GESTIÓN DE CALIDAD SOFTWARE

Conceptos de Introducción

Autor: Santiago Meliá. DLSI. UA

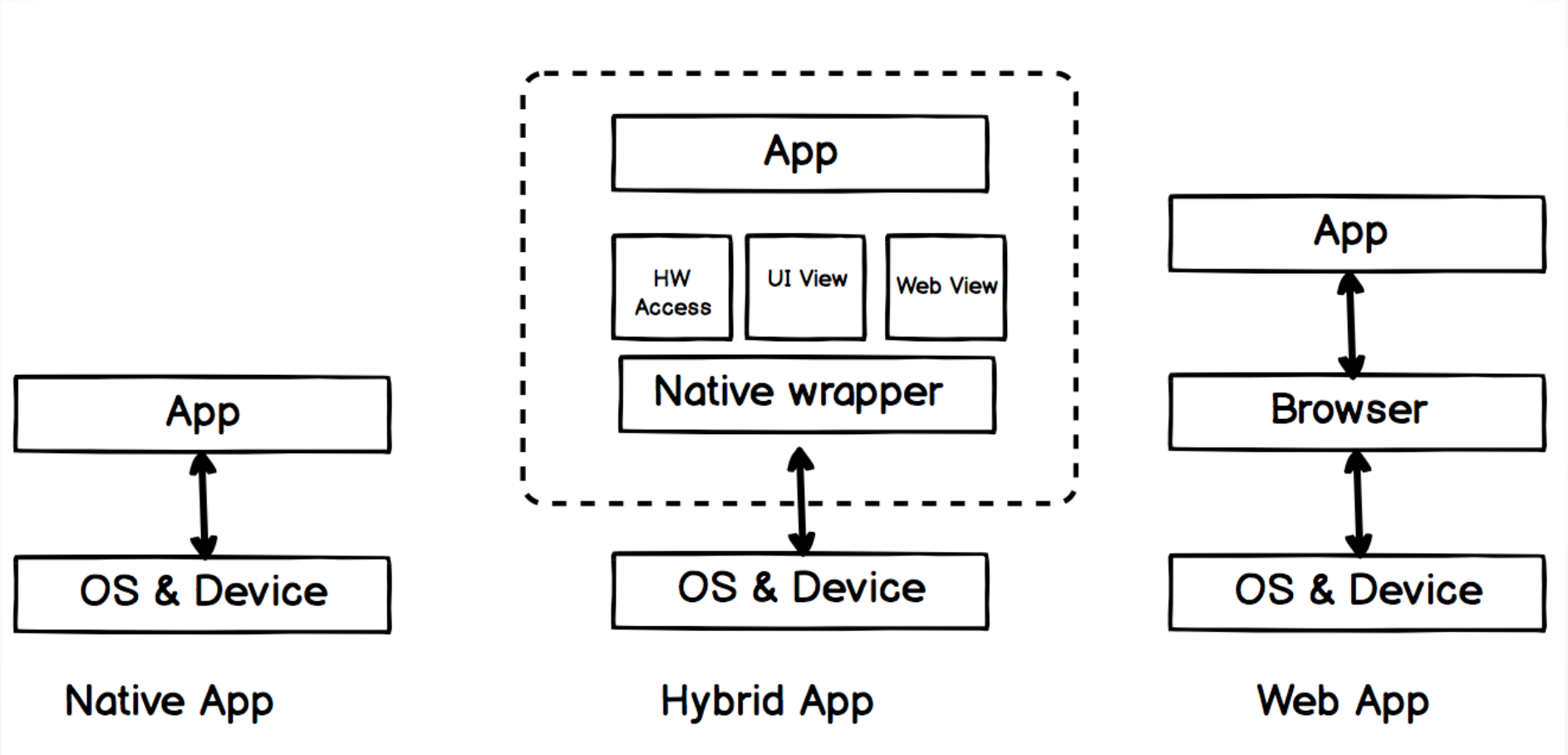
Índice

- Introducción a Ionic 4
- Carácterísticas principales de Ionic 4
- Arquitectura de Ionic 4 con Angular 7
- Patrón MVC y los elementos de una página en Ionic
- Instalación del entorno de desarrollo
- Uso de ionic CLI crear y ejecutar la App
- Creación de la primera App

¿Qué es Ionic?

- Ionic framework es un conjunto de herramientas de Interfaz de usuario de código abierto para desarrollar aplicaciones multiplataforma mediante tecnologías Web
- Permite desarrollar para iOS, Android, Web con las PWA (Progressive Web Application) y escritorio (Electrón)
- Ionic 4 está integrado de forma oficial con Angular 7, aunque está en desarrollo su integración con React y Vue

Aplicaciones Híbridas vs Nativas



¿Qué es un Web View?

- Un Web View es un navegador Web a pantalla completa que se ejecuta en los dispositivos móviles o escritorio
- Ionic accede a los API Hardware mediante un puente (bridge layer) normalmente javascript, concretamente pueden usarse 2 en ionic:
  - Cordova
  - Capacitor
- El equipo de Ionic ha desarrollado Capacitor permitiendo no depender de terceros, pero no es tan completo como Cordova, lo que hace que se usen ambos

Ionic en cifras

- +5 Millones de Apps creadas
- +30 mil preguntas en StackOverflow
- +100 Meetups
- +5 Millones de desarrolladores

Características principales de Ionic

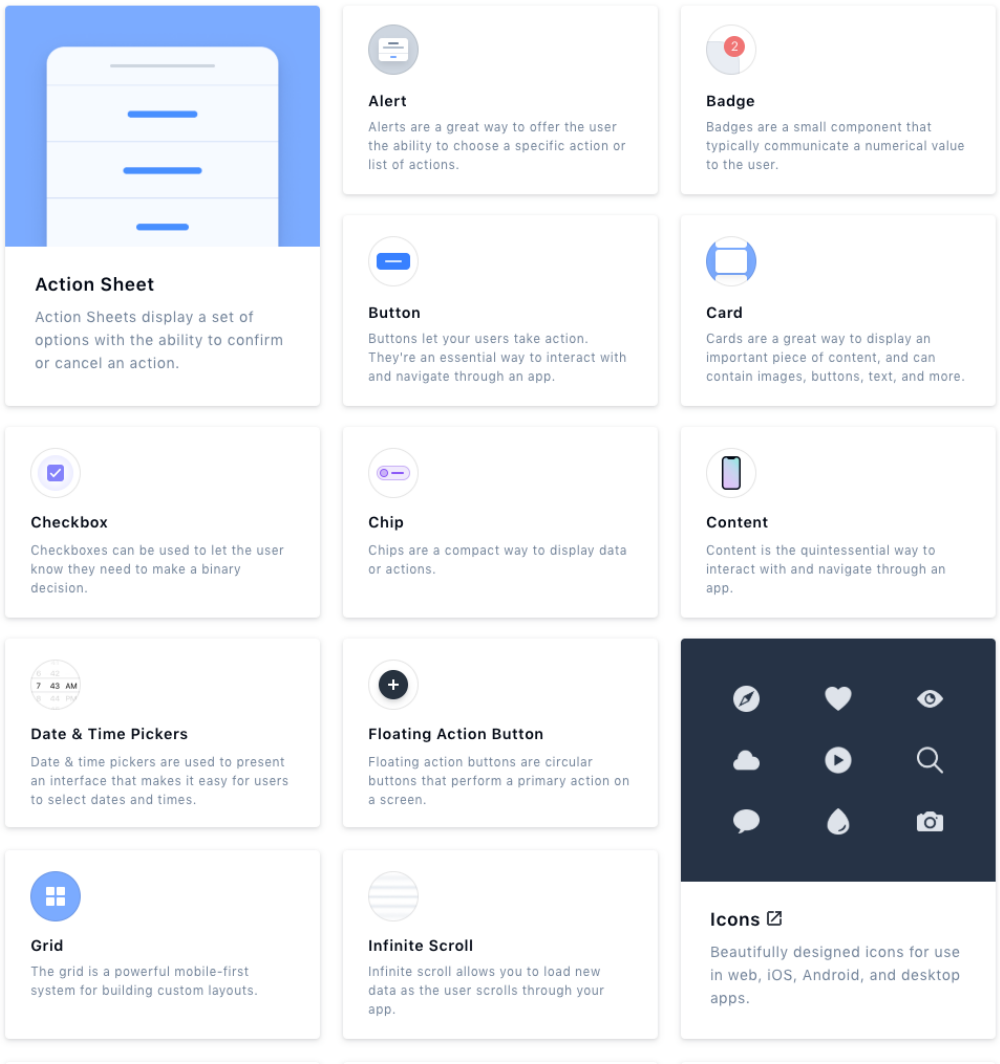
- **Ionic UI Components** – proporciona una librería de componentes de IU con javascript nativo
- **Platform Continuity** – proporciona una solución multiplataforma en su apariencia y funcionalidad
- **Navegación paralela** para dispositivos móviles
- **Acceso nativo** a los elementos del móvil
- **Tematización** para dar una apariencia común a toda la app

UI Components

- Los componentes de interfaz de usuario de Ionic son elementos reusables que sirven para construir los bloques de una aplicación
- Su implementación está basada en los estándares Web: HTML, CSS y javascript
- Aunque los componentes están pre-construidos y diseñados desde el inicio, pueden ser customizados para que cada aplicación tenga su propia apariencia
- Los componentes corren nativamente en el navegador sin librerías de terceros (Stencil)

UI Components

- Ionic proporciona todos los componentes necesarios para definir la interfaz de una aplicación móvil



Multiplataforma (Platform Continuity)

- Es una característica incorporada por Ionic que permite a los desarrolladores usar el mismo código para múltiples plataformas
- Cada componente adapta su apariencia a la plataforma en la que se ejecuta
- En los dispositivo Apple (iPhone y iPad) usan el propio **iOS design language**
- En los dispositivos Android usan el lenguaje de diseño de google **Material design**
- En las aplicaciones Web (Progressive Web App) proporcionan el tema Material Design por defecto, aunque es configurable

Navegación

- Tradicionalmente las aplicaciones Web usan una historia lineal. Solo navegan hacia delante y hacia atrás (Back button)
- Las aplicaciones móviles usan una navegación paralela “no lineal”. P.e. Una interfaz móvil de tabs puede tener una pila de navegación diferente por cada tab
- Ionic proporciona una navegación de historias paralelas que pueden ser anidadas por separado
- Ionic 4 recomienda utilizar Angular Router (Ver la doc de angular.io)

Acceso Nativo

- Tanto iOS como Android proporcionan kits de desarrollo software (SDKs) que permiten renderizar cualquier Ionic App, mientras permiten un completo acceso a sus librerías Nativas
- Proyectos como Capacitor (de Ionic) o Cordova (de Apache) son los más utilizados para dar acceso a las librerías nativas
- Permiten así a acceder a las características del dispositivo como a su acelerómetro, cámara, GPS, etc.

Tematización (Theming)

- Ionic es construido con CSS, el cual viene con los estilos predefinidos para iOS y Android, lo que permite que sea extremadamente fácil modificarlos
- Cada plataforma tiene un “mode” por defecto, se denominan: ios, md (material design) y core
- Viene con 9 colores predefinidos que puede cambiar el color de muchos componentes.
- Cada color es un colección de múltiples propiedades, las cuales se pueden cambiar con la herramienta Color Generator o manualmente si se desea

Principales ventajas de Ionic

- **Multiplataforma** – móvil, Web y escritorio
- **Reducida curva de aprendizaje** - Basado en conocidos estándares de la Web como HTML, CSS y Javascript
- **Diseño limpio, simple y funcional** (Material Design en Android y iOS design language)
- **Rápido desarrollo** mediante su componentización y el uso de scaffolding
- **Código Abierto** – el framework es gratis. Solo monetizan en ionicpro para hacer deploy y monitorización

Ecosistema del framework Ionic

Arquitectura de @ionic/Angular

- Tomamos la integración de Ionic 4 con Angular 7 al ser el framework SPA más utilizado. Está implementado con HTML y TypeScript
- Podemos identificar los siguentes bloques de construcción de una aplicación @ionic/angular
  1. Módulos o NgModules
  2. Componentes
  3. Páginas
  4. Templates
  5. Data binding
  6. Directivas
  7. Servicios
  8. Dependency Injection

Módulos

- Son los bloques básicos para construir una aplicación @ionic/angular
- Un módulo o NgModule define un contexto de compilación para uno o más componentes
- Un NgModule permite asociar a sus componentes con código relacionado, como p.e. servicios, para formar unidades funcionales
- El sistema de módulos de Angular es diferente e independiente al sistema de módulos de Javascript. De hecho se debe aprender a utililizar ambos sistemas modulares juntos

Módulo raiz o AppModule

- Cada aplicación generada con ionic/Angular cuenta con un *root module* o modulo de raíz llamado **AppModule**, el cual provee el mecanismo de arranque que inicia nuestra aplicación
- Una aplicación generalmente contiene varios módulos
- Si se desea usar otro feature module, se necesita registrarlo dentro de este fichero

```
@NgModule({
  declarations: [AppComponent],
  entryComponents: [],
  imports: [BrowserModule, IonicModule.forRoot(), AppRoutingModule],
  providers: [
    StatusBar,
    SplashScreen,
    { provide: RouteReuseStrategy, useClass: IonicRouteStrategy }
  ],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

Lazy loading

- Un módulo puede importar funcionalidades de otros módulos o exportar sus propias funcionalidades a otros
- Una buena práctica es crear diferentes módulos para tu aplicación y organizar tu código de manera funcional, para poder crear aplicaciones escalables cuando son complejas
- A cada uno de los modulos definidos para particionar la funcionalidad de la aplicación son llamados **feature modules**
- **Lazy loading** consiste en optimizar la carga, solo cargando aquellos feature modules que sean necesarios y así mejorar el rendimiento de nuestra aplicación

Componentes

- Un componente es una clase con el decorador @Component que provee la meta-información (metadata) necesaria para que Angular pueda usarlo
- Cada componente define una clase que contiene datos y lógica, y está vinculada a un template HTML

```
import { Component, Input } from '@angular/core'

@Component({
  selector:'app-card-list',
  templateUrl: './card-list.component.html'
})

export class CardListComponent {
  @Input () items: any[] = [];
  @Input () listName: string;
  @Input () navigateTo: any;
}
```

Define una etiqueta que permite reutilizarse en diferentes partes de la app
<app-card-list> </app-card-list>

El Template HTML (View)
<ion-list *ngIf="items.length > 0">
<app-card-list> </app-card-list>
</ion-label> </ion-item></ion-list>
….

Root Componente o AppComponent

- Cada aplicación de ionic/angular tiene al menos un componente el root component o AppComponent
- El ***root component*** conecta la jerarquía de componentes con el DOM en el fichero index.html, que contiene la etiqueta <app-root> </app-root> que inicia la app

```
@Component({
  selector: 'app-root',
  templateUrl: 'app.component.html'
})
export class AppComponent {
  constructor(
    private platform: Platform,
    private splashScreen: SplashScreen,
    private statusBar: StatusBar
  ) {
    this.initializeApp();
  }
```

Páginas

- Cada página o Page se define como un módulo que contiene una referencia a un componente
- El modulo referencia los recursos que requiere la página para su funcionamiento (lazy loading)
- El componente a su vez referencia a la parte estática (HTML y SCSS) de la vista
- El propio componente define el comportamiento dinámico de la vista de la página

(para ver con detalle ir a la pag. 31)

Templates

- Representa al elemento vista del patron MVC de ionic/angular
- Solo contiene un fragmento HTML del componente o página que representa
- Casi toda la sintaxis HTML es permitida a excepción de <script>, que es ignorado y aparece un warning en la consola del navegador
- Elementos como html, body o base solo residen en el index.html inicial
- Se puede extender el vocabulario HTML con componentes y directivas por del desarrollador

Directivas

- Las directivas o structural directives son responsables del HTML layout
- Las directivas moldean la estructura DOM, añadiendo, borrando o modificando elementos
- Se definen mediante un asterisco (*):

```
<div *ngIf="hero" class="name">{{hero.name}}</div>
```

- Las directivas más comunes son ngIf, ngFor y ngSwitch
- Angular permite definir nuestras propias directivas para nuestra app

Data Binding

- Hay dos tipos de data-binding:
  - *Event binding* o enlace de eventos, que responden a la interacción del usuario al modificar algún input en la aplicación, actualizando los datos en el componente
  - *Property binding* o enlace de propiedades, que permite agregar valores modificados desde nuestro componente a nuestro HTML

- Antes de que se muestre una vista, Angular evalúa las directivas y resuelve la sintaxis del data binding en el template para modificar los elementos HTML y el DOM, según los datos y la lógica de nuestra aplicación.

- Angular cuenta con ***two-way data binding***, que significa que los cambios en el DOM también se reflejan en nuestros datos y viceversa

Data Binding

- Aquí se muestran las 4 formas de Data Binding

De Componente al DOM. Denominada interpolación en su interior se introduce una expresión que resuelve angular

```
<li>{{hero.name}}</li>
```

Del Componente al DOM (se usa el binding de propiedades). A la derecha de la propiedad de corchetes se pone la expresión angular entre comillada

```
<app-hero-detail [hero]="selectedHero"></app-hero-detail>
```

Del DOM al Componente. A partir de un evento el DOM notifica al componente para que se ejecute una operación

```
<li (click)="selectHero(hero)"></li>
```

Enlace de doble dirección. Sirve para modificar la propiedad del componente desde el cambio en la interfaz. O para actualizar la interfaz cuando cambia su valor en el componente

```
<input [(ngModel)]="hero.name">
```

Servicios

- Un servicio es normalmente una clase con un propósito bien definido asociado a la lógica de negocio de la app

- Angular separa a los componentes de los servicios por razones de modularidad y reuso. Hace que los componentes sean más eficientes y ligeros

- El componente puede delegar en el servicio ciertas tareas como traer los datos del servidor, validar la entrada del usuario, etc.

Inyección de dependencia

- Ionic/Angular usa la inyección de dependencia para que el componente use los servicios. Así que los servicios son clases inyectables (@Injectable())

```
@Injectable ()

export class AlertService {

  constructor(public alertController: AlertController) {}

  async presentAlert(message: string) {
    const alert = await this.alertController.create({
      header: 'Alert',
      message,
      buttons: ['OK']
    });

    await alert.present();
  }
}
```

Inyección de dependencia

- Se inyecta por en el constructor el servicio para luego ser utilizado en el componente

```
export class CardDetailPage {


  private cardId: string;

  card: Card;


  constructor(private route: ActivatedRoute,
    private cardService: CardService,
    private loaderService: LoaderService,
    private alertService: AlertService) {}

   this.alertService.presentAlert('Connection Error, please reload the page');
```

Inyección de dependencia de los servicios

Modelo Vista Controlador

- Ionic basado en Angular 7 tiene una arquitectura basada en componentes donde se disponen los roles de Vista al template o plantilla y Controlador o ViewModel al componente
- Las propiedades del componente son disponibles por tu plantilla y así, la plantilla se actualizará automáticamente cualquier cambio de una dichas propiedades
- El modelo es representado por los servicios que se encargan de almacenar y recuperar los datos, ya sean locales o remotos

Estructura de la página

- Por cada una de las páginas se crea una subcarpeta dentro de la carpeta App con los siguientes ficheros:
- **nombre.page.html** – (Vista) plantilla HTML con componentes de ionic y propios que definen la vista
- **nombre.page.ts** – (Controlador) componente que tiene la parte dinámica de la página mediante una clase tipo componente que define la metainformación de la página y el binding
- **nombre.module.ts** – define su feature module para cargar solo los recursos que necesita dicha página
- **nombre.page.scss** – (Vista) definimos los estilos específicos para esta página
- **nombre.page.spec.ts** – contiene las pruebas unitarias sobre la página

Modulo de la página

- Nombre.module.ts – Define un feature module que carga los recursos necesarios para la página

```
const routes: Routes = [
  {
    path: '',
    component: AlterPage
  }
];

@NgModule({
  imports: [
    CommonModule,
    FormsModule,
    IonicModule,
    RouterModule.forChild(routes)
  ],
  declarations: [AlterPage]
})
export class AlterPageModule {}
```

Controlador de la página

- nombre.page.ts – define la metainformación que referencia a la parte estática, y contiene la clase que representa al controlador con los aspectos dinámicos de la IU

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-alter',
  templateUrl: './alter.page.html',
  styleUrls: ['./alter.page.scss'],
})
export class AlterPage implements OnInit {

  constructor() { }

  ngOnInit() {
  }

}
```

Vista de la página

- TemplateURL – contiene la fracción de HTML que representa la página actual
- Se utilizan los UI components de ionic definidos en Stencil, o en el propio proyecto

```
<ion-header>
  <ion-toolbar>
    <ion-title>alter</ion-title>
  </ion-toolbar>
</ion-header>

<ion-content padding>


</ion-content>
```

Estilo de la página

- nombre.page.scss – (Sassy CSS) introduce o modifica los estilos que sean específicos de la página. Permite bucles y condicionales

```
ion-content {
    padding-bottom: 64px;
    --background: var(--ion-color-light);
}

ion-item {
    border-radius: 0;
    border-bottom: 1px dotted var(--ion-color-light);
}

ion-card {
    border-radius: 0;
}
```

Instalación de Ionic CLI

- Las aplicaciones de Ionic son creadas y desarrolladas mediante líneas de comando
- Primero debemos instalar node.js con su gestor de paquetes npm, en una versión superior a la 8. Actualmente se recomienda la 10.15

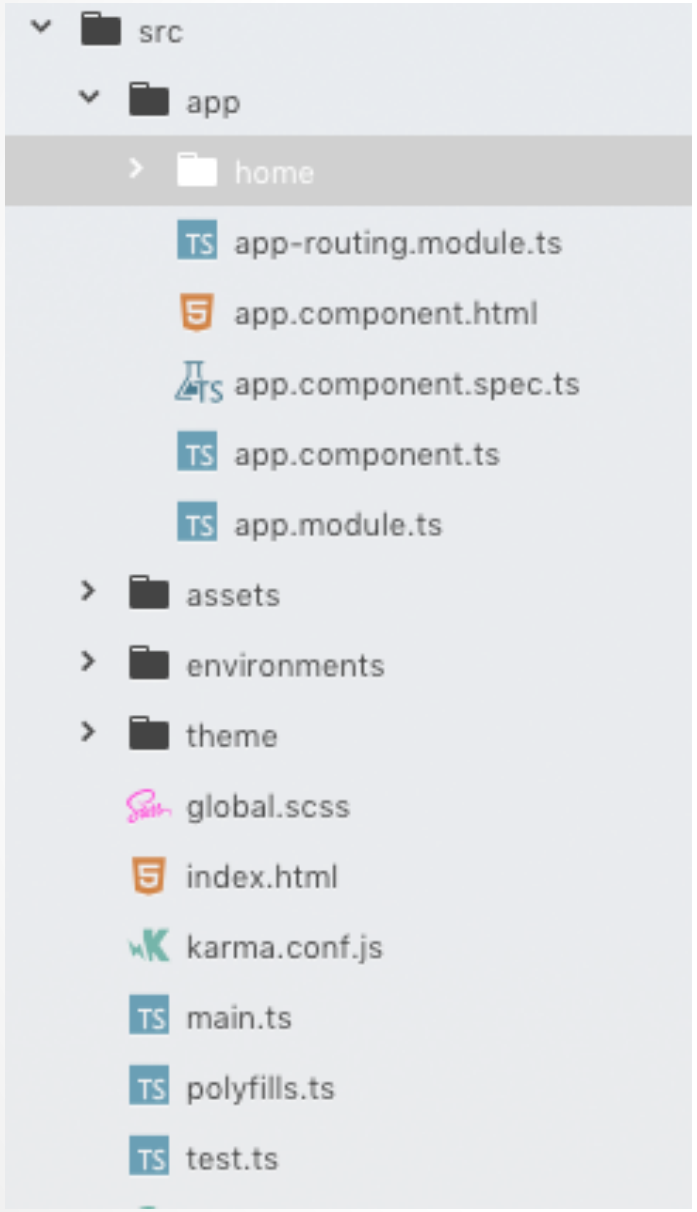
- Instalamos Ionic CLI con: **npm install –g ionic**

Creación de una aplicación Ionic

- Existen varias plantillas que por defecto te crean toda la estructura de archivos necesaria para crear una app
- Las plantillas más comunes son:
  - **Blank** – una app con solo una página Home vacía
  - **Tabs** – crea una app con un menú de tabs y tres páginas
  - **Sidemenu** – crea una app con un menú lateral

- Comando: **ionic start <nombreApp> <plantilla>**
- Se puede ver la lista de todas las plantillas con:
  - **Ionic start --list**

Estructura del proyecto



- En la carpeta Src es donde se sitúan los componentes de ionic/angular
- En src/ tiene el archivo index.html que inicia la aplicación con la referencia al componente root
- En src/app/ con tiene el root component y root module
- En src/app se crearán las carpetas para contener las Páginas, Servicios , Componentes, etc.

Ejecutar una aplicación Ionic

- Podemos ejecutar nuestra aplicación Ionic sobre node.js con la versión browser:

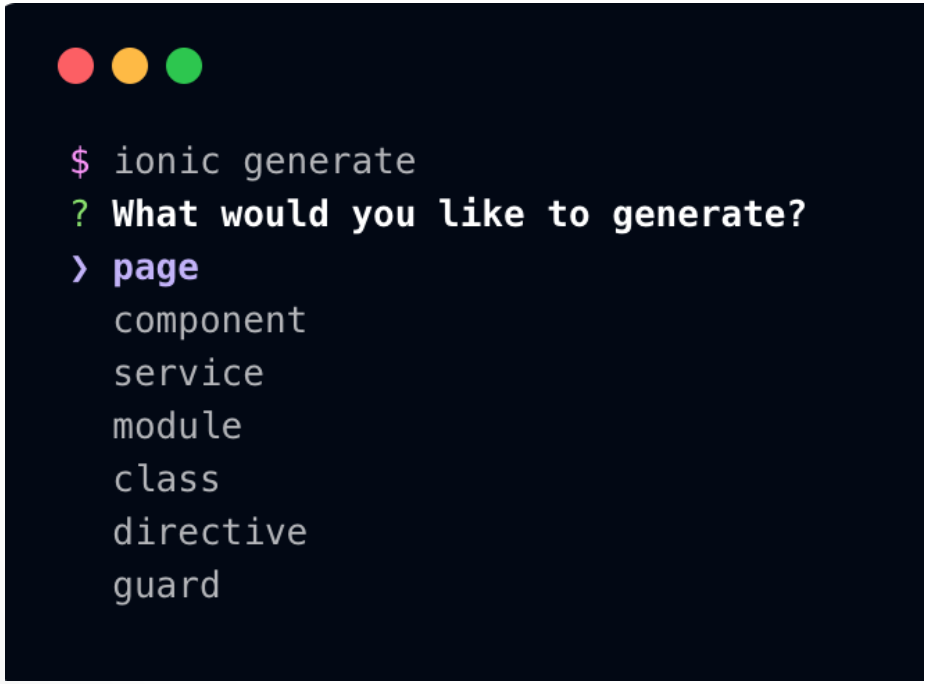
- Sentencia**: ionic serve**

- Si queremos comparar la aplicación y verla en iOS y Android simultáneamente podeis usar el @ionic/lab:

- Sentencia: **ionic serve --lab**

Generación de elementos de ionic con Scaffolding

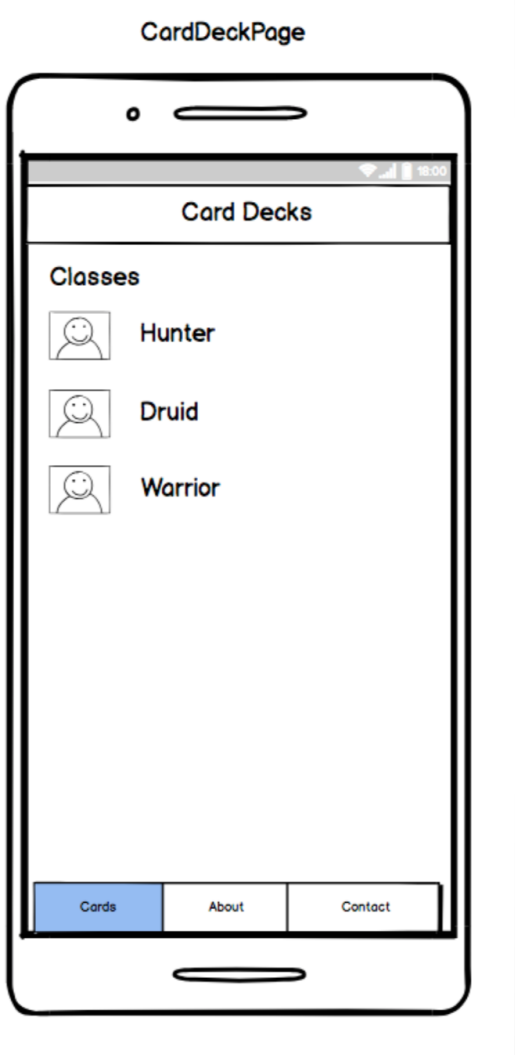
- Utilizando ionic CLI pueden generarse diferentes elementos de ionic
- La sentencia: **ionic generate <tipo> <nombre>**
- Si ponemos solo **ionic generate** nos sale un menú donde nos va a ir preguntando por cada parámetro (ver figura)
- Ionic CLI usa por debajo Angular CLI para generar dichos elementos



- Cuando se crea una page se actualiza automáticamente el enrutador de la app (router) para así no tener que introducirlo manualmente

1ª Aplicación Ionic: Entrega

- Vamos a realizar la primera aplicación HeartStoneApp
- Es una aplicación para jugar con cartas de rol
- En la primera página se muestra un menú de tabs que carga la página Cards
- Muestra una lista con una imagen y un texto de cada clase de mazos de cartas (Card-Decks).



Guíon de la creación de la aplicación HeartStoneApp

1. Creamos una aplicación con la plantilla tab (ver pag. 37)
2. A continuación creamos una página con el scaffolding de ionic que se denomine Card-decks (ver 40)
3. Se deber cambiar el enrutamiento del menú de tabs para que el primer icono de la izquierda apunte a la página card-decks (ver pag. 44 y 45)
4. Introducimos un componente <ion-list> para que contenga una serie de componentes <ion-item>. (ver pag. 46)
5. Introducimos a el <ion-list> una cabecera <ion-list-header> que ponga el texto ’Classes’ (ver 47)

Guíon de la creación de la aplicación HeartStoneApp

6. En la clase card-deck.page.ts definimos un array de strings con una lista de mazos:

readonly cardDecks:string [] = [‘Druid’, ‘Mage’, ‘Warrior’, ‘Rogue’, ’Shaman’];

7. Usamos un *ngFor=“let cardDeck of cardDecks” para recorrer dicho array y mostrar la lista de mazos con el nombre y su imagen (ver. 47)
8. Descomprimir el fichero images.zip (en moodle) para que las imágenes de los mazos queden en la carpeta ‘src/assets/images’

Modificación del menú de tabs (vista)

- Cambiamos el enrutamiento del menú de tabs para que en lugar de apuntar a tabs1 cargue la página card-decks
- Cambiamos el la cadena Routes en dos puntos:
  - En la lista de hijos (Children)
  - Cambiar la ruta Path:'tab1' debe indicar ahora 'cards'
  - En la carga inicial con Path:'' se ha de indicar que redirija a '/tabs/cards

```
children: [
  {
    path: 'cards',
    children: [
      {
        path: '',
        loadChildren: '../card-decks/card-decks.module#CardDecksPageModule'
      }
    ]
  },
```

…el último path (por defecto)

```
path: '',
redirectTo: '/tabs/cards',
pathMatch: 'full'
```

Modificación del menú de tabs (vista)

- Cambiamos la vista del menú de tabs para que el enlace apunte a cards y aparezca debajo del icono la etiqueta CardDecks

```
<ion-tab-bar slot="bottom">
  <ion-tab-button tab="cards">
    <ion-icon name="flash"></ion-icon>
    <ion-label>CardDecks</ion-label>
  </ion-tab-button>

  <ion-tab-button tab="tab2">
    <ion-icon name="apps"></ion-icon>
    <ion-label>Tab Two</ion-label>
  </ion-tab-button>
```

Creación de la lista

- Se Introduce un elemento lista <ion-list> que contiene una lista de items <ion-item>. Cada uno con una imagen <ion-avatar> y una etiqueta <ion-label>

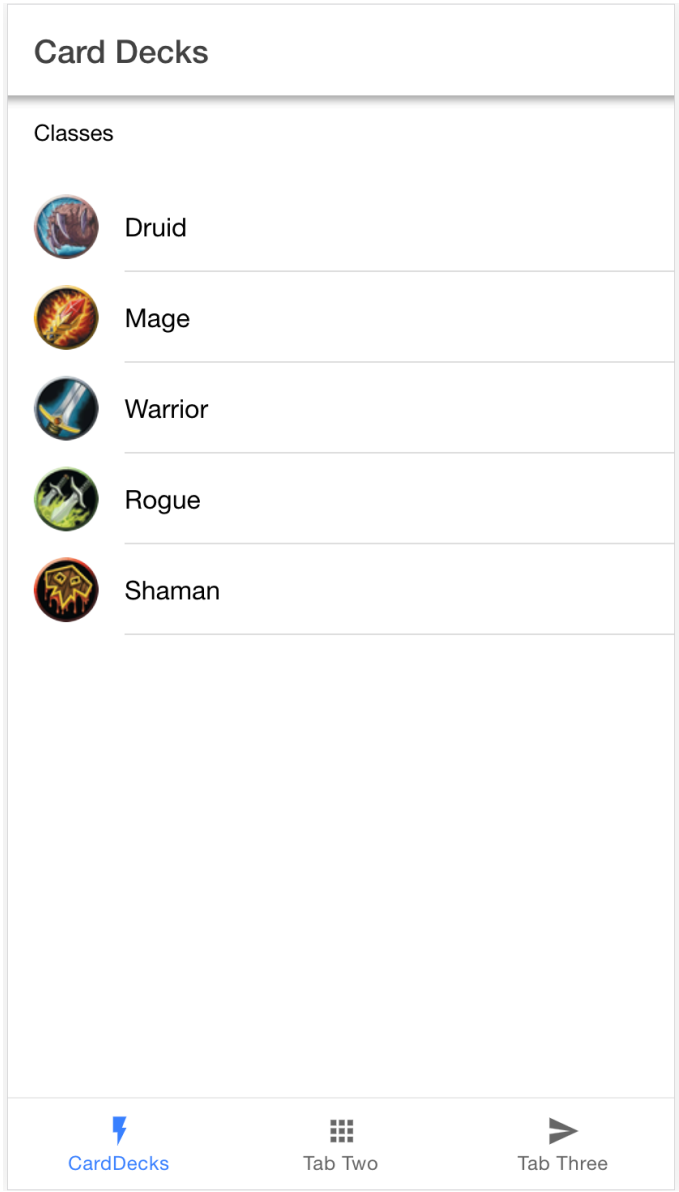
```
<ion-list>
    <ion-item>
      <ion-avatar slot="start">
        <img src="../assets/images/avatar.png"/>
      </ion-avatar>
      <ion-label>
        <h2> Nombre del Avatar </h2>
      </ion-label>
   </ion-item>
</ion-list>
```

Creación de la lista con la carga de datos

- Se utiliza la directiva ngForm para recorrer el array cardDecks
- Se utiliza la interpolación para asignar el nombre de la imagen y el nombre de la etiqueta de cada cardDeck

```
<ion-list-header>
  <ion-label>Classes</ion-label>
</ion-list-header>
<ion-list>
    <ion-item *ngFor="let cardDeck of cardDecks">
      <ion-avatar slot="start">
        <img [src]="'../../assets/image/' + cardDeck + '.png'"/>
      </ion-avatar>
      <ion-label>
        <h2> {{cardDeck}} </h2>
      </ion-label>
   </ion-item>
</ion-list>
```

Solución final



Documentación

- Ionic:
  - https://ionicframework.com/docs
- Angular:
  - https://angular.io/docs