

UNIT 3

Ionic



Part 1 **Introduction. Setup.** **Navigation. Basic components.**

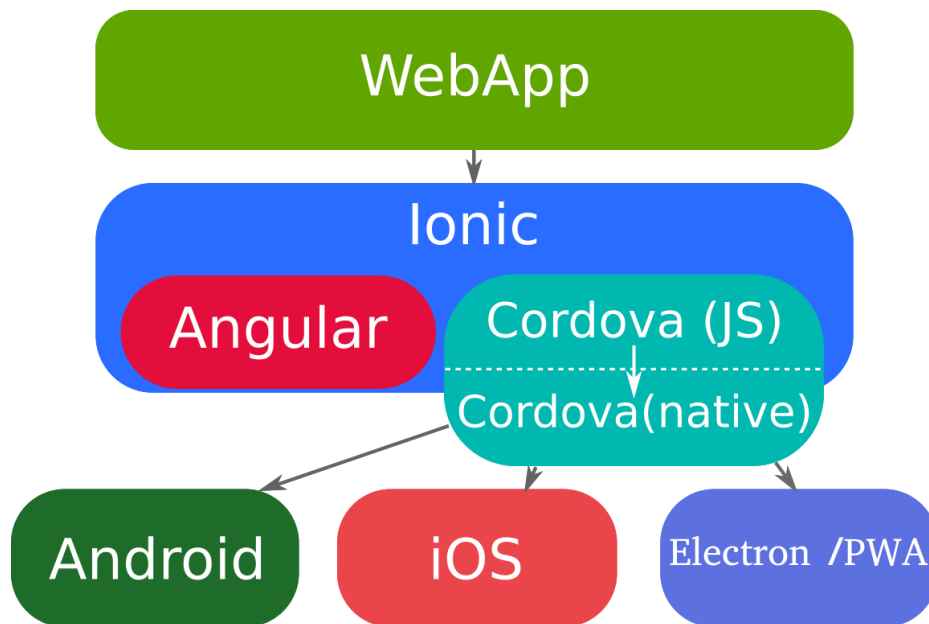
Client-side Web Development
2nd course – DAW
IES San Vicente 2020/2021
Author: Arturo Bernal Mayordomo

Index

Introduction.....	3
Installing and running Ionic.....	4
Testing in the browser.....	4
App structure.....	4
Basic components.....	5
Buttons.....	5
Icons.....	5
Lists.....	5
Inputs.....	6
Datetime.....	6
Cards.....	7
Checkbox.....	7
Radio.....	8
Toggle.....	8
Badges.....	9
Toasts.....	9
Alerts.....	9
Theming our app (styles).....	11
Navigation.....	12
Navigating forward and backwards (animations).....	12
Navigation with side menu.....	13
Navigation with tabs.....	14

Introduction

[Ionic](#) is a framework designed to build hybrid (cross-platform) **mobile apps** using web technologies. Ionic can be used in combination with Angular and [Cordova](#) (or [Capacitor](#)). It uses common web technologies like ES2015/Typescript, CSS/SASS and HTML. Being cross-platform means you only need to build the app once and it will work on different mobile OS (Android, iOS) desktop WebApps ([Electron](#)) or Progressive WebApps ([PWA](#)).



Using Cordova or Capacitor means we have a layer between our JavaScript (or TypeScript) code and the native API on the device that allows us to access directly the device's sensors, camera, vibration, and services (Contacts, Gallery, SMS, Google, Facebook, ...). We can extend that functionality using plugins.

We are not limited to developing mobile applications with Ionic. We can develop web pages that run on the browser, or Progressive Web Apps ([info](#), [more info](#)) that run everywhere (browser, desktop and mobile) and have access to native services like storage and camera.

Installing and running Ionic

Like we did in order to build and manage Angular projects, we're going to use a tool called [Ionic CLI](#) to do the same with Ionic projects. To install this tool execute:

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

If you want some help about the ionic command run "**ionic help**". To create a new project just execute ([more info](#)):

```
ionic start appName [blank|tabs|sidemenu]
```

- **blank** → creates an empty template.
- **tabs** → creates a template with tabbed navigation.
- **sidemenu** → Creates a template with side menu navigation.

Testing in the browser

To test your app in a browser, just type:

```
ionic serve
```

This command will start a new web server, usually in **localhost:8100**. As this framework is designed for mobile devices, it's recommended you switch your view to emulate a mobile device from the Developer Tools of your browser.



App structure

An Ionic app is just an Angular app, with a few more things (especially when we start working with Capacitor). These are the main differences:

- **www** → This is where the Ionic app is compiled (ionic build).
- **src/theme** → In the file **variables.scss** you can add or modify Ionic main theme (colors and other variables, fonts, etc.). Ionic apps are created to work with SASS by default.
- **ionic.config.json** → Ionic project's configuration (most of the configuration is done in angular.json).

```
> e2e
> node_modules
> src
> www
◆ .gitignore
{} angular.json
≡ browserslist
⦿ ionic.config.json
{} ionic.starter.json
K karma.conf.js
{} package-lock.json
{} package.json
{} tsconfig.app.json
TS tsconfig.json
{} tsconfig.spec.json
{} tslint.json
```

Basic components

In this section we'll review some basic [Ionic components](#) we can use to build a simple app. In the near future, we'll learn about more advanced components.

Buttons

Buttons can have multiple styles as you can see in the [documentation](#). A button in Ionic is created by placing the **ion-button** directive inside an element (which can be a `<button>` element, but it's not mandatory).

A button, like many components in Ionic, can have a color (use the color directive). The color's value must be defined in **src/theme/variables.scss** (\$colors).

```
<ion-button ion-button color="light">Light</ion-button>
<ion-button ion-button>Default</ion-button>
<ion-button ion-button color="secondary">Secondary</ion-button>
<ion-button ion-button color="danger">Danger</ion-button>
<ion-button ion-button color="dark">Dark</ion-button>
```



Other useful attributes are **fill** ("solid" → with background color, "clear" → transparent background, "outline" → transparent with border), or **expand** ("block" → full-width button, "full" → same as block, but with no left and right margins).

Icons

To place an icon inside a list item, button, etc. we use the `<ion-icon>` element. This element has a **name** attribute where we put which icon we want to show. You can see the full list of icons available [here](#). The attribute **slot** ("start", "end"), indicates where to align the icon inside a list item, button, etc.

Lists

We can create a list of items easily. Those lists can have simple items (single line of text) or complex items (image, multiple lines, sliding buttons, ...). You can check the official documentation with examples [here](#).

The main component of a list is `<ion-list>`. This component starts a new list. Inside this component we can have other components (items, headers, etc.):

<ion-item> → Represents a list item. It can have anything inside, usually a `<ion-label>`, but also `<ion-button>`, `<ion-avatar>`, `<ion-input>`, `<ion-radio>`, etc. However, for simple lists, we usually want a list item to behave like a button (if you click on it, we'll perform some action). [Examples](#).

<ion-item-group> → This component divides the list into groups of items. You can have an **<ion-item-divider>** at the beginning of each group to add a separation

item (like a header). [Documentation](#).

<ion-list-header> → This component represents a list header. It should be placed at the beginning of a list (if you want to add a header).

<ion-item-sliding> → Used for creating sliding items in a list. It combines a **<ion-item>** along one or two **<ion-item-options>** containing buttons (side="start" → slide to the right, side="end" → slide to the left). [Documentation](#).

Inputs

[Inputs](#) are the most used elements in forms. In Ionic, inputs are usually placed with their corresponding label inside a list item. You can put those inputs inside a form element and apply the corresponding validation that Angular supports.

```
<form #loginForm="ngForm" (ngSubmit)="login()">
  <ion-list>
    <ion-item>
      <ion-label position="floating">Email</ion-label>
      <ion-input type="email" name="email" [(ngModel)]="email" required>
    </ion-item>
    <ion-item>
      <ion-label position="floating">Password</ion-label>
      <ion-input type="password" name="password" [(ngModel)]="password" required>
    </ion-item>
  </ion-list>
  <p>
    <ion-button type="submit" color="primary" expand="block"
      [disabled]="loginForm.invalid">Login</ion-button>
  </p>
</form>
```

Email	Email
pepe@correo.es	pepe@correo.es
Password	Password
....
LOGIN	LOGIN

Datetime

The Ionic's [Datetime](#) component integrates its own date and time picker to easily select and display selected date, time, or both. It works similarly to the `<input type="datetime-local">` HTML element.

This component accepts several attributes/directives. For instance **displayFormat** defines how the date will be shown to the user, and **pickerFormat** defines how data will be presented in the date picker control (by default uses the same format as displayFormat's value).

Also, we can use **[(ngModel)]** directive to bind this control to a value. This value is not a Date object. Instead it has to be a string representing a date in the [ISO 8601 format](#) (ex: 1994-12-15T13:47 or 1994-12-15T13:47:20.789+5:00). This format is usually understood directly by several database systems.

```

<ion-list>
  <ion-item>
    <ion-label>Date</ion-label>
    <ion-datetime displayFormat="DD/MM/YYYY HH:mm"
      pickerFormat="DD MMMM YYYY HH mm"
      [(ngModel)]="myDate"></ion-datetime>
  </ion-item>
</ion-list>

```

Date					16/09/2012 09:12
			CANCEL	DONE	
14	July	2014	07	10	
15	Au...	2013	08	11	
16	Sep...	2012	09	12	
17	Oct...	2011	10	13	
18	Nov...	2010	11	14	

Also, we can add the **min** and **max** attributes, to limit the date we can choose. For example, to pick any date between 1st January 2010 and 31st December 2020:

```

<ion-datetime displayFormat="DD/MM/YYYY HH:mm" pickerFormat="DD MMMM YYYY HH mm"
  min="2010" max="2020-12-31" [(ngModel)]="myDate"></ion-datetime>

```

Cards

[Cards](#) are use to present important content in a design pattern that's becoming more and more used everyday. They can be seen as a more complex component than ion-item (you can put almost anything inside a card, a list for example).

You can watch what possibilities cards have in the official Ionic documentation. Inside a card you can place:

- **ion-card-header** → Card's header (top element, optional).
- **ion-card-content** → It adds a padding to the content we put inside.
- **ion-card-title** → Can be put inside ion-card-content. It makes the text bigger.
- **ion-image**, **ion-list**, **ion-item**, **ion-row**, **div**, **p**, ...

Checkbox

Ionic's [ion-checkbox](#) component works the same way as HTML's element `<input type="checkbox">`, but this component adapt's to each platform style and can have different colors (**color** attribute).

We can use the **disable** (boolean) attribute to disable the control and the **checked** (boolean) attribute to set the default value. And of course, we can use the `[(ngModel)]` directive.

```

<ion-list>
  <ion-item>
    <ion-label>Cheese</ion-label>
    <ion-checkbox disabled="true" checked="true" color="primary"></ion-checkbox>
  </ion-item>

  <ion-item>
    <ion-label>Sausage</ion-label>
    <ion-checkbox color="secondary" [(ngModel)]="sausage"></ion-checkbox>
  </ion-item>

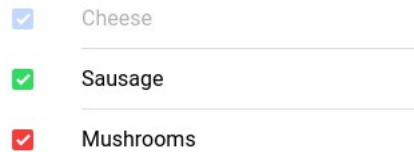
  <ion-item>

```

```

    <ion-label>Mushrooms</ion-label>
    <ion-checkbox color="danger" [(ngModel)]="mushrooms"></ion-checkbox>
  </ion-item>
</ion-list>

```



Radio

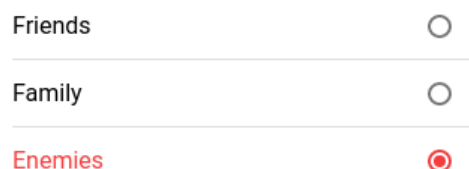
Radio buttons are presented in Ionic as list elements where you can only pick one of them. These elements (**ion-item**) should be placed inside a **radio-group** element (the **[(ngModel)]** directive should be placed here).

The [RadioButton](#) component is represented by the **ion-radio** component, and the corresponding **value** attribute should be in this element. It also can have other attributes like: **color** → Color set when selected, **checked** → checked by default, or **disabled** → can't be selected.

```

<ion-list>
  <ion-radio-group [(ngModel)]="relationship">
    <ion-item>
      <ion-label>Friends</ion-label>
      <ion-radio value="friends" checked color="primary"></ion-radio>
    </ion-item>
    <ion-item>
      <ion-label>Family</ion-label>
      <ion-radio value="family" color="secondary"></ion-radio>
    </ion-item>
    <ion-item>
      <ion-label>Enemies</ion-label>
      <ion-radio value="enemies" color="danger"></ion-radio>
    </ion-item>
  </ion-radio-group>
</ion-list>

```



Toggle

A [Toggle](#) component works the same way as a Checkbox.

```

<ion-list>
  <ion-item>
    <ion-label>Cheese</ion-label>
    <ion-toggle disabled="true" checked="true" color="primary"></ion-toggle>
  </ion-item>

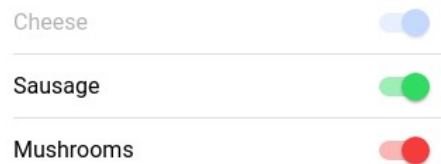
  <ion-item>
    <ion-label>Sausage</ion-label>
    <ion-toggle color="secondary" [(ngModel)]="sausage"></ion-toggle>
  </ion-item>

  <ion-item>
    <ion-label>Mushrooms</ion-label>
    <ion-toggle color="danger" [(ngModel)]="mushrooms"></ion-toggle>
  </ion-item>
</ion-list>

```




```
</ion-item>
</ion-list>
```



Badges

[Badges](#) are just small components used to highlight some value (usually numerical) to the user, like notifications. They can have different colors:

```
<div>
  <ion-badge color="secondary">10</ion-badge>
  <ion-badge color="primary">20</ion-badge>
  <ion-badge color="dark">30</ion-badge>
  <ion-badge color="danger">40</ion-badge>
</div>
```



A more complex version of Badges are [Chips](#).

```
<ion-chip color="dark">
  <ion-label>With Icon</ion-label>
  <ion-icon name="close-circle" color="danger" (click)="delete(chip)">
</ion-chip>
```



Toasts

Sometimes we just need to display a quick message to the user, for example to inform if an operation was successful or not. Also we want this message to be in an absolute position and for a period of time. The simplest way to do this is using [Toast](#).

To create toast messages, we'll need to inject Ionic's ToastController.

```
<ion-button (click)="showToast()">Show Toast</ion-button>

export class HomePage {
  constructor(public toastCtrl: ToastController) {}

  async showToast() {
    const toast = await this.toastCtrl.create({
      message: 'I\'m a toast message',
      duration: 2000, // 2 seconds
      position: 'middle',
      color: 'dark'
    });
    await toast.present();
  }
}
```



I'm a toast message

Alerts

[Alert](#) dialogs are a way to show the user more complex information inside a modal

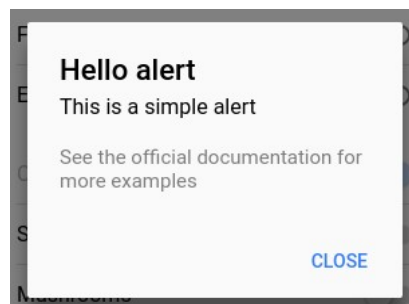
component. They're more complex than toasts but they work similarly. We need to use Ionic's `AlertController` to create Alert dialogs.

You can find more information and examples in the official documentation. These are the most common options used when creating an Alert dialog:

- **header** → Dialog's title
- **subHeader** → Text smaller and below the title.
- **message** → More subtle text.
- **inputs** → Array of input controls that this dialog can have (they can be text, radio, checkbox, number, ... inputs).
- **buttons** → Array of buttons this dialog will have at the bottom. A behavior and custom text can be added to each button.

```
<ion-button ion-button (tap)="showAlert()">Show Alert</ion-button>
```

```
export class HomePage {  
  ...  
  constructor(public toastCtrl: ToastController,  
               public alertCtrl: AlertController) {}  
  ...  
  showAlert() {  
    const alert = await this.alertCtrl.create({  
      header: 'Hello alert',  
      subHeader: 'This is a simple alert',  
      message: 'See the official documentation for more examples',  
      buttons: ['Close']  
    });  
    await alert.present();  
  }  
}
```



Theming our app (styles)

Ionic has [9 default colors](#) with variations of each one (shade and tint). All Ionic style properties are defined using [CSS variables](#). Using CSS variables instead of SASS variables makes possible to change their values dynamically in run-time.

To override the default colors you can use the [Color Generator](#) tool and copy the generated result in the **src/global.scss** file. To add a new color you define a CSS class selector named `.ion-color-{new color}` and set the corresponding values, including hexadecimal and RGB values of the **base** color and **contrast** color (opposite of the base color and visible against it), a **shade** color (slightly darker version) and **tint** color (slightly lighter version).

```
.ion-color-favorite {  
  --ion-color-base: #69bb7b;  
  --ion-color-base-rgb: 105,187,123;  
  --ion-color-contrast: #ffffff;  
  --ion-color-contrast-rgb: 255,255,255;  
  --ion-color-shade: #5ca56c;  
  --ion-color-tint: #78c288;  
}  
  
<ion-button color="twitter">Hello</ion-button>
```

Global CSS and SASS for your application should be placed inside **src/global.scss** file. Each component/page has also it's own SASS file, so you can edit the style only for that component there.

Navigation

From version 4, Ionic uses Angular router, with a little customization layer (uses **ion-router-outlet** instead of router-outlet). In Ionic, pages are Angular components which have the 'Page' suffix instead of 'Component', and are **lazy loaded** in the router (so a module is created for every page).

To create new pages with the Ionic CLI tool, just execute, for example:

```
ionic g page products
```

Ionic CLI commands work very similarly to the Angular CLI commands. In fact we can use directly Angular CLI for most of the work. This will generate a new component inside **src/app** representing a new page.

Navigating forward and backwards (animations)

To navigate to another page or route, you use the **[routerLink]** directive. However, Ionic adds another directive called **[routerDirection]** which defines the transition animation the new loaded route will have, and accepts values like **'root'** (no animation), **'forward'** (default) or **'backward'**.

```
<ion-item [routerDirection]=" 'forward' " [routerLink]="url">
  <ion-label>
    {{urlName}}
  </ion-label>
</ion-item>
```

If we want to navigate by code, we can use Angular **Router**, with the navigate method (forward animation), or use the equivalent in @ionic/angular, **NavController** service, with the different methods that add animations.

```
export class HomePage implements OnInit {

  constructor(public nav: NavController, public router: Router, public location:
Location) { }

  goRouter() {
    this.router.navigate(['/route']); // Forward animation
  }

  goNavDefault() {
    this.nav.navigate(['/route'], {}); // Forward animation
  }

  goNavForward() {
    this.nav.navigateForward(['/route']); // Forward animation
  }

  goNavBack() {
    this.nav.navigateBack(['/route']); // Backward animation
  }

  goNavRoot() {
    this.nav.navigateRoot(['/route']); // No animation
  }
}
```

Navigation with side menu

If we need an app with a side menu, this menu will be defined in the **AppComponent** template inside a [Menu](#) (ion-menu) element. A menu is like a page, has its header and content. And usually the content is a list with the pages you can directly navigate to.

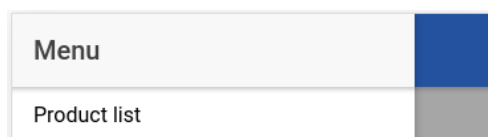
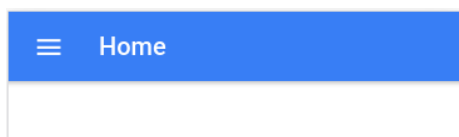
```
<ion-app>
  <ion-split-pane>
    <ion-menu [content]="content">
      <ion-header>
        <ion-toolbar>
          <ion-title>Menu</ion-title>
        </ion-toolbar>
      </ion-header>

      <ion-content>
        <ion-list>
          <ion-item [routerDirection]="root" [routerLink]="['/products']">
            <ion-label>Product list</ion-label>
          </ion-item>
        </ion-list>
      </ion-content>
    </ion-menu>
    <ion-router-outlet main></ion-router-outlet>
  </ion-split-pane>
</ion-app>
```

This menu is controlled from the **app.component.ts** file.

Usually, in a root or main page (pages that accessed directly from the menu), we want to place a [menu button](#) that displays the side menu. This is how we place this button in the header section:

```
<ion-header>
  <ion-toolbar>
    <ion-buttons slot="start">
      <ion-menu-button></ion-menu-button>
    </ion-buttons>
    <ion-title> Home </ion-title>
  </ion-toolbar>
</ion-header>
```



When we navigate to pages that derivate from a root page, like a product's details from the product list page, a [back button](#) should be added in the header that will go to the previous page.

```
<ion-header>
  <ion-toolbar>
    <ion-buttons slot="start">
      <ion-back-button></ion-back-button>
    </ion-buttons>
    <ion-title> Products </ion-title>
  </ion-toolbar>
</ion-header>
```



Navigation with tabs

When our application is based on tabs instead of a side menu, or we have a page that has tabs, navigation structure is slightly different. First of all, we have to define those tabs in a separate page.

```
<ion-tabs>
  <ion-tab-bar slot="bottom">
    <ion-tab-button tab="home">
      <ion-icon name="home"></ion-icon>
      <ion-label>Home</ion-label>
    </ion-tab-button>

    <ion-tab-button tab="about">
      <ion-icon name="information-circle"></ion-icon>
      <ion-label>About</ion-label>
    </ion-tab-button>

    <ion-tab-button tab="contact">
      <ion-icon name="contacts"></ion-icon>
      <ion-label>Contacts</ion-label>
    </ion-tab-button>
  </ion-tab-bar>
</ion-tabs>
```

The **tab** attribute indicates which route is going to show when we click (tap) the corresponding tab. Lets see how we create the tab routes for those pages:

```
const routes: Routes = [
  {
    path: '',
    component: TabsPage,
    children: [
      {
        path: 'home',
        loadChildren: '../home/home.module#HomePageModule'
      },
      {
        path: 'about',
        loadChildren: '../about/about.module#AboutPageModule'
      },
      {
        path: 'contact',
        loadChildren: '../contact/contact.module#ContactPageModule'
      },
      {
        path: '',
        redirectTo: '/tabs/home',
        pathMatch: 'full'
      }
    ]
  }
];
```

As you can see, the tabs component (in this case TabsPage) is loaded always, and depending the route (/home, /about, /contact), the corresponding page is loaded. All pages in Ionic by default have they own module and are **lazy loaded** for maximum efficiency.

First of all, we'll create the necessary pages and put the tabs in the ProductDetail page. Those tabs will be connected to ProductInfo and ProductComments pages.

```
# product-detail.html
```

```

<ion-header>
  <ion-toolbar>
    <ion-buttons slot="start">
      <ion-back-button defaultHref="/home"></ion-back-button>
    </ion-buttons>
    <ion-title>
      Product details
    </ion-title>
  </ion-toolbar>
</ion-header>

<ion-tabs>
  <ion-tab-bar slot="bottom">
    <ion-tab-button tab="info">
      <ion-icon name="information-circle"></ion-icon>
      <ion-label>Information</ion-label>
    </ion-tab-button>

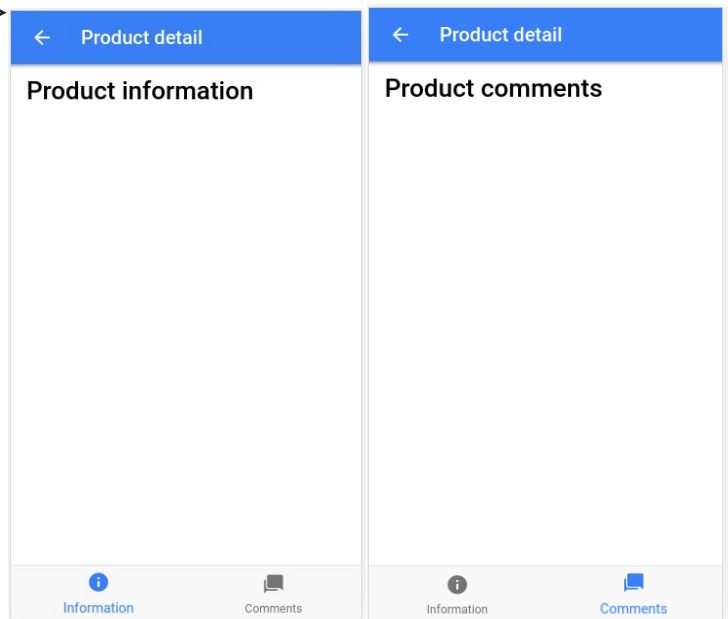
    <ion-tab-button tab="comments">
      <ion-icon name="chatbubbles"></ion-icon>
      <ion-label>Comments</ion-label>
    </ion-tab-button>
  </ion-tab-bar>
</ion-tabs>

# product-info.html
<ion-content padding>
  <h1>Product information</h1>
</ion-content>

# product-comments.html
<ion-content padding>
  <h1>Product comments</h1>
</ion-content>

# details.router.module.ts
const routes: Routes = [{
  path: '',
  component: ProductDetailPage,
  children: [
    {
      path: 'info',
      loadChildren: '../product-info/product-info.module#ProductInfoModule'
    },
    {
      path: 'comments',
      loadChildren: '../product-comments/product-comments.module#ProductCommentsModule'
    },
    {
      path: '',
      redirectTo: '/info',
      pathMatch: 'full'
    }
  ]
}];
@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule]
})
export class TabsPageRoutingModule {}

```



We don't need to create a header for ProductInformation and ProductComments because the header in ProductDetail is going to be on top of them. We could instead, create a custom ion-header in every other page, but not in the tabs page.