Ionic

Desarrollo de Aplicaciones Híbridas

Tabla de contenidos

1

Resumen

Resumen del tutorial de Ionic.

Objetivos

- Conocer la arquitectura de las aplicaciones Ionic
- Aprender a utilizar los componentes básicos de Ionic para crear una app.
- Interactuar con APIs
- Incorporar el uso de la cámara y el GPS a apps Ionic

2

Introducción

Ionic es un framework para el desarrollo de aplicación móviles híbridas con HTML, CSS y JavaScript (Angular). Las apps tienen una única base de código y se pueden empaquetar y distribuir como aplicaciones nativas en las principales tiendas de aplicaciones para móviles, como App Store, Google Play y Microsoft Store.

La gran ventaja de desarrollar con Ionic es que se pueden aprovechar los conocimientos que se tienen del desarrollo de aplicaciones web para el desarrollo de aplicaciones móviles multiplataforma. Si además, se conoce Angular, esta transición es bastante directa.

Por último, gracias a los plugins de Apache Cordova[1], podemos acceder a las capacidades nativas del dispositivo mediante JavaScript.

[1] https://cordova.apache.org/plugins

3

Configuración del entorno

- Instalar Node.js
- Instalar Apache Cordova `sudo npm install -g cordova`
- Instalar Ionic CLI `sudo npm install -g ionic`
- Instalar TypeScript `sudo npm install -g typescript`

```
$ ionic info

cli packages: (/usr/local/lib/node_modules)

    @ionic/cli-utils  : 1.19.2
    ionic (Ionic CLI) : 3.20.0

local packages:

    @ionic/app-scripts : 3.1.8
    Ionic Framework    : ionic-angular 3.9.2

System:

    Node : v8.9.4
    npm  : 3.5.2
    OS   : Linux 4.13

Misc:

    backend : pro
```

4

Primera app Ionic

Para el desarrollo de proyectos Ionic usaremos Ionic CLI. Con él podremos iniciar una app, generar elementos de la app (componentes, páginas, providers, …), iniciar un servidor web local para probar la app en el proceso de desarrollo, y demás operaciones útilies en el proceso de desarrollo de una app Ionic.

Parar crear una app Ionic basta con escribir desde una terminal `ionic start` seguido del nombre de la app. Por ejemplo, `ionic start holamundoionic`.

A continuación tendremos que seleccionar el tipo de proyecto que queremos crear: basado basado en pestañas, en blanco, con barra lateral, … Elegiremos la opción `tutorial`.

```
$ ionic start holamundoionic

? What starter would you like to use: (Use arrow keys)
# tabs ..... ionic-angular A starting project with a simple
 tabbed interface
  blank ..... ionic-angular A blank starter project
  sidemenu ..... ionic-angular A starting project with a side menu
 with navigation in the content area
  super ..... ionic-angular A starting project complete with
 pre-built pages, providers and best practices for Ionic development.
  conference ..... ionic-angular A project that demonstrates a
 realworld application
  tutorial ..... ionic-angular A tutorial based project that goes
 along with the Ionic documentation
  aws ..... ionic-angular AWS Mobile Hub Starter
```

En el paso siguiente tendremos que indicar si queremos integrar la app con Cordova, de forma que habilite en la app las funciones nativas de los dispositivos. Para esta primera app indicaremos que no.

A continuación se creará un directorio con el nombre del proyecto, se descargará la plantilla del tipo de proyecto elegido y se instalarán las dependencias, una operación que dura unos minutos.

Por último, tenemos que indicar si queremos instalar el Ionic Pro SDK. Para este primer proyecto, también elegiremos que no.

Con esto, el proyecto ya estará creado.

> El proyecto cuenta con un repositorio Git. Al final de la creación del proyecto se realizará de forma automática una operación `add` y una operación `commit` con la versión preliminar del proyecto.

Podemos ver la aplicación funcionando entrando en el directorio de la aplicación y ejecutando `ionic serve`

```
$ cd holamundoionic
$ ionic serve
....
[07:33:35]  lint started ...
[07:33:35]  build dev finished in 15.63 s
[07:33:35]  watch ready in 15.71 s
[07:33:35]  dev server running: http://localhost:8100/

[OK] Development server running!
     Local: http://localhost:8100
     External: http://192.168.1.13:8100
     DevApp: holamundoionic@8100 on mtorres-ThinkPad-T440

[07:33:38]  lint finished in 3.65 s
```

Tras unos instantes tenemos funcionando la app en un servidor de pruebas con función *livereload* al que podremos acceder por el puerto 8100 tal y como se indica al final de las indicaciones de la creación del proyecto.

Se recomienda activar el modo desarrollador del navegador

- Mozilla Firefox: Web Developer → Toggle Tools. En la barra de herramientas activar el Responsive Design Mode.
- Google Chrome: More Tools → Developer Tools. En la barra de herramientas activar el Toggle device toolbar.

Ionic Lab

Ionic ofrece el Ionic Lab en el que podremos ver la app en las tres plataformas principales (Android, iOS y Windows). También nos ofrece una referencia rápida a los componentes Ionic y a la documentación oficial de Ionic.

Para activar el Ionic Lab lanzaremos la aplicación con `ionic serve -l`. La aplicación estará disponible en `http://localhost:8100/ionic-lab`

4.1. El archivo `index.html`

El código de la app se encuentra en `src`. Ahí encontramos el archivo `index.html`, el archivo desde el que se carga la app.

Ejemplo 4.1. Fragmento del archivo `src/app/index.html`

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
...
<body>
...

  <!-- Ionic's root component and where the app will load -->
  <ion-app></ion-app> ❶

...
</body>
</html>
```

❶ Indicación de cargar el componente raíz

El archivo `index.html` tiene un fragmento JavaScript comentado relacionado con la activación de *service worker* relacionado con las aplicaciones web progresivas[1]

El archivo `app.html` contiene el componente raiz de la app, `<ion-app></ion-app>`. Este componente cargará el contenido de `src/app/app.html`, la vista raiz de la app.

El archivo `src/app/app.html`.

```
<ion-menu [content]="content">

  <ion-header> ❶
    <ion-toolbar>
      <ion-title>Pages</ion-title>
    </ion-toolbar>
  </ion-header>

  <ion-content> ❷
    <ion-list>
      <button ion-item *ngFor="let p of pages" (click)="openPage(p)">
        {{p.title}}
```

[1] https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API

```
      </button>
    </ion-list>
  </ion-content>

</ion-menu>

<ion-nav [root]="rootPage" #content swipeBackEnabled="false"></ion-nav>
```

❶ Cabecera de la app

❷ Contenido de la app. Mediante un bucle cargará la lista de páginas y le asociará un controlador de evento

4.2. Páginas de la app y el archivo `app.module.ts`

Las app Ionic están compuestas de componentes página, situados en la carpeta `src/pages`. Durante la ejecución, estas páginas son organizadas en una pila y son gestionadas por el NavController[2] mostrándolas con una operación `push` y liberándolas con una operación `pop`.

Dado que una app Ionic es una aplicación Angular, los componentes, providers y directivas tienen que ser declaradas en el archivo `src/app/app.module.ts` antes de que las use la aplicación.

Declaración de los componentes página en el archivo `src/app/app.module.ts`.

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule, ErrorHandler } from '@angular/core';
import { IonicApp, IonicModule, IonicErrorHandler } from 'ionic-
angular';
import { MyApp } from './app.component';

import { HelloIonicPage } from '../pages/hello-ionic/hello-ionic'; ❶
import { ItemDetailsPage } from '../pages/item-details/item-details';
import { ListPage } from '../pages/list/list';

import { StatusBar } from '@ionic-native/status-bar';
import { SplashScreen } from '@ionic-native/splash-screen';

@NgModule({
```

[2] https://ionicframework.com/docs/api/navigation/NavController/

```
  declarations: [ ❷
    MyApp,
    HelloIonicPage,
    ItemDetailsPage,
    ListPage
  ],
  imports: [
    BrowserModule,
    IonicModule.forRoot(MyApp),
  ],
  bootstrap: [IonicApp],
  entryComponents: [ ❸
    MyApp,
    HelloIonicPage,
    ItemDetailsPage,
    ListPage
  ],
  providers: [
    StatusBar,
    SplashScreen,
    {provide: ErrorHandler, useClass: IonicErrorHandler}
  ]
})
export class AppModule {}
```

❶ Importación de las páginas

❷ Incorporación de las páginas al array `declarations[]`

❸ Incorporación de las páginas al array `entryComponents[]`

4.3. Modificación de las páginas generadas.

A modo de ejemplo modificaremos el contenido de la página de inicio (`src/app/pages/hello-ionic/hello-ionic.html`) incluyendo texto ficticio Lorem Ipsum[3]. Añadiremos al final del tag `<ion-content>` una lista de dos elementos.

```
  <ul>
    <li>Lorem ipsum dolor sit, amet consectetur.</li>
    <li>Suspendisse lobortis senectus justo facilisis hendrerit, euismod
 interdum morbi.</li>
  </ul>
```

[3] https://getlorem.com/es/

Tras guardar los cambios se recargará la app de forma automática y deberá mostrar algo similar a lo siguiente

5

Creación de una app de ejemplo

A modo de ejemplo crearemos una app que muestre los repositorios de un usuario de GitHub y muestre el archivo `README.md` del repositorio seleccionado.

5.1. Generación de páginas en Ionic

Las páginas son generadas con `ionic generate page` *`nombrePagina`* o `ionic g page` *`nombrePagina`* en su forma abreviada.

Para generar la página `repos` escribiriemos

```
$ ionic g page repos

[OK] Generated a page named repos!
```

Este comando generará una carpeta `src/app/pages/repos` con cuatro archivos:

- `repos.html`: Vista de la página. Aquí colocaremos el contenido de la página
- `repos.module.ts`: Módulo de la clase generada
- `repos.scss`: Estilos propios de la página
- `repos.ts`: Controlador de la página

5.2. Modificación de la página generada

En primer lugar cambiaremos el título de la página y el texto mostrado en la vista. También debemos incluir un botón para activar el menú desde la página de repositorios

Ejemplo 5.1. El archivo `src/app/pages/repos/repos.html`

```
<ion-header>
  <ion-navbar>
    <button ion-button menuToggle> ❶
      <ion-icon name="menu"></ion-icon>
    </button>
    <ion-title>Repositorios</ion-title> ❷
  </ion-navbar>
</ion-header>


<ion-content padding>
  Página vista de repositorios ❸
</ion-content>
```

❶ Incorporación de un botón para mostrar el menú

❷ Modificación del título

❸ Modificación del contenido

A continuación cambiaremos en el controlador el texto mostrado en la consola al cargar la vista.

Ejemplo 5.2. El archivo `src/app/pages/repos/repos.ts`

```
import { Component } from '@angular/core';
import { IonicPage, NavController, NavParams } from 'ionic-
angular';

@IonicPage()
@Component({
  selector: 'page-repos', ❶
  templateUrl: 'repos.html',
})
export class ReposPage {

  constructor(public navCtrl: NavController, public navParams:
 NavParams) {
  }

  ionViewDidLoad() {
    console.log('Cargada la página de repositorios'); ❷
  }

}
```

❶ Selector asignado a la vista y que incluiremos en las vistas en la que quedamos incluir esta página

❷ Mensaje a mostrar en la consola al cargar la página

5.3. Modificación del archivo `app.module.ts`

Tras crear una página deberemos actualizar `app.module.ts` añadiendo la página a la zona de importaciones y modificando las propiedades `declarations` y `entryPoints` de `@NgModule`.

Aprovecharemos para eliminar las referencias de las páginas `ItemDetailsPage` y `ListPage` creadas para el tutorial, y para importar `HttpModule`, el cual será necesario más adelante cuando definamos el servicio.

Ejemplo 5.3. Actualización del archivo `app.module.ts` para incluir la página de repositorios

```
...
import { HelloIonicPage } from '../pages/hello-ionic/hello-ionic';
import { ReposPage } from '../pages/repos/repos'; ❶

import {HttpModule} from '@angular/http'; ❷

...
@NgModule({
  declarations: [
    MyApp,
    HelloIonicPage,
    ReposPage ❸
  ],
  imports: [
    BrowserModule,
    IonicModule.forRoot(MyApp),
    HttpModule ❹
  ],
  ...
  entryComponents: [
    MyApp,
    HelloIonicPage,
    ReposPage ❺
  ],
  ...
})
export class AppModule {}
```

❶ Importación de la página de repositorios. Hay que indicar la ruta al archivo desde la ubicación de `app.module.ts`

❷ Importación de `HttpModule` para la creación del servicio

❸ Incoporación de la página de repositorios al array `declarations`

❹ Incoporación de `HttpModule` al array `imports`

❺ Incoporación de la página de repositorios al array `entrytComponents`

5.4. Modificación del archivo app.component.ts

En este archivo realizaremos los cambios relacionados con el menú lateral para eliminar las referencias a las páginas no necesarias y para incluir la referencia a la página creada. Será necesario añadir la página a la ista de importaciones.

En el archivo también indicaremos el texto que se usará en el menú lateral para hacer referencia a la página creada.

Ejemplo 5.4. Actualización del archivo `app.component.ts` para incluir la página de repositorios

```
...
import { HelloIonicPage } from '../pages/hello-ionic/hello-ionic';
import { ReposPage } from '../pages/repos/repos'; ❶
...
export class MyApp {
  @ViewChild(Nav) nav: Nav;

  // make HelloIonicPage the root (or first) page
  rootPage = HelloIonicPage; ❷
  pages: Array<{title: string, component: any}>;

  constructor(
    public platform: Platform,
    public menu: MenuController,
    public statusBar: StatusBar,
    public splashScreen: SplashScreen
  ) {
    this.initializeApp();

    // set our app's pages
    this.pages = [ ❸
      { title: 'Inicio', component: HelloIonicPage },
      { title: 'Repos de un usuario', component: ReposPage }
    ];
  }
  ...
}
```

❶ Importación de la página de repositorios. Mantenemos la página de inicio del tutorial por comodidad. Lo habitual sería cambiarla pero exige otras modificaciones que evitaremos por ahora

❷ Seguimos manteniedo la página de inicio del tutorial

❸ Array con las páginas a cargar desde el menú lateral y sus títulos asociados.

5.5. Modificación de la página de inicio

Cambiar la cabecera y el contenido de la página de esta forma

Ejemplo 5.5. Actualización del archivo `src/app/pages/hello-ionic.html`

```
<ion-header>
  <ion-navbar>
    <button ion-button menuToggle>
      <ion-icon name="menu"></ion-icon>
    </button>
    <ion-title>GitHub Ionic</ion-title> ❶
  </ion-navbar>
</ion-header>


<ion-content padding> ❷

  <h3>GitHub Ionic</h3>

  <p>
    App Ionic que muestra los repositorios GitHub de un usuario
 accediendo a la <a href = "https://developer.github.com/v3/">API
 de GitHub</a>
  </p>

</ion-content>
```

❶ Nueva cabecera

❷ Nuevo contenido

Con todos los cambios realizados la aplicación deberá verse así

5.6. Creación del modelo para los repositorios

Crearemos un modelo para almacenar cada uno de los repositorios GitHiub de un usuario. Se trata de una clase que guarda los campos que queremos extraer de los repositorios a través la API, ignorando aquellos campos en los que no estemos interesados. En nuestro caso nos quedaremos con el nombre y descripción de los repositorios (`name` y `description`).

Para una mayor organización, almacenaremos los modelos del proyecto en una carpeta `src/models`.

5.7. Creación de un servicio para obtener los repos de un usuario

Vamos a crear un servicio para obtener los repos de un usuario GitHub desde `https://api.github.com/users/`*`nombreusuario`*`/repos`, donde *`nombreusuario`* corresponde con un usuario GitHub. En esta aplicación, el nombre de usuario será obtenido a partir de un cuadro de texto que crearemos más adelante.

Los servicios se crean con `ionic generate provider` *`servicio`* o `ionic g provider` *`servicio`* en su forma abreviada. Para crear el servicio `repos` escribiremos

```
$ ionic g provider repos
```

Esto creará una carpeta `src/providers` y un archivo `src/providers/repos.ts`.

El decorador `@Injectable`

Al crear un servicio, Ionic CLI le añade el decorador `@Injectable`. A aquellos componentes en los que queramos usar el servicio, lo *inyectaremos* en su constructor y así podrán acceder a la funcionalidad implementada por los métodos del servicio.

Al inyectar un servicio en el constructor se creará una variable de instancia desde la que podremos acceder a la funcionalidad implementada por los métodos del servicio.

El servicio contará con un método que devolverá un array con los repositorios del usuario proporcionado. En el servicio generado tenemos que hacer las siguientes modificaciones:

- Cambiar la importación de `HttpClient` por la de `Http`

- Modificar el tipo de datos del parámetro `http` en el constructor para adaptarlo a la importación del paso anterior.
- Importar el modelo del repositorio, `Observable` y el método `map`. Dichos componentes no están disponibles en Angular y son incorporados de la librería de ReactJS (RxJS).
- Crear un método `getRepos()` que obtendrá los repos del usuario que se pase como parámetro. Este método devuelve los repositorios como un *observable*[1]

Observables

Debido a la naturaleza asíncrona de JavaScript, necesitamos una forma de consumir de los servicios de forma que se vayan incorporando a él los datos conforme se vayan obteniendo y que les lleguen a los componentes que consumen los servicios. Para ello, los servicios definirán métodos observables y los consumidores se suscribirán a su contenido.

Imagina un observable como un flujo de datos al que te puedes suscribir.

[1] https://angular.io/guide/observables

Ejemplo 5.6. El servicio `src/providers/repos/repos.ts

```
import { Http } from '@angular/http'; ❶
import { Injectable } from '@angular/core';

import { Repo } from '../../models/repo'; ❷
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/operator/map';

@Injectable()
export class ReposProvider {

  constructor(public http: Http) { } ❸

  getRepos(username): Observable<Repo[]> { ❹
    return this.http.get(`https://api.github.com/users/${username}/
repos`).map(res => <Repo[]>res.json()); ❺
  }
}
```

❶ Modificación del componente Http importado

❷ Importación de la clase del modelo del repositorio, del componente `Observable` y del método `map` (El método `map` crea un array con el resultado de llamar a la función proporcionada sobre cada elemento del array)

❸ Modificación del tipo del parámetro del constructor y eliminación del cuerpo predefinido del constructor.

❹ Método que devuelve la lista de repositorios como un array observable a partir de una llamada a la API de GitHub

❺ Arrow function para devolver la respuesta JSON como un array de objetos repositorio.

Arrow functions

Son una forma compacta de definir funciones anónimas

```
// Ejemplo 1
([param] [, param]) => {
  statements
}

// Ejemplo 2
param => expression
```

Y este sería su código JavaScript equivalente

```
// Ejemplo 1
function ([param] [, param]) {
  statements
}

// Ejemplo 2
function (param) {
  return expression
}
```

5.8. Creación del controlador de la vista de repositorios

El controlador devolverá un array de objetos repositorio a partir de un nombre de usuario de GitHub. Por tanto, contará con una variable de instancia para recibir el nombre de usuario de la vista y con un array de objetos repositorio (los cuales habrá que importar) que entregará a la vista para que los presente. Además, realizará inyección de dependencias inyectando en su constructor el servicio creado. Por tanto, habrá que importar también el servicio.

Ejemplo 5.7. El archivo `src/app/pages/repos/repos.ts`

```
import { Component } from '@angular/core';
import { IonicPage, NavController, NavParams } from 'ionic-
angular';

import { Repo } from '../../models/repos'; ❶
import { ReposProvider } from '../../providers/repos/repos'; ❷

@IonicPage()
@Component({
  selector: 'page-repos',
  templateUrl: 'repos.html',
})
export class ReposPage {
  repos: Repo[];
  public username;

  constructor(public navCtrl: NavController,
    public navParams: NavParams,
    private reposProvider: ReposProvider) {  } ❸

  getRepos() { ❹
    this.reposProvider.getRepos(this.username).subscribe(reposArray
 => {
      this.repos = reposArray;
    })
  }


  ionViewDidLoad() {  }
}
```

❶ Importación del modelo de repositorio

❷ Importación del servicio

❸ Inyección del servicio

❹ Método que se suscribe al servicio y construye el array de repositorios

5.9. Creación de la vista del listado de repositorios

La vista estará formada por un cuadro de texto para introducir el nombre usuario de GitHub, un botón para realizar la petición y la lista de repositorios del usuario. Tanto el cuadro de texto como la lista estarán vinculados a variables de instancia en su controlador.

```
<ion-header>
  <ion-navbar>
    <button ion-button menuToggle>
      <ion-icon name="menu"></ion-icon>
    </button>
    <ion-title>Repositorios</ion-title>
  </ion-navbar>
</ion-header>

<ion-content padding>
  <ion-list inset>
    <ion-item>
      <ion-input [(ngModel)]="username" type="text"
 placeholder="GitHub username"></ion-input> ❶
    </ion-item>
  </ion-list>
  <div padding>
    <button block (click)="getRepos()" ion-button>Obtener
 repositorios</button> ❷
  </div>

  <ion-list>
    <button ion-item *ngFor="let repo of repos"> ❸
      <h2>{{ repo.name }}</h2> ❹
      <p>{{ repo.description }}</p>

      <ion-icon name="arrow-forward" item-right></ion-icon> ❺
    </button>
  </ion-list>
</ion-content>
```

❶ Cuadro de texto ligado a la variable de instancia `username` del controlador mediante `ngModel`

❷ Botón con el controlador de evento asociado para la obtención de los repositorios. No se pasa el nombre de usuario ya que está almacenado en la variable de instancia del controlador

❸ Iteración sobre la lista de repositorios creando un botón para cada repositorio encontrado. La lista de repositorios `repos` es una variable de instancia del controlador

❹ Nombre y descripción del repositorio

❺ Botón de flecha para mostrar más adelante detalles del repositorio

5.10. Añadir una página de detalle del repositorio

Continuando con nuestra aplicación estamos interesados en mostrar detalles de un repositorio al seleccionarlo en la lista de resultados. Esto lo podemos hacer pasando a una página de detalle para mostrar los resultados. Seguiremos estos pasos:

1. Crear la página de detalle

```
ionic g page repo-details
```

2. Modificar el archivo `src/app/app.module.ts` añadiendo la página a la zona de importaciones y añadiendo la clase de la página a los arrays `declarations` y `entryPoints` de `@NgModule`.

```
...
import { RepoDetailsPage } from '../pages/repo-details/repo-details';
...

@NgModule({
  declarations: [
    ...
    RepoDetailsPage
  ],
  entryComponents: [
    ...
    RepoDetailsPage
  ],
  ...
})
...
```

3. Crear un método en el controlador de la página que hace la llamada (`src/app/pages/repos/repos.ts`) que se encargue de abrir la página de detalle. El método tomará como parámetro el repositorio del que se quiere mostrar la información detallada, por lo que habrá que importar la clase del modelo del repositorio. Para abrir la página de detalle, le pasaremos al NavController la página de detalle, por lo que también habrá que importarla.

Ejemplo 5.9. Modificación de `src/app/pages/repos/repos.ts` para abrir la pi#na de detalles

```
...
import { Repo } from '../../models/repo'; ❶
import { RepoDetailsPage } from '../repo-details/repo-
details'; ❷
...

  getDetails(repo: Repo) { ❸
    this.navCtrl.push(RepoDetailsPage, {repo: repo}); ❹
  }
```

❶ Importar el modelo del repositorio para poder pasar información a la página de detalle

❷ Importar la página de detalle a abrir

❸ Método que se encarga de abrir la página de detalle

❹ Abrir la página de detalle pasándosela a NavController junto con un parámetro `repo`. Los parámetros son pasados como una lista clave-valor

4. Obtener los parámetros en la página de detalle `src/app/pages/repo-details/repo-details.ts`

Ejemplo 5.10. El archivo `src/app/pages/repo-details/repo-details.ts`

```
import { Component } from '@angular/core';
import { IonicPage, NavController, NavParams } from 'ionic-
angular';

import { Repo } from '../../models/repo'; ❶

@IonicPage()
@Component({
  selector: 'page-repo-details',
  templateUrl: 'repo-details.html',
})
export class RepoDetailsPage {
  repo: Repo; ❷

  constructor(public navCtrl: NavController, public navParams:
 NavParams) {
    this.repo = navParams.get('repo'); ❸

  }

  ionViewDidLoad() {
    console.log('ionViewDidLoad RepoDetailsPage');
  }

}
```

❶ Importar el modelo del repositorio

❷ Variable de instancia para el repositorio recogido como parámetro. Esta variable nos permite poder manejar el repositorio desde la vista

❸ Recuperar el repositorio pasado como parámetro

5. Mostrar los detalles en la vista

Ejemplo 5.11. El archivo `src/app/pages/repo-details/repo-details.html`

```
<ion-header>

  <ion-navbar>
    <ion-title>Detalles</ion-title>
  </ion-navbar>

</ion-header>

<ion-content padding>
  <h2>Información del repositorio</h2>
  <div *ngIf="repo.description; else elseBlock"> ❶
    <p>{{repo.description}}</p> ❷
  </div>
  <ng-template #elseBlock>No hay descripción disponible</ng-
template> ❸
</ion-content>
```

❶ Uso del `If-then-else` de Angular para mostrar la descripción si existe

❷ Mostrar la descripción del repositorio

❸ Indicar que no hay información disponible