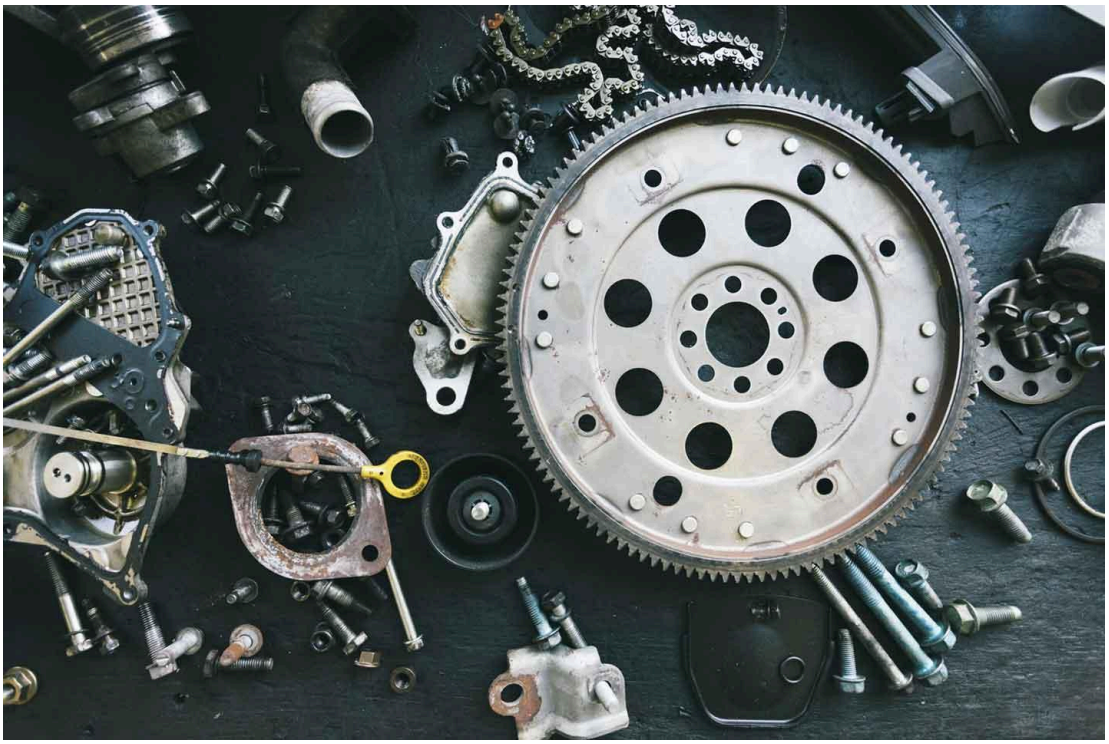
Ionic v8

Inicio



Progresa
Una formación de futuro

ÍNDICE

Componentes

Índice:

Setup

Para empezar vamos a crear un nuevo proyecto donde usaremos todos los componentes que tiene Ionic en diferentes páginas de nuestra aplicación.

Pero antes de crear un nuevo proyecto, vamos a crear una carpeta donde guardar los proyectos. En nuestro ordenador creamos una carpeta llamada **Ionic**.
Ahora vamos a crear nuestro proyecto en dicha carpeta. Para ello tenemos que tener un cmd o terminal situado en dicha carpeta. El comando de ionic para crear un proyecto es **ionic start**. Hay varios tipos de proyecto para crear con Ionic que podemos encontrar descritos en la documentación oficial:
- https://ionicframework.com/docs/installation/cli
- **Blank**: totalmente vacía
- **Tabs**: con pestañas
- **SideMenu**: Con un menú lateral

Nosotros la vamos a crear vacía y luego veremos como añadir las diferentes opciones. En nuestro **cmd** o **terminal** situado en nuestra carpeta Ionic:

ionic start MyApp blank --capacitor

Nos preguntará el framework que queremos utilizar, seleccionamos **Angular**. A continuación el tipo de arquitectura, si ngmodules o standalone, en nuestro caso **standalone**.

Una vez creado, abrimos la carpeta que nos acaba de crear, MyApp, con Webstorm. A continuación abrimos una terminal de Webstorm y ejecutamos el compilador de Ionic para ver nuestra App por defecto que nos crea Ionic. En la **terminal**:

ionic serve

INICIO

Ionic nos crea por defecto una página inicial Home. Vamos a crear nosotros nuestra propia página de Inicio. Para ello en una nueva **terminal**, siempre en la ruta del proyecto:

ionic generate page pages/inicio –spec=false

Ahora borramos la página home que nos crea Ionic por defecto y modificamos nuestro archivo de rutas, para quitar la ruta de home. En **app.routes.ts**:

- borramos el path home
- Redirigimos la ruta vacía ‘ ‘ que apunta a ‘Home’ a ‘Inicio’

```
const routes: Routes = [
 {path: '', redirectTo: 'inicio', pathMatch: 'full'},
 {path: 'inicio',
   loadChildren: () => import('./pages/inicio/inicio.module')
.then( m => m.InicioPageModule)
}
,];
```

Una vez ya tenemos nuestro inicio, vamos a crear nuestras primeras páginas para empezar a trabajar con los componentes.

ionic generate page pages/grid-buttons –spec=false

ionic generate page pages/alert –spec=false

ionic generate page pages/action-sheet –spec=false

ionic generate page pages/avatar –spec=false

Ahora que tenemos las páginas vamos a empezar con nuestra página de inicio. Lo primero que vamos a hacer es abrir el html de la página que corresponderá la parte visible.

Por defecto las páginas de Ionic vemos que tienen:

- **ion-header**: La parte superior de la app o cabecera.
- **ion-content**: La parte central de nuestra App.

En este **inicio.page.html** vamos a añadir una lista de items que serán los componentes que vayamos añadiendo a nuestra aplicación:

```
<ion-header class="ion-no-border">
 <ion-toolbar>
   <ion-title class="ion-text-center">Mi App</ion-title>
 </ion-toolbar>
</ion-header>

<ion-content class="ion-padding">
 <ion-list>
                  <ion-list-header      color="danger">Mis
componentes</ion-list-header>
   <ion-item>
     <ion-icon slot="start" name="card"></ion-icon>
     <ion-label>Etiqueta</ion-label>
   </ion-item>
 </ion-list>
</ion-content>
```

Explicación de los elementos usados

`<ion-header class="ion-no-border" [translucent]="true">`

Nos quita la línea divisora entre la cabecera y el contenido. Translucent nos proporciona un difuminado en la cabecera para iOS.

`<ion-title class="ion-text-center">`

Nos permite poner un título en la cabecera y con la clase lo centramos.

`<ion-list>`

Creamos una lista de elementos de Ionic

```
<ion-list-header color="danger">
```

Cabecera de la lista con color primario de la aplicación

```
<ion-item>
```

Objetos de ionic que pueden contener varios elementos, como texto, iconos, imágenes, etiquetas…

```
<ion-icon slot="start" name="card">
```

Un icono de Ionic, con el slot le decimos el lugar donde aparecerá, con name que icono mostramos.

```
<ion-label>
```

Lo utilizamos para usar texto dentro de los objetos de Ionic

Todos los iconos que utilicemos en la aplicación deberemos precargarlos con la función addIcons en el constructor de la App o del componente principal:

```
constructor() {
 addIcons({card})
}
```

Lógica de inicio

A continuación vamos a definir una interfaz de un componente que tendrá:

- El nombre del componente
- La ruta que cargará la página de este componente
- El icono del componente.

En la **terminal**:

ionic generate interface common/interfaces

Hay 2 estrategias de uso de interfaces: crear un archivo por interfaz o crear varias interfaces en un mismo archivo. Todo depende de lo grandes que sean las interfaces. En nuestro caso es una interfaz pequeña así que usaremos un archivo genérico para almacenarla.

En el archivo **interfaces.ts**, recién creado en la carpeta common, definimos la interfaz Componente y la exportamos para poder usarla en el resto de nuestra aplicación:

```
export interface Componente {
 nombre: string;
 ruta: string;
 icono: string;
}
```

También vamos a definir un archivo json donde guardaremos la información del menú, en nuestro caso las páginas de los componentes que iremos creando, para no guardarla en un array dentro de nuestro ts. En la carpeta **assets** **botón derecho** → **new** → **directory** y lo llamamos **data**.

En data botón derecho → new → File y lo llamamos **menu.json** y añadimos los 2 primeros componentes:

```
[
 {
 "nombre": "Grid y Buttons",
 "ruta": "/grid-buttons",
 "icono": "grid"
},
 {
   "nombre": "Alert",
   "ruta": "/alert",
   "icono": "logo-apple-appstore"
 },
 {
 "nombre": "Action Sheet",
 "ruta": "/action-sheet",
 "icono": "american-football"
},
 {
   "nombre": "Avatar",
   "ruta": "/avatar",
   "icono": "beaker"
 }]
```

Utilización de un Servicio

Ahora que ya tenemos los elementos de nuestro menú, vamos a traerlos a nuestro ts y añadirlos a un array. Para poder hacer esto vamos a necesitar un servicio de Angular. Para ello vamos a la **terminal**:

ionic generate service services/data –skip-tests

En el servicio vamos a definir la función que conectará con el archivo json y traerá los componentes para nuestro menú. En **data.service.ts**:

```
export class DataService {
private http: HttpClient = inject(HttpClient);
 getComponentesMenu() : Observable<Componente[]{
   return this.http.get('/assets/data/menu.json');
 }
}
```

Explicación

```
private http: HttpClient = inject(HttpClient);
```

Aquí inyectamos el cliente http para poder utilizarlo en el servicio. Se trata de un módulo que nos permite hacer peticiones http(get, post, put, delete) a urls. En nuestro caso nos sirve para traer los datos del menu.json con un get.

```
 return this.http.get('/assets/data/menu.json');
```

Aquí utilizamos el protocolo http inyectado con una petición get para traer los datos de la url, en este caso es una ruta local.

Además hay que hacer un **provider del HttpClient** en **main.ts** que no hace Angular automáticamente:

```
bootstrapApplication(AppComponent, {
 providers: [
   { provide: RouteReuseStrategy, useClass: IonicRouteStrategy },
   provideIonicAngular(),
   provideRouter(routes, withPreloading(PreloadAllModules)),
   provideHttpClient(withFetch())
 ],
});
```

GUARDAR LA INFORMACIÓN

Ahora que ya tenemos la información recogida por el servicio, tenemos que guardarla en algún sitio. Esto lo vamos a hacer en nuestro **inicio.page.ts**:

```
export class InicioPage implements OnInit {

 componentes: Componente[] = [];
 constructor(private dataService: DataService) { }

 ngOnInit() {
   this.cargarComponentes();
 }

private cargarComponentes() {
 this.dataService.getComponentesMenu().subscribe(
   {
     next:    (data: Componente[]) => {
       this.componentes = data;
     },
     error:    (err: Error) => console.error(err),
     complete:    () => {
       console.log('Data loaded.')}
   }
 );
}
}
```

Explicación

```
 private dataService: DataService = inject(DataService);
```

Inyectamos el servicio para poder usar sus funciones

```
 componentes: Componente[] = [];
```

Definimos el array donde guardaremos nuestros componentes.

```
 ngOnInit() {this.cargarComponentes();}
```

En el ngOnInit llamamos a las funciones que queremos que se ejecuten en cuanto cargue el componente. En nuestro caso la que rellenará el array de componentes si todo va bien, o devolverá un error en caso contrario.

```
private cargarComponentes() {
 this.dataService.getComponentesMenu().subscribe(
   {
     next:    (data: Componente[]) => {
       this.componentes = data;
     },
     error:    (err: Error) => console.error(err),
     complete:    () => {
       console.log('Data loaded.')}
   }
 );
}
```

Esta es la función que conecta con el servicio y cargamos los componentes que nos devuelve del archivo menu.json en nuestro array componentes.

VISIBILIZAR LOS DATOS

Para ver los componentes que hemos cargado, tenemos que realizar unos cambios en nuestro archivo **inicio.page.html** que es el que va a mostrar los elementos del menú:

```
<ion-header class="ion-no-border">
 <ion-toolbar>
   <ion-title class="ion-text-center">Mi App</ion-title>
 </ion-toolbar>
</ion-header>

<ion-content class="ion-padding">

 <ion-list>
                    <ion-list-header       color="danger">Mis
componentes</ion-list-header>
   @for (componente of componentes;track componente){
     <ion-item [routerLink]="componente.ruta">
                                      <ion-icon     slot="start"
[name]="componente.icono"></ion-icon>
       <ion-label>{{componente.nombre}}</ion-label>
     </ion-item>
   }

 </ion-list>

</ion-content>
```

Explicación

```
@for (componente of componentes;track componente){
```

Utilizamos el @for de Angular para recorrer nuestro array de componentes y crear un item por cada uno.

```
<ion-item [routerLink]="componente.ruta">
```

Añadimos un routerLink para ir a la página del componente, utilizando el atributo ruta del componente.

```
<ion-icon slot="start" [name]="componente.icono"></ion-icon>
```

Cargamos el icono que hemos especificado en el atributo del componente en el name del ion-icon.

```
<ion-label>{{componente.nombre}}</ion-label>
```

Por último vemos el texto del Componente mediante el label