header [translucent]="true">
  <ion-toolbar>
    <ion-title>
      Ei mollis
    </ion-title>
  </ion-toolbar>
</ion-header>

<ion-content [fullscreen]="true">
  <ion-card>
    
    <ion-card-header>
      <ion-card-subtitle>minim dicant sensibus</ion-card-subtitle>
      <ion-card-title>Melius fabella</ion-card-title>
    </ion-card-header>
    <ion-card-content>
      Mei eu mollis albucius, ex nisl contentiones vix. Duo persius volutpat at, cu iuvaret epicuri mei. Duo poss
    </ion-card-content>
  </ion-card>
</ion-content>
```

Ei mollis



minim dicant sensibus

Melius fabella

Mei eu mollis albucius, ex nisl contentiones vix. Duo persius volutpat at, cu iuvaret epicuri mei. Duo posse pertinacia no, ex dolor contentiones mea. Nec omnium utamur dignissim ne. Mundi lucilius salutandi an sea, ne sea aequo iudico comprehensam. Populo delicatissimi ad pri. Ex vitae accusam vivendum pro.

Ejercicio 2. Desarrollo Aplicación Side

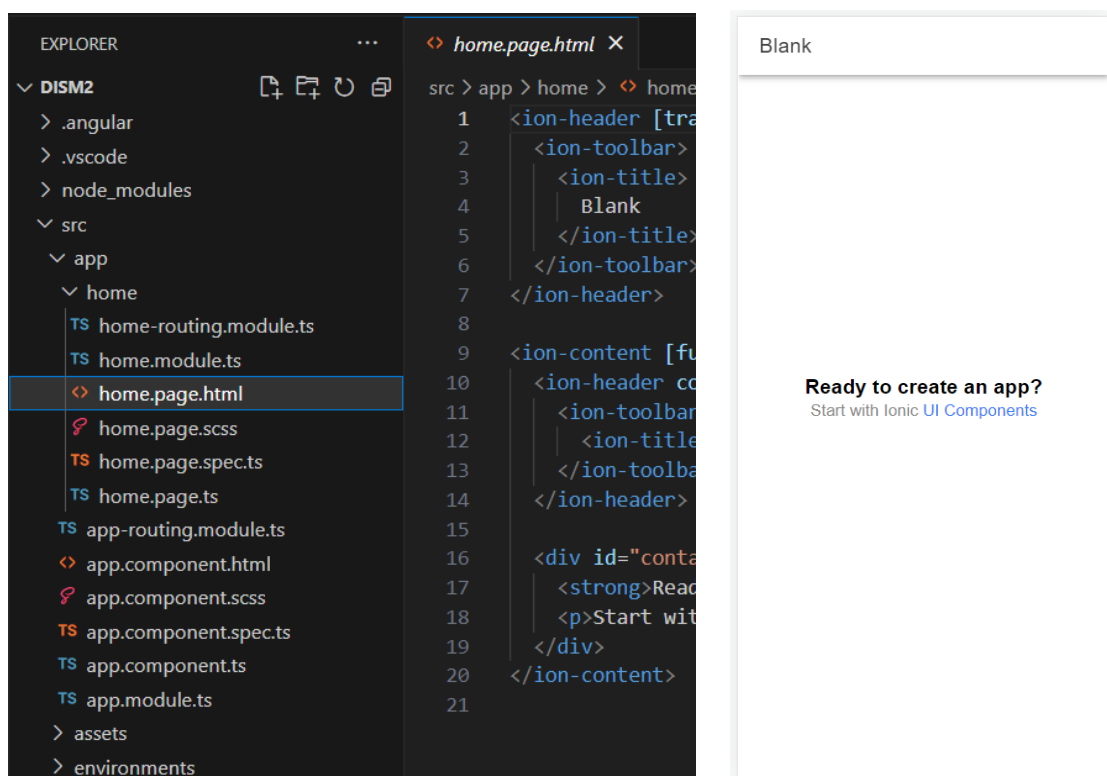
Para generar una aplicación mediante paginación side, podemos utilizar el asistente de ionic para generar este tipo de aplicación o bien como es nuestro caso hacerlo nosotros desde una aplicación blank.

Desde Ventana de comandos, escribimos

```
ionic start DISM2
```

En las opciones le indicamos framework angular y type blank.

Ejecutamos y observamos que solo tenemos una página en nuestra App.



A continuación vamos a añadir una nueva página mediante el cli de ionic, así como la paginación al estilo side.

Ejecutamos dentro de la carpeta DISM2:

```
ionic generate page credits
```

La plantilla principal de nuestra aplicación, la que se va a cargar siempre al inicio se encuentra en el archivo **app.component.html**. Vamos a modificarla con este contenido:

```
<ion-app>
  <ion-split-pane contentId="main-content">
    <ion-menu contentId="main-content">
      <ion-header>
        <ion-toolbar>
          <ion-title>Menú</ion-title>
        </ion-toolbar>
      </ion-header>
      <ion-content>
        <ion-list>
          <ion-menu-toggle auto-hide="false">
            <ion-item [routerDirection]="root" [routerLink]="'/home'">
              <ion-icon slot="start" [name]="home"></ion-icon>
              <ion-label>Inicio</ion-label>
            </ion-item>
            <ion-item [routerDirection]="root" [routerLink]="'/credits'">
              <ion-icon slot="start" [name]="help-circle"></ion-icon>
              <ion-label>Créditos</ion-label>
            </ion-item>
          </ion-menu-toggle>
        </ion-list>
      </ion-content>
    </ion-menu>
    <ion-router-outlet id="main-content"></ion-router-outlet>
  </ion-split-pane>
</ion-app>
```

Lo primero que hacemos es envolver todo en una etiqueta **ion-split-pane**, esto permite adaptarse a pantallas más grandes (como tablets) y muestra el menú justo al lado de su contenido. Le asignamos la propiedad **contentId="main-content"** que será el id a asignar a la etiqueta **ion-router-outlet** donde irá el contenido de la aplicación.

Después tenemos **ion-menu** que es el componente que contiene el menú y al que asignaremos el parámetro **contentId="main-content"**, dentro del mismo añadimos su contenido igual que si fuera una página, hemos añadido un **ion-header** con un toolbar y dentro el título del menú.

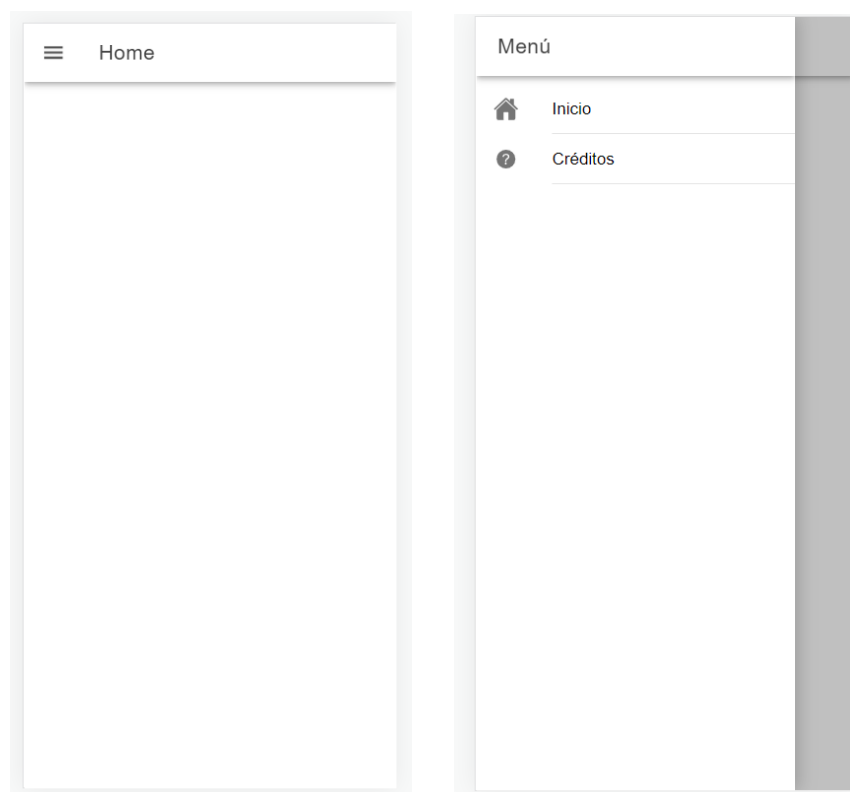
Después tenemos un **ion-content** y dentro hemos creado un listado con dos items con las opciones del menú.

En el parámetro **routerLink** de cada item definimos la ruta que tenemos que cargar al pulsar. La ruta está entre comillas simples dentro de las comillas dobles, esto es para que lo tome como un literal en lugar de una variable.

Ahora nos falta añadir a cada página **home.page.html** y **credits.page.html** en su cabecera un **ion-menu-button**:

```
src > app > home > <> home.page.html > ...
1  <ion-header [translucent]="true">
2    <ion-toolbar>
3      <ion-title>
4        Home
5      </ion-title>
6      <ion-buttons slot="start">
7        <ion-menu-button></ion-menu-button>
8      </ion-buttons>
9    </ion-toolbar>
10 </ion-header>
11
12 <ion-content [fullscreen]="true">
13   <ion-header collapse="condense">
14     <ion-toolbar>
15       <ion-title size="large">Home</ion-title>
16     </ion-toolbar>
17   </ion-header>
18 </ion-content>
```

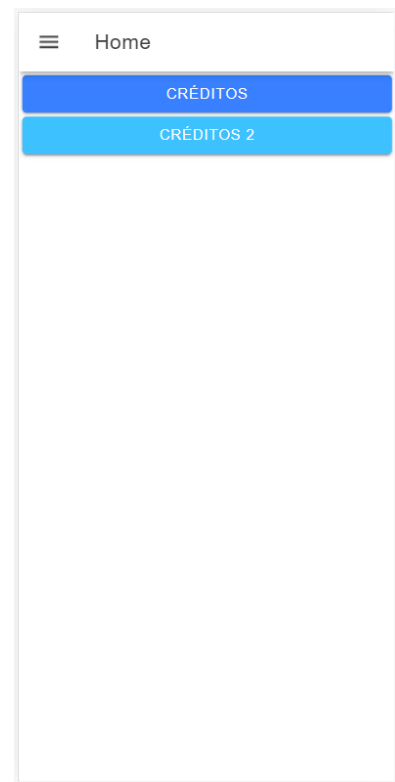
El resultado final:



Podemos añadir un botón que nos lleve de una página a otra en la propia página [home.page.html](#) o añadir código typescript en [home.page.ts](#), que realice esa labor:

```
src > app > home > <> home.page.html > ...
11
12 <ion-content [fullscreen]="true">
13   <ion-header collapse="condense">
14     <ion-toolbar>
15       <ion-title size="large">Home</ion-title>
16     </ion-toolbar>
17   </ion-header>
18   <ion-button [routerLink]="['/credits']" color="primary" expand="block">Créditos</ion-button>
19   <ion-button (click)="mostrarPagina()" color="secondary" expand="block">Créditos 2</ion-button>
20 </ion-content>
21 |
```

```
src > app > home > TS home.page.ts > ...
1 import { Component } from '@angular/core';
2 import { Router } from '@angular/router';
3
4 @Component({
5   selector: 'app-home',
6   templateUrl: 'home.page.html',
7   styleUrls: ['home.page.scss'],
8 })
9 export class HomePage {
10
11   constructor(private router:Router) {}
12
13   mostrarPagina() {
14     this.router.navigate(['credits']);
15   }
16 }
```



Realizar los cambios oportunos para permitir en Créditos los dos botones para regresar a Home, de forma similar al ejemplo anterior.

Ejercicio 3. Servicios

Los servicios son proveedores que se encargan del manejo de datos, bien sean de la base de datos, desde una API REST, etc.

En este caso vamos a acceder a una API REST para obtener desde un servidor remoto un listado de usuarios. Para este pequeño ejemplo vamos a utilizar RANDOM USER GENERATOR (<https://randomuser.me/>) que como se indica en su web es una API libre de código abierto para generar datos de usuario aleatorios para realizar pruebas.

Lo que vamos a hacer es simplemente realizar una llamada a esta API donde recibiremos como respuesta un listado de usuarios que mostraremos en nuestra App.

El primer paso será crear una aplicación en blanco (blank,angular,ngmodules):

```
ionic start DISM3 blank
```

Seleccionamos Angular como framework, posteriormente accedemos a la carpeta generada **\DISM3**.

A continuación, vamos a crear un servicio donde gestionaremos la comunicación con el servidor remoto:

```
ionic g service services/http
```

Podemos comprobar cómo se ha creado una carpeta **services** y dentro, un archivo .ts con el nombre **http.service.ts**.

Para realizar las peticiones al servidor vamos a utilizar un paquete de angular llamado **HttpClient**, así que vamos a importarlo en nuestro servicio:

```
TS http.service.ts X
src > app > services > TS http.service.ts > ? HttpService
1  import { Injectable } from '@angular/core';
2  import { HttpClient } from '@angular/common/http'
3
4  @Injectable({
5    providedIn: 'root'
6  })
7  export class HttpService {
8    constructor(public http: HttpClient) { }
9  }
10
```

Ahora debemos importar y declarar como provider en **app.module.ts** el servicio que acabamos de crear, también debemos importar y declarar en los imports **HttpClientModule**:

```
TS app.module.ts X
src > app > TS app.module.ts > ...
1  import { NgModule } from '@angular/core';
2  import { BrowserModule } from '@angular/platform-browser';
3  import { RouteReuseStrategy } from '@angular/router';
4
5  import { IonicModule, IonicRouteStrategy } from '@ionic/angular';
6
7  import { AppComponent } from './app.component';
8  import { AppRoutingModule } from './app-routing.module';
9
10 import { HttpClientModule } from '@angular/common/http'
11 import { HttpService } from './services/http.service'
12
13 @NgModule({
14   declarations: [AppComponent],
15   imports: [BrowserModule, IonicModule.forRoot(), AppRoutingModule, HttpClientModule],
16   providers: [{ provide: RouteReuseStrategy, useClass: IonicRouteStrategy }, HttpService],
17   bootstrap: [AppComponent],
18 })
19 export class AppModule {}
```

A continuación vamos a añadir un método que llamaremos **loadUsers** a nuestro servicio para obtener la lista de usuarios desde el servidor, por lo tanto editamos el archivo **http.service.ts** y añadimos la siguiente función después del constructor:











```
TS http.service.ts X
src > app > services > TS http.service.ts > ...
1  import { Injectable } from '@angular/core';
2  import { HttpClient } from '@angular/common/http'
3
4  @Injectable({
5    providedIn: 'root'
6  })
7  export class HttpService {
8    constructor(public http: HttpClient) { }
9    loadUsers() {
10     return this.http.get('https://randomuser.me/api/?results=25')
11   }
12 }
```

Ahora vamos a crear la vista en **home.page.html** para mostrar un botón que llamará a la función **cargarUsuarios**, y un listado de items con los usuarios que crearemos recorriendo con ***ngFor** (angular) el array usuarios que posteriormente vamos a crear en el controlador:

```
<> home.page.html ●
src > app > home > <> home.page.html > ...
1  <ion-header [translucent]="true">
2    <ion-toolbar>
3      <ion-title>
4        Usuarios
5      </ion-title>
6    </ion-toolbar>
7  </ion-header>
8
9  <ion-content [fullscreen]="true">
10    <ion-list>
11      <ion-item *ngFor="let usuario of usuarios">
12        <ion-avatar slot="start">
13          <img [src]="usuario.picture.medium">
14        </ion-avatar>
15        <ion-label>
16          <h2>{{usuario.name.first}}</h2>
17          <p>{{usuario.email}}</p>
18        </ion-label>
19      </ion-item>
20    </ion-list>
21    <ion-button expand="block" (click)="cargarUsuarios()">Cargar Usuarios</ion-button>
22  </ion-content>
```


Ahora toca modificar **home.page.ts** para obtener los datos desde el servicio y mostrarlos en la vista para ello editamos **home.page.ts** e importamos el **service httpProvider** que acabamos de crear, declaramos un array usuarios, añadimos un método para obtener los usuarios del servicio:

```
TS home.page.ts X
src > app > home > TS home.page.ts > ...
1  import { Component } from '@angular/core';
2  import { HttpService } from '../services/http.service';
3
4  @Component({
5    selector: 'app-home',
6    templateUrl: 'home.page.html',
7    styleUrls: ['home.page.scss'],
8  })
9  export class HomePage {
10   usuarios: any[]=[];
11   constructor(private http: HttpService) {}
12   cargarUsuarios () {
13     this.http.loadUsers().subscribe(
14       (res:any) => {
15         this.usuarios=res.results;
16       },
17       (error) => {
18         console.error(error);
19       }
20     )
21   }
22 }
```

Usuarios	
	Benito benito.crespo@example.com
	Andrea andrea.mitchell@example.com
	Todd todd.daniels@example.com
	Emily emily.nielsen@example.com
	آرتین artyn.shylyrd@example.com
	Anthony anthony.phillips@example.com
	Hans-Joachim hans-joachim.beyer@example.com
	Nanna nanna.moller@example.com
	Jeanne jeanne.jean-baptiste@example.com
	Yarina yarina.yakubovskiy@example.com

Ejercicio 4. Creación de una fake Api Rest

Las APIs REST forman el back end para aplicaciones móviles. Cuando se desarrollan aplicaciones, algunas veces no disponemos de las APIs REST listas para ser usadas para propósitos de desarrollo. Para realizar peticiones GET/POST/PUT/ DELETE dentro de la app móvil necesitamos un servidor que permita tratar con datos JSON.

json-server proporciona la funcionalidad para establecer un servidor fake API REST con un mínimo esfuerzo, permitiendo peticiones GET / POST /PUT / DELETE.

<https://github.com/typicode/json-server>

Lo primero, será instalar la librería json-server:

```
npm install -g json-server
```

Una vez instalado, crearemos un fichero **db.json**, con datos :

```
{
  "usuarios": [
    {
      "id": "1",
      "nombre": "Sergio"
    },
    {
      "id": "2",
      "nombre": "Estela"
    },
    {
      "id": "3",
      "nombre": "Susana"
    },
    {
      "id": "4",
      "nombre": "Hugo"
    }
  ]
}
```

El siguiente paso será ejecutar json-server:

```
json-server db.json
```

Y a continuación si accedemos a la url <http://localhost:3000/usuarios> , podremos realizar pruebas bien con el navegador o mediante la aplicación Postman (<https://www.postman.com/>).

Ejercicio 5. Desarrollo Aplicación Consumo fake REST API

En este ejercicio vamos a desarrollar una aplicación en blanco, en la que al cargar la aplicación nos muestre los datos de la fake Rest Api anterior y a su vez nos permita añadir, editar y borrar datos.

Extenderemos la API Fake anterior con dos nuevos campos: Email y Edad.

Creemos la App DISM4, mediante el cli de Ionic, con las opciones Blanc y Angular.

A continuación, una vez generada la App, generamos dos nuevas páginas (edición, nuevo)

```
ionic generate page editar
```

```
ionic generate page nuevo
```

El siguiente paso consiste en modificar **app.module.ts**, para añadir el módulo **httpClientmodule**, para poder consumir una api.

```
src > app > TS app.module.ts > ...
1  import { NgModule } from '@angular/core';
2  import { BrowserModule } from '@angular/platform-browser';
3  import { RouteReuseStrategy } from '@angular/router';
4
5  import { IonicModule, IonicRouteStrategy } from '@ionic/angular';
6
7  import { AppComponent } from './app.component';
8  import { AppRoutingModule } from './app-routing.module';
9
10 import { HttpClientModule } from '@angular/common/http';
11
12 @NgModule({
13   declarations: [AppComponent],
14   imports: [BrowserModule, IonicModule.forRoot(), AppRoutingModule, HttpClientModule],
15   providers: [{ provide: RouteReuseStrategy, useClass: IonicRouteStrategy }],
16   bootstrap: [AppComponent],
17 })
18 export class AppModule {}
19
```

El siguiente paso será crear una clase con el objeto a consumir, en nuestro caso **Usuario**.

```
ionic generate class models/usuario
```

Abrimos la clase generada y la modificamos con:

```
src > app > models > TS usuario.ts > ...
1   export class Usuario {
2     id: number=0;
3     nombre: string="";
4     edad: string="";
5     email: string="";
6   }
7
```

Ahora vamos a generar el servicio encargado de consumir la fake api:

```
ionic generate service services/api
```

A continuación modificamos el fichero api.services.ts, con:

```
src > app > services > TS api.service.ts > A ApiService
1   import { Injectable } from '@angular/core';
2   import { HttpClient, HttpHeaders, HttpResponse } from '@angular/common/http';
3   import { Usuario } from '../models/usuario';
4   import { Observable, throwError } from 'rxjs';
5   import { retry, catchError } from 'rxjs/operators';
6
7   @Injectable({
8     providedIn: 'root'
9   })
10  export class ApiService {
11
12    // API Path
13    basePath='http://localhost:3000/';
14
15    constructor(private http: HttpClient) { }
16
17    // Opciones Http
18    httpOptions= {
19      headers: new HttpHeaders ({
20        'Content-Type': 'application/json'
21      })
22    }
23
24    // Manejador de errores API
25    handleError(error: HttpResponse) {
26      if (error.error instanceof ErrorEvent) {
27        console.log("Ha ocurrido un error: ",error,error.message);
28      }
29      else {
30        console.error("Codigo Error $ {error.status}, "+"Body: ${error.error}");
31      }
32      return throwError ('Ha sucedido un problema, reintentalo más tarde');
33    }
34
```

Lo primero es importar la clase usuario, indicar donde tenemos nuestra API y una gestión de errores.

Y a continuación en la misma clase vamos añadiendo las diferentes operaciones de servicio asociados a un CRUD.

```

36 createItem(item:Usuario): Observable<Usuario> {
37   return this.http.post<Usuario>(this.basePath+'usuario/',JSON.stringify(item),this.httpOptions).pipe(retry(2),catchError(this.handleError));
38 }
39
40 getItem(id:number): Observable<Usuario> {
41   return this.http.get<Usuario>(this.basePath+'usuario/'+id).pipe(retry(2),catchError(this.handleError));
42 }
43
44 getList(): Observable<Usuario> {
45   return this.http.get<Usuario>(this.basePath+'usuario/').pipe(retry(2),catchError(this.handleError));
46 }
47
48 updateItem(item:Usuario) {
49   console.log("UPDATE: "+JSON.stringify(item));
50   return this.http.put(this.basePath+"usuario/"+item.id,JSON.stringify(item),this.httpOptions).pipe(retry(2),catchError(this.handleError));
51 }
52
53 deleteItem(id:number) {
54   return this.http.delete<Usuario>(this.basePath+"usuario/"+id,this.httpOptions).pipe(retry(2),catchError(this.handleError));
55 }
56 }
57

```

El siguiente paso será introducir la lógica en **home.page.ts**, primero importamos *ApiService* para poder hacer uso de las llamadas a nuestra Api y luego importamos *Router* y *navigatioExtras* para navegar por nuestras páginas y poder pasar parámetros entre páginas, a continuación, añadimos la lógica asociada para permitirnos listar usuarios, borrar y editar:

```

src > app > home > TS home.page.ts > ...
1  import { Component } from '@angular/core';
2  import { ApiService } from '../services/api.service';
3  import { Router } from '@angular/router';
4  import { Usuario } from '../models/usuario';
5  import { NavController } from '@ionic/angular';
6
7
8  @Component({
9    selector: 'app-home',
10   templateUrl: 'home.page.html',
11   styleUrls: ['home.page.scss'],
12 })
13 export class HomePage {
14
15   UsuarioData: any;
16
17   constructor(public apiService:ApiService, private nav: NavController) {
18     this.UsuarioData=[];
19   }
20   ionViewWillEnter() {
21     this.getAllUsuarios();
22   }
23   getAllUsuarios() {
24     this.apiService.getList().subscribe(Response => { this.UsuarioData = Response; });
25   }
26   deleteUsuario(item:Usuario) {
27     this.apiService.deleteItem(item.id).subscribe(Response => this.getAllUsuarios());
28   }
29   editUsuario(item: Usuario) {
30     this.nav.navigateForward('editar', {state: {item}});
31   }
32 }

```

Ahora modificamos **home.page.html**, añadiendo el contenido para mostrar el listado de usuarios, un botón para editar y otro para borrar, finalmente tras la lista de elementos, situamos un nuevo botón para añadir usuarios:

```
src > app > home > <> home.page.html > ...
1 <ion-header [translucent]="true">
2   <ion-toolbar>
3     <ion-title>
4       Listado Usuarios
5     </ion-title>
6   </ion-toolbar>
7 </ion-header>
8
9 <ion-content class="ion-padding">
10  <ion-list>
11    <ion-item *ngFor="let item of UsuarioData">
12      <ion-label>
13        <h2>Id: {{ item.id }}</h2>
14        <h2>Nombre: {{ item.nombre }}</h2>
15        <p>Email: {{item.email}}</p>
16        <p>Edad: {{item.edad}}</p>
17      </ion-label>
18      <ion-button (click)="editUsuario(item)" color="warning" size="small"><ion-icon name="create"></ion-icon></ion-button>
19      <ion-button (click)="deleteUsuario(item)" color="danger" size="small"><ion-icon name="trash"></ion-icon></ion-button>
20    </ion-item>
21  </ion-list>
22  <ion-button [routerLink]="['/nuevo']" expand="block">Nuevo Usuario</ion-button>
23 </ion-content>
24
```

El siguiente paso será modificar **editar.page.html**, donde añadimos los campos a editar empleando *databinding* de Angular mediante *ngModel*, luego creamos botón para actualizar registro y otro para volver a la página home:

```
src > app > editar > <> editar.page.html > ...
1 <ion-header [translucent]="true">
2   <ion-toolbar>
3     <ion-title>Editar</ion-title>
4   </ion-toolbar>
5 </ion-header>
6
7 <ion-content class="ion-padding">
8   <ion-item>
9     <ion-label>Id: {{UsuarioData.id}}</ion-label>
10  </ion-item>
11  <ion-item>
12    <ion-label>Nombre: </ion-label>
13    <ion-input [(ngModel)]="UsuarioData.nombre" placeholder="Nombre"></ion-input>
14  </ion-item>
15  <ion-item>
16    <ion-label>Edad: </ion-label>
17    <ion-input [(ngModel)]="UsuarioData.edad" placeholder="Edad"></ion-input>
18  </ion-item>
19  <ion-item>
20    <ion-label>Email: </ion-label>
21    <ion-input [(ngModel)]="UsuarioData.email" placeholder="Email"></ion-input>
22  </ion-item>
23  <ion-button (click)="update()" color="success" expand="block">Actualizar</ion-button>
24  <ion-button [routerLink]="['/home']" color="warning" expand="block">Cancelar</ion-button>
25 </ion-content>
26
```

y **editar.page.ts**, con la lógica necesaria para la página anterior:

```
src > app > editar > TS editar.page.ts > ...
1 import { Component, OnInit } from '@angular/core';
2 import { ActivatedRoute, Router } from '@angular/router';
3 import { Usuario } from '../models/usuario';
4 import { ApiService } from '../services/api.service';
5
6 @Component({
7   selector: 'app-editar',
8   templateUrl: './editar.page.html',
9   styleUrls: ['./editar.page.scss'],
10 })
11 export class EditarPage implements OnInit {
12   id:number=0;
13   UsuarioData: any;
14   Data: any;
15
16   constructor(public activatedRoute: ActivatedRoute,public router: Router,public apiService:ApiService) {
17   }
18
19   ngOnInit() {
20     this.activatedRoute.queryParams.subscribe(params=> {
21       this.activatedRoute.queryParams.subscribe(p => {
22         this.Data = this.router.getCurrentNavigation()?.extras?.state as Usuario;
23         this.UsuarioData=this.Data.item;
24       })
25     });
26   }
27   update() {
28     this.apiService.updateItem(this.UsuarioData).subscribe(response => {
29       this.router.navigate(['home']);
30     })
31   }
32 }
```

Para finalizar tendremos que modificar la página **nuevo.page.html** y **nuevo.page.ts** para permitir añadir nuevos Usuarios, de forma similar a editar:

```
src > app > nuevo > <> nuevo.page.html > ...
1 <ion-header [translucent]="true">
2   <ion-toolbar>
3     <ion-title>Nuevo</ion-title>
4   </ion-toolbar>
5 </ion-header>
6
7 <ion-content class="ion-padding">
8   <ion-item>
src > app > nuevo > TS nuevo.page.ts > ...
1 import { Component, OnInit } from '@angular/core';
2 import { Usuario } from '../models/usuario';
3 import { ApiService } from '../services/api.service';
4 import { ActivatedRoute, Router } from '@angular/router';
5
6 @Component({
7   selector: 'app-nuevo',
8   templateUrl: './nuevo.page.html',
9   styleUrls: ['./nuevo.page.scss'],
10 })
11 export class NuevoPage implements OnInit {
12   UsuarioData: any;
13   constructor(public activatedRoute: ActivatedRoute,public router: Router,public apiService:ApiService) { }
14
15   ngOnInit() {
16     this.UsuarioData=new Usuario();
17   }
18
19   newUsuario() {
20     this.apiService.createItem(this.UsuarioData).subscribe((Response)=> { this.router.navigate(['home'])});
21   }
22 }
```

Y el resultado final:

Listado Usuarios

Id: 1

Nombre: Sergio

Email: sergio@ua.es

Edad: 23

✓

✕

Id: 2

Nombre: Estela

Email: estela@ua.es

Edad: 19

✓

✕

Id: 3

Nombre: Susana

Email: susana@ua.es

Edad: 27

✓

✕

Id: 4

Nombre: Hugo

Email: hugo@ua.es

Edad: 18

✓

✕

NUEVO USUARIO

Editar

Id: 1

Nombre: Sergio

Edad: 23

Email: sergio@ua.es

ACTUALIZAR

CANCELAR

Nuevo

Nombre: Nombre

Edad: Edad

Email: Email

AÑADIR

CANCELAR

Ejercicio 6. Geolocalización

En este ejercicio vamos a desarrollar una aplicación en blanco, en la que obtendremos nuestra geolocalización, el primer paso será crear una aplicación en blanco habilitando **Capacitor**:

```
ionic start GeolocalizacionApp blank --type=angular
```

Instalamos los plugins necesarios:

```
cd GeolocalizacionApp
npm install cordova-plugin-geolocation
npm install @awesome-cordova-plugins/geolocation
npm install @awesome-cordova-plugins/core --save
ionic cap sync
```

A continuación añadimos el plugin instalado en la App creada en **app.module.ts**:

```
src > app > TS app.module.ts > ...
1  import { NgModule } from '@angular/core';
2  import { BrowserModule } from '@angular/platform-browser';
3  import { RouteReuseStrategy } from '@angular/router';
4  import { IonicModule, IonicRouteStrategy } from '@ionic/angular';
5  import { AppComponent } from './app.component';
6  import { AppRoutingModule } from './app-routing.module';
7  import { Geolocation } from '@awesome-cordova-plugins/geolocation/ngx';
8
9  @NgModule({
10   declarations: [AppComponent],
11   imports: [BrowserModule, IonicModule.forRoot(), AppRoutingModule],
12   providers: [Geolocation, { provide: RouteReuseStrategy, useClass: IonicRouteStrategy }],
13   bootstrap: [AppComponent],
14 })
15 export class AppModule {}
```

Modificamos el fichero **home.page.ts** para introducir el código necesario para obtener la geolocalización y mostrarla en pantalla:

```
src > app > home > TS home.page.ts > ...
1  ∨ import { Component } from '@angular/core';
2  import { Geolocation } from '@awesome-cordova-plugins/geolocation/ngx';
3
4  ∨ @Component({
5      selector: 'app-home',
6      templateUrl: 'home.page.html',
7      styleUrls: ['home.page.scss'],
8  })
9  ∨ export class HomePage {
10
11      latitude: any=0;
12      longitude: any=0;
13
14      constructor(private geolocation:Geolocation) {}
15  ∨ options= {
16      timeout: 10000,
17      enableHighAccuracy:true,
18      maximumAge: 3600
19  }
20  ∨ getCurrentCoordinates() {
21  ∨      this.geolocation.getCurrentPosition().then((resp)=> {
22          this.latitude=resp.coords.latitude;
23          this.longitude=resp.coords.longitude;
24  ∨      }).catch((error) => {
25          console.log("Error obteniendo Geolocalización");
26      })
27  }
28 }
```

Y por último modificamos **home.page.html**, para que utilice la función para obtener las coordenadas:

```
src > app > home > home.page.html > ion-header > ion-toolbar
Go to component
1 <ion-header [translucent]="true">
2   <ion-toolbar>
3     <ion-title>
4       Ejemplo Ionic Geolocalización
5     </ion-title>
6   </ion-toolbar>
7 </ion-header>
8
9 <ion-content [fullscreen]="true">
10  <div class="ion-padding">
11    <ion-button (click)="getCurrentCoordinates()" expand="block">Obtener Geolocalización</ion-button>
12    <ion-list>
13      <ion-list-header>
14        <ion-label>Coordenadas de Localización</ion-label>
15      </ion-list-header>
16      <ion-item>
17        <ion-label>Latitud</ion-label>
18        <ion-badge color="danger" slot="end">{{latitude}}</ion-badge>
19      </ion-item>
20      <ion-item>
21        <ion-label>Longitud</ion-label>
22        <ion-badge color="danger" slot="end">{{longitude}}</ion-badge>
23      </ion-item>
24    </ion-list>
25  </div>
26 </ion-content>
```

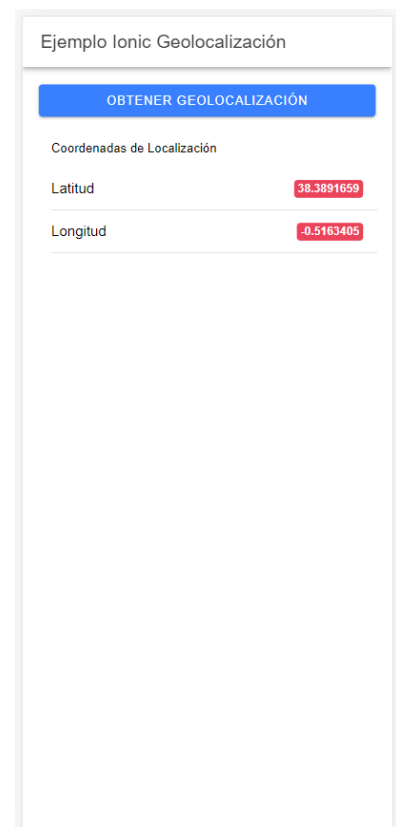
Ejecutamos la aplicación con:

```
ionic serve --lab
```

y podemos comprobar que podemos obtener los datos de posicionamiento.

Adicionalmente podríamos obtener los datos de georreferenciación con:

<https://ionicframework.com/docs/native/native-geocoder>



Ejercicio 7. Diseño API Rest OpenApi

Para el diseño de la API Rest mediante OpenApi emplearemos cualquiera de sus utilidades para escribir código y visualizarlo, para nuestro ejercicio vamos a diseñar un simple Hola Mundo:

<https://swagger.io/tools/swagger-editor/>
<https://editor.swagger.io/>

```
1 openapi: 3.0.0
2 servers:
3   - description: Servidor ApiRest DTIC
4     url: http://dtic.org/HolaMundo/1.0.0
5 info:
6   description: Esto es un Hola Mundo
7   version: "1.0.0"
8   title: Hola Mundo Dtic
9   contact:
10    email: info@dtic.org
11  license:
12    name: Apache 2.0
13    url: 'http://www.apache.org/licenses/LICENSE-2.0.html'
14
15 paths:
16   /holamundo:
17     get:
18       tags:
19         - HolaMundoServicio
20       description: GET Hola Mundo
21       parameters:
22         - in: query
23           name: nombreEntradaHolaMundo
24           description: nombre Entrada Hola Mundo
25           required: true
26           schema:
27             type: string
28       responses:
29         '200':
30           description: Respuesta Hola Mundo Correcto
31           content:
32             application/json:
33               schema:
34                 type: object
35                 properties:
36                   nombre:
37                     type: string
38                   contador:
39                     type: integer
40         '400':
41           description: Hola Mundo Error
42           content:
43             application/json:
44               schema:
45                 type: object
46                 properties:
47                   message:
48                     type: string
```

Conforme vamos escribiendo el código podemos visualizar como quedaría visualmente la API, incluso probarla.

Hola Mundo Dtic

1.0.0 OAS3

Esto es un Hola Mundo

[Contact the developer](#)

[Apache 2.0](#)

Servers

http://dtic.org/HolaMundo/1.0.0 - Servidor ApiRest DTIC

HolaMundoServicio

GET

/holamundo

GET Hola Mundo

Parameters

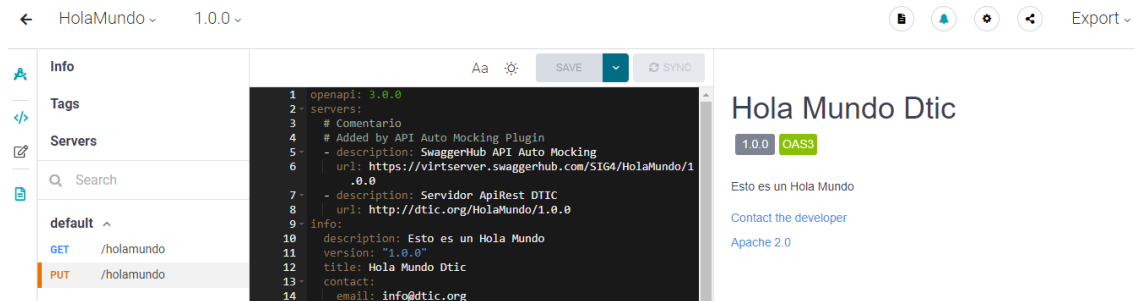
Try it out

Name	Description
nombreEntradaHolaMundo <small>* required</small> string (query)	nombre Entrada Hola Mundo <div>nombreEntradaHolaMundo</div>

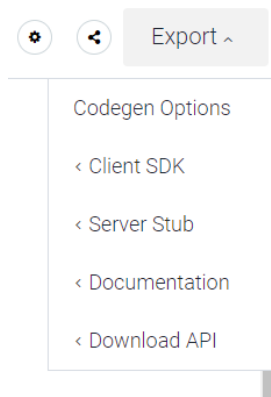
Ejercicio 8. Generación de stub nodejs mediante Swagger

En este ejercicio generaremos mediante swagger el stub para nodejs , de forma que solo necesitemos modificar este esqueleto para generar nuestro servicio web.

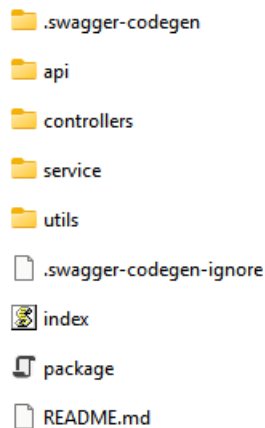
Accedemos a swagger y una vez dentro del editor:



Pulsamos en EXPORT → Server Stub → Nodejs-SERVER



Esto nos generará un zip, con la siguiente estructura:



El fichero README.md nos especifica los pasos a seguir para lanzar nuestra API Rest. La carpeta API, contiene el fichero OpenApi diseñado mediante swagger, en controllers y services se encuentran los ficheros generados donde debemos de establecer nuestra lógica de negocio.

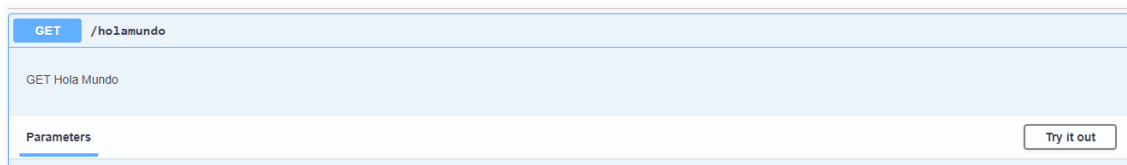
Por Ejemplo, para nuestro HolaMundo, podemos modificar dentro de DefaultService.js y cambiar el resultado de la petición:

```
service > JS DefaultService.js > holamundoGET > holamundoGET
1  'use strict';
2
3
4  /**
5   * GET Hola Mundo
6   *
7   * nombreEntradaHolaMundo String nombre Entrada Hola Mundo
8   * returns inline_response_200
9   */
10 exports.holamundoGET = function(nombreEntradaHolaMundo) {
11   return new Promise(function(resolve, reject) {
12     var examples = {};
13     examples['application/json'] = {
14       "contador" : 999,
15       "nombre" : "Hola Mundo Prueba 2022-23"
16     };
17     if (Object.keys(examples).length > 0) {
18       resolve(examples[Object.keys(examples)[0]]);
19     } else {
20       resolve();
21     }
22   });
23 }
24
```

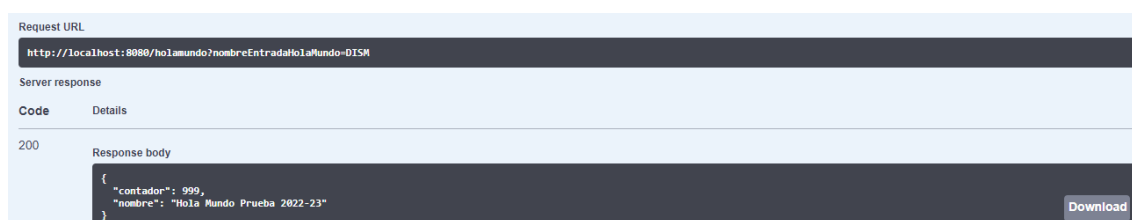
Si ejecutamos:

npm start

Accedemos a: <http://localhost:8080/docs> y hacemos un Try it Out



Y comprobamos resultado:



Ejercicio 9. Generación API con acceso a BD a partir de OpenAPI

A partir de la BD DISM, donde disponemos de una tabla Usuarios, deberemos diseñar mediante OpenApi las operaciones CRUD para dicha table (consultar, borrar, actualizar, nuevo) asociándolas a las peticiones correspondientes (get, delete, post, put).

El primer paso crear la BD, para ello mediante MySQL restauraremos la BD del script proporcionado.

El siguiente paso será diseñar la API mediante OpenApi con la ayuda de swagger editor.

Posteriormente crearemos mediante el generador de código de swagger el servicio web , para ello, Generate Server -> NodeJs Server.

Por último, añadiremos la lógica de negocio necesaria para acceder a BD (Ejercicios anexo)

Finalmente podremos comprobar como en el ejercicio 8 al acceder a la URL <http://localhost:8080/docs> el correcto funcionamiento de la API.