GESTIÓN DE CALIDAD SOFTWARE

Aspectos avanzados: Storage, Menú, Maps y Theming

Autor: Santiago Meliá. DLSI. UA

Índice

- Ionic Storage
  - Introducción
  - Uso del Storage en una App
  - Almacenamientos Clave-Valor y objetos JSON
- Back button
- Angular Google Maps (AGM)
  - AgmMaps, AgmMarker, AgmWindowInfo
- Menú
  - Ion-menu, ion-menu-toggle, MenuController, ion-menu-button
- Theming
  - Modo y Plataforma
  - Variables CSS
  - Definir temas

Ionic Storage

- Storage es una forma sencilla de almacenar parejas clave – valor y objetos JSON en el dispositivo cliente
- Storage usa una variedad de motores de almacenamiento y elige la mejor disponible en función de la plataforma
- Cuando se ejecuta en una App nativa prioriza el uso de SQLLite, como una de las BBDD más estables y disponibles
- Cuando se ejecuta en Web o PWA, Storage hace uso de indexedDB, WebSQL y localstorage en ese orden

Instalar Storage en el proyecto

- Para poder utilizarlo instalamos el plugin cordova-sqllite-storage, en el raíz del proyecto en desarrollo
  - **Ionic cordova plugin add cordova-sqlite-storage**
- A continuación instalamos el paquete de ionic/storage
  - **npm install –save @ionic/storage**

Importar el Ionic Storage

- Lo primero es importar el modulo IonicStorageModule a nivel del módulo raiz app.module
- Se utiliza el método forRoot en su importación

```
import { IonicStorageModule } from '@ionic/storage';

@NgModule({
  declarations: [
    // ...
  ],
  imports: [
    BrowserModule,
    IonicModule.forRoot(MyApp),
    IonicStorageModule.forRoot()
  ],
  bootstrap: [IonicApp],
  entryComponents: [
    // ...
  ],
  providers: [
    // ...
  ]
})
export class AppModule {}
```

Almacenamiento Clave-Valor

- Se inyecta el storage en el componente
- En el ejemplo se utiliza el mecanismo de almacenamiento basado en parejas clave-valor

```
import { Storage } from '@ionic/storage';

export class MyApp {
  constructor(private storage: Storage) { }

  ...

  // set a key/value
  storage.set('name', 'Max');

  // Or to get a key/value pair
  storage.get('age').then((val) => {
    console.log('Your age is', val);
  });
}
```

Almacenamiento de objetos JSON

- Guardamos objetos completos JSON utilizando una clave para almacenarla
- En el ejemplo se intenta recuperar una variable JSON con la clave ‘favoriteCards’:

```
favoriteCards: Card[] = [];

    constructor(private route: ActivatedRoute,
                private cardService:CardService,
                private storage: Storage) {

        this.storage.get('favoriteCards').then ((favoriteCards) => {

          this.favoriteCards = favoriteCards || {};
        });

    }
```

- La primera vez se completa la estructura con un valor vacío

Almacenamiento de objetos JSON

- Se utiliza el método set para guardar el objeto JSON
- En el ejemplo se almacena una lista de cartas favoritas donde se añade una carta cuando esta no es favorita previamente

```
public favoriteCard (card: Card){
    if (card.favorite){
      card.favorite = false;
      delete this.favoriteCards[card.cardId];
    }else{
      card.favorite = true;
      this.favoriteCards[card.cardId] = card;
    }

    this.storage.set('favoriteCards', this.favoriteCards);

  }
```

Introducir un botón back

- Se utiliza el ion-back-button para introducir el botón hacia atrás
- Si se utiliza la navegación de Angular es recomendable introducir el campo defaultHref="/". Otra alternativa, es introducir directamente la ruta que apunta a la página a la que queremos volver
- En el ejemplo navegamos de la página card-listing a la página card-decks

```
<ion-toolbar>
  <ion-title>{{cardDeck}}</ion-title>
  <ion-buttons slot="start">
    <ion-back-button icon="arrow-back" defaultHref="/tabs/cards"></ion-back-button>
  </ion-buttons>
</ion-toolbar>
```

Angular Google Maps

- Permite una integración de google maps basada en un API Javascript
- Es una solución más ligera que la alternativa nativa que plantea @ionic-native/google-maps
- Para su uso de debe instalar el paquete de Angular Google Maps en el proyecto

- **npm install @agm/core**

Angular Google Maps

- Se debe incluir la librería en el módulo de la App o de la página en la que se desee utilizar
- Además se debe añadir un Api Key que se obtiene desde el Google Cloud Platform
- Se usa una clave tipo: API Javascript Key

```
import { AgmCoreModule } from '@agm/core';

@NgModule({
  imports: [
    CommonModule,
    FormsModule,
    ReactiveFormsModule,
    IonicModule,
    RouterModule.forChild(routes),
    AgmCoreModule.forRoot({
      apiKey: 'AIzaSyD9BxeSvt3u--Oj-_GD-qG2nPr1uODrR0Y'
    })
  ],
```

AgmMap

- El componente principal es el AgmMap que se encarga de renderizar un mapa de google en la página donde esté incrustado
- Para poder ser visualizado es obligatorio definirle una altura predefinida (height) en su estilo

```
agm-map {
  width: 100%;
  height: 94vh;
}
```

- Se deben definir tres inputs para fijar el mapa, la latitud (latitude), longitud (longitude) y el zoom (por defecto es 8) define el nivel de zoom del mapa

AgmMarker

- Permite la definición de un marcador del mapa dentro de un AgmMap
- Es una directiva de AGM denominada agm-marker, a la cual se le debe definir su latitud y longitud para situar el marcador en el mapa
- Cada marcador puede contener una ventana de información que permite mostrar los datos, imágenes y un enlace que muestre su contenido sobre el mapa. Se denomina AgmInfoWindow
- El AgmInfoWindow se define mediante la etiqueta agm-info-window

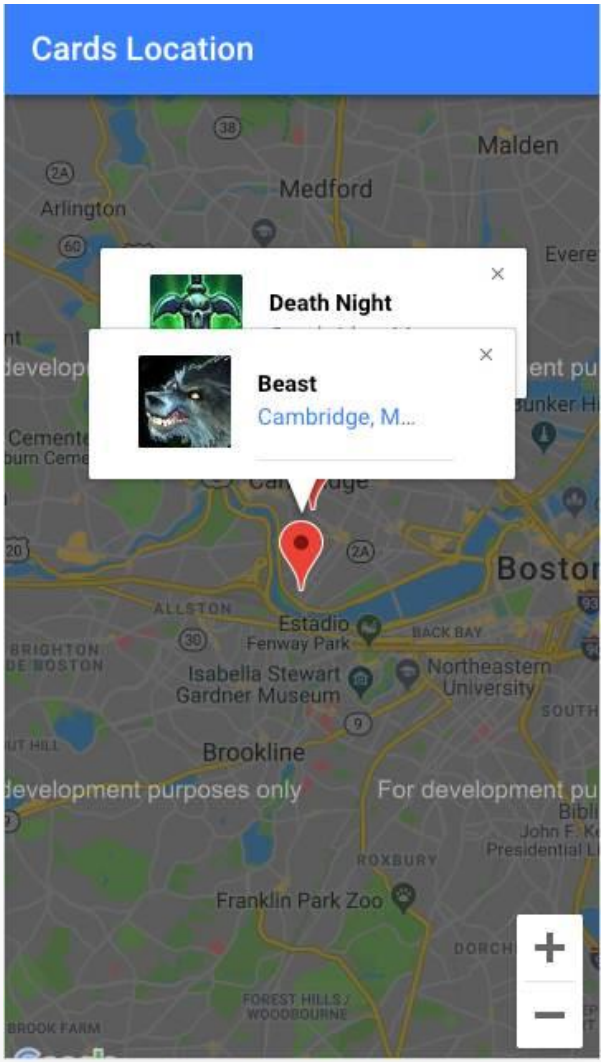
Ejemplo de mapa en HearthStone

- Se muestran un conjunto de cartas distribuidas en un mapa
- Cada carta con su imagen y un texto descriptivo

```
<div id="nearby-map" class="nearby-map">
  <agm-map [latitude]="nearCards[0].lat" [longitude]="nearCards[0].long" [zoom]="12">
    <agm-marker *ngFor="let card of nearCards"
     [latitude]="card.lat" [longitude]="card.long">
      <agm-info-window>
        <ion-item tappable routerLink="/card-detail/{{card.id}}">
          <ion-thumbnail slot="start">
            <img [src]="card.thumbnail">
          </ion-thumbnail>
          <ion-label>
            <h3><strong>{{card.title}}</strong></h3>
            <p no-margin>
              <ion-text color="primary">{{card.city}}, {{card.state}} •
                <span class="fw700">{{ card.price }}</span>
              </ion-text>
            </p>
          </ion-label>
        </ion-item>
      </agm-info-window>
    </agm-marker>
  </agm-map>
</div>
```

Ejemplo del Mapa en HearthStone

- En un fichero mock-cards.ts introduzco cartas con sus direcciones




```
const addresscards: Array<any> = [
  {
    id: 1,
    address: '18 Henry st',
    city: 'Cambridge',
    state: 'MA',
    zip: '01742',
    title: 'Beast',
    long: -71.11095,
    lat: 42.35663,
    picture: 'assets/image/Beast.png',
    thumbnail: 'assets/image/Beast.png',
    images: [
      'assets/image/Beast.png'
    ],
    tags: 'Oriental',
    description: 'Lorem ipsum dolor sit amet, consectetur
    label: 'open',
    period: 'none',
    price: '$$$',
    rating: 4.4
  },
```

Menú

- El componente menú es una caja de navegación que se desliza desde un lateral de la vista actual
- Por defecto, se desliza desde la izquierda, pero dicho lado puede modificado
- El menú será mostrado de forma diferente según su modo, sin embargo, su aparencia se puede cambiar en cualquiera de los tipos de menú disponibles
- Un elemento del menú es hermano (sibling) del elemento del contenido raíz
- Adjunto al contenido puede haber cualquier número de menús

Componente menú (ion-menu)

- Normalmente se añade el componente menú en el app.component o componente raiz de la App, de forma que hacemos el menú accesible para toda la App
- El componente principal del menú es ion-menu que tiene como propiedades más importantes:
  - side: si queremos ponerlo a la izquierda ‘start’ por defecto, o a la derecha ’end’
  - type: tipos de menú, puede ser ‘overlay’, ‘reveal’ o ‘push’
  - menuId: Se define un id para el menú para asociarle posteriormente elemento

Componente menú (ion-menu)

- Un ion-menu normalmente contiene una cabecera ion-header y un contenido (ion-content)
- Mientras en la cabecera podremos introducir cualquier tipo de control para mostrar información
- En el contenido se muestran las opciones del menú mediante una lista

```
<ion-menu side="start" menuId="custom" class="my-custom-menu">
  <ion-header>
    <ion-toolbar color="tertiary">
      <ion-title>Custom Menu</ion-title>
    </ion-toolbar>
  </ion-header>
  <ion-content>
    <ion-list>
      <ion-item>Menu Item</ion-item>
      <ion-item>Menu Item</ion-item>
      <ion-item>Menu Item</ion-item>
      <ion-item>Menu Item</ion-item>
      <ion-item>Menu Item</ion-item>
    </ion-list>
  </ion-content>
```

Elementos del menú (ion-menu-toggle)

- El componente MenuToggle se usa para alternar un menú abierto o cerrado
- Por defecto, solo es visible cuando el menú seleccionado está activo. Un menú está activo cuando se puede abrir y cerrar
- Si el menú está deshabilitado, el menú se marca como no activo y el ion-menu-toggle se oculta.

- Propiedades:
  - autoHide : permite ocultar automáticamente el contenido cuando el menú no está activo. Por defecto es verdadero.
  - Menu: menuId al cual se asocia, si no se especifica buscará el primer menú que se defina en la App.

Elementos del menú (ion-menu-toggle)

- Es típico introducir los ion-menu-toggle en una lista (ion-list), donde cada ion-item están dentro de un componente ion-menu-toggle

```
<ion-content class="bg-profile">
  <ion-list>
    <ion-list-header color="dark">
      <ion-label>Menu</ion-label>
    </ion-list-header>
    <ion-menu-toggle auto-hide="false" *ngFor="let p of appPages">
      <ion-item [routerLink]="[p.url]" color="primary">
        <ion-icon slot="start" [name]="p.icon" color="light"></ion-icon>
        <ion-label>
          {{p.title}}
        </ion-label>
      </ion-item>
    </ion-menu-toggle>
  </ion-list>
</ion-content>
```

Elementos del menú (ion-menu-toggle)

- En el AppComponent se debe almacenar la lista de direcciones del menú para ser mostradas en los ion-menu-toggle
- Se usa una estructura de array (p.e. appPages) para tener la lista de páginas, direcciones y los iconos correspondientes

```
export class AppComponent implements OnInit{

  public appPages: Array<Pages>;

  constructor(=) {
    this.initializeApp();

    this.appPages = [
      {
        title: 'Card Decks',
        url: '/tabs/cards',
        direct: 'forward',
        icon: 'folder'
      },
      {
        title: 'Location',
        url: '/nearCards',
        direct: 'forward',
        icon: 'compass'
      },
      {
        title: 'Favorites',
        url: '/favorites',
        direct: 'forward',
        icon: 'star'
      }
    ];
```

MenuController

- El controlador de menú facilita el manejo de un menú por código.
- Los métodos proporcionados se pueden usar para mostrar el menú, habilitar el menú, alternar el menú, etc.
- Es inyectado el MenuController en el componente que contiene el menú

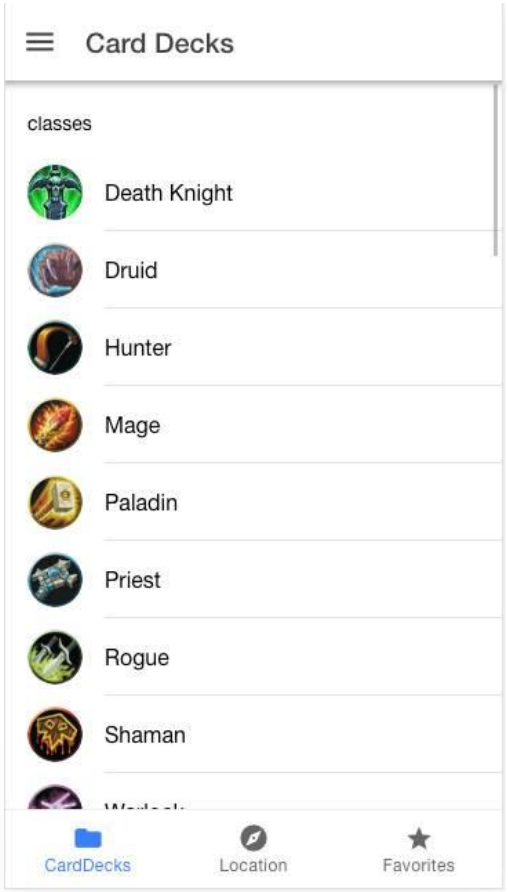
```
constructor(
  private platform: Platform,
  private splashScreen: SplashScreen,
  private statusBar: StatusBar,
  private router: Router,
  private ctrMenu:MenuController
) {
  this.initializeApp();

  this.appPages = [];

}

ngOnInit() {
  this.router.events.subscribe((event: RouterEvent) => {
    if (event instanceof NavigationEnd && event.url === '/login') {
      this.ctrMenu.enable(false);
    }
  });
}
```

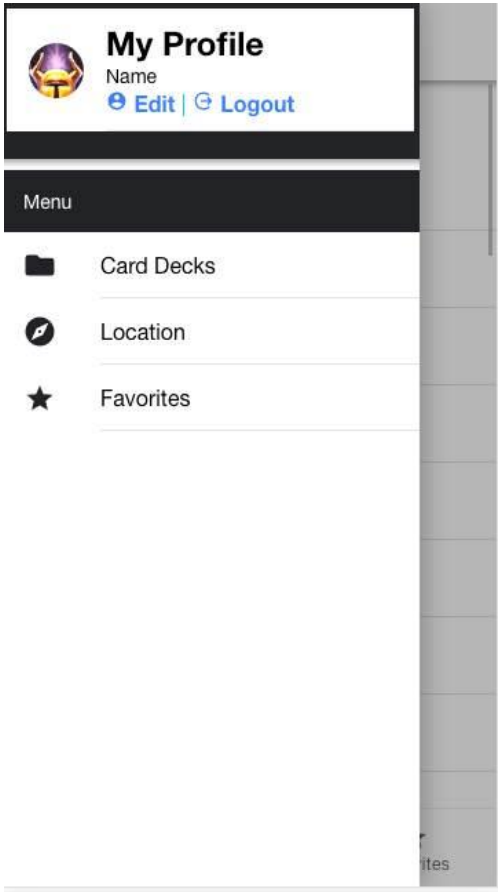
Botón Menu (ion-buttons)

- El botón menú (forma de hamburguesa) crea el icono y la funcionalidad para abrir el menú en la página donde se defina
- Normalmente se introduce en la cabecera de la página junto al título

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

Ejemplo de Menú en HearthStoneApp





Theming

- Ionic está diseñado para ser una pizarra en blanco que pueda ser personalizada para ajustarse a una marca, mientras mantiene los estándares de iOS y Android
- El theming de ionic está construido mediante CSS, los cuales vienen con estilos ya predefinidos para que sea muy sencillo modificarse
- Ionic tiene 9 colores por defecto que pueden usarse para cambiar el color de muchos componentes
- Todos los componentes de ionic son tematizados mediante variables CSS, que permite añadir valores dinámicos mediante el preprocesamiento de Sass

Estilos de las plataformas

- Ionic proporciona estilos específicos basándose en el dispositivo donde la aplicación es ejecutada
- Especificar el estilo de los componentes en función del dispositivo permite escribir el componente un vez pero mostrar una apariencia al usuario dependiendo de donde esté
- Ionic usa el atributo **mode** en los componentes para personalizar su aparencia en función de la plataforma
- Cada plataforma tiene su mode por defecto que puede ser modificado globalmente mediante un servicio IonicModule

Estilos en las plataformas

- Por defecto, tenemos los siguientes valores:
  - Android: mode = md (Material Design)
  - iOS: mode = ios (ios Styles)
  - Cualquier otra plataforma (Web o Escritorio): mode = md
- El modo y la plataforma no es lo mismo, si se desea que una plataforma use un modo en concreto, se debe realizar desde el IonicModule

IonicModule

- El modulo de Ionic (IonicModule) proporciona un modo de cambiar las propiedades de los componentes globalmente para una App.
- Se puede fijar el mode, la apariencia del botón tab (tab button), las animaciones, etc.
- En el ejemplo se fuerza a que el modo sea siempre material design y se desactiva el efecto ripple (ondas generadas con las pulsaciones)

```
import { IonicModule } from '@ionic/angular';

@NgModule({
  ...
  imports: [
    BrowserModule,
    IonicModule.forRoot({
      rippleEffect: false,
      mode: 'md'
    }),
    AppRoutingModule
  ],
  ...
})
```

Plataforma

- El servicio Platform puede ser usado para obtener información sobre el dispositivo actual
- Se pueden obtener las plataformas asociadas al dispositivo usando el servicio platform, incluyendo si la App está siendo vista en una tablet, o es un dispositivo móvil o es un navegador
- Se puede obtener la orientación del dispositivo, si usa una dirección de lenguaje u otra (de izquierda a derecha)
- Con esta información se puede personalizar tu aplicación para ajustarte perfectamente al dispositivo

Plataforma

- Para su uso tan solo hay que importar el servicio Platform e inyectarlo en el componente donde lo vayamos a utilizar
- Es común su uso en el componente inicial AppComponent
- Se usa el método ready que devuelve una promise cuando la plataforma está preparada y la funcionalidad nativa está lista
- En el ejemplo se fija el estilo del statusBar, y se oculta el splashScreen durante un segundo a partir de que la plataforma esté lista

```
export class AppComponent {

  public appPages: Array<Pages>;

  constructor(
    private platform: Platform,
    private splashScreen: SplashScreen,
    private statusBar: StatusBar,
    private translate: TranslateProvider,
    private translateService: TranslateService,
    public navCtrl: NavController
  ) {
    this.appPages = [];

    this.initializeApp();
  }

  initializeApp() {
    this.platform.ready().then(() => {
      this.statusBar.styleDefault();
      setTimeout(() => {
        this.splashScreen.hide();
      }, 1000);
```

Variables CSS

- Los componentes Ionic usan variables CSS para personalizar su apariencia
- Las variables CSS permiten que un valor sea almacenado en un solo sitio y referenciado en múltiples lugares
- Permiten también cambiar la CSS dinámicamente en tiempo real (lo que requiere un preprocesador de CSS)
- Las variables CSS hacen sencillo la personalización de los componentes Ionic para ajustarse a una marca o a un tema

Variables Globales CSS

- Las variables CSS pueden ser definidas globalmente mediante el selector **:root y** con **.ios y .md** para **ios** y android respectivamente
- Cuando creamos un proyecto con IonicCLI, se crea un fichero denominado src/theme/variables.scss, donde se pueden sobreescribir las variables Ionic por defecto

```
/* Set variables for all modes */
:root {
  /* Set the background of the entire app */
  --ion-background-color: #ff3700;

  /* Set the font family of the entire app */
  --ion-font-family: -apple-system, BlinkMacSystemFont, "Helvetica Neue",
}

/* Set text color of the entire app for iOS only */
.ios {
  --ion-text-color: #000;
}

/* Set text color of the entire app for Material Design only */
.md {
  --ion-text-color: #222;
}
```

Variables Globales CSS

- El fichero variables.css suele albergar los colores estándares para toda la aplicación
- Especificamente se definen los 9 colores principales que sirven para colorear los componentes
- Para cambiar la apariencia globalmente, ajustándose a dichos colores podemos hacer uso del Color Generator
- https://ionicframework.com/docs/theming/color-generator
- Que genera las variables para los 9 colores, y pueden ser copiadas sin errores al fichero variables.css

Colores

- Ionic tiene 9 colores por defecto que pueden usarse para cambiar el color de muchos componentes
- Cada color tiene realmente una colección de 4 propiedades
- Un componente Ionic para cambiar sus colores por defecto usando el atributo color
- En el ejemplo, tanto el texto como el background se basan en las propiedades contrast y base del color asignado (respectivamente)

```
:root {
  /** primary **/
  --ion-color-primary: #3880ff;
  --ion-color-primary-rgb: 56, 128, 255;
  --ion-color-primary-contrast: #ffffff;
  --ion-color-primary-contrast-rgb: 255, 255, 255;
  --ion-color-primary-shade: #3171e0;
  --ion-color-primary-tint: #4c8dff;
```

```
<ion-button color="primary">Primary</ion-button>
```

Colores por capas

- Cada color consiste en las siguientes 4 propiedades: base, contrast, shade y tint
- Base y contrast requieren tener la propiedad también en RGB para poder dar soporte a los navegadores que tienen “The Alpha Problem”
- Para cambiar los valores de un color, todas las propiedades deben ser modificadas
- Por ejemplo, si elegimos un color oscuro como #006600 (verde oscuro) como base, debemos dar un color de contraste claro #ffffff (blanco), mientras que shade y tint representan diferentes estados del botón

Añadir un color

- Si queremos añadir un nuevo color, a parte de los 9 predefinidos. Debemos añadir una nueva clase que tenga el formato: **.ion-color-{color},** donde {color} sería el nombre del color a añadir
- Se añade una variable color llamada ‘favorite’:

```
.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="favorite">Favorite</ion-button>
```

Definir un tema

- Ionic proporciona un conjunto de variables globales que son usadas por todos los componentes para cambiar el tema por defecto en toda la aplicación

- Existen 2 tipos diferentes de variables de tema:
  - Aplication Colors: son útiles para crear facilmente temas oscuros o temas que se ajusten a una marca. Ver en: https://ionicframework.com/docs/theming/advanced#application-colors
  - Stepped Colors (colores escalonados): para incrementar la importancia y profundidad de un diseño, se necesita usar diferentes tonos de fondo y colores de texto

Stepped Colors

- Cuando actualizamos las application colors, podemos cambiar la apariencia de muchos componentes, los cuales pueden verse apagados o dañados
- En algunos componentes, queremos usar un tono más oscuro que el background por defecto o más claro que el texto
- Se usan los stepped colors para definir esas variaciones sutiles
- Ionic proporciona un generador de Stepped colors donde se debe ajustar el background y el text color:
- https://ionicframework.com/docs/theming/advanced#generate-stepped-color-variables

Ejercicio 4 (optativo): HearthStoneApp

1. Instalamos el plugin y la librería del storage desde la consola en nuestra aplicación (ver pag. 4)
2. Importamos el **IonicStorageModule** en el app.module (ver pag. 5)
3. Importamos e inyectamos el modulo **Storage** en la página *card-listing* (ver pags. 6)
4. Definimos una estructura llamada favoriteCards de tipo array de Cards (ver pag. 7)
5. En el constructor de card-listing invocamos al get del storage para recuperar de la clave 'favoriteCards' (ver pag. 7)
6. Añadimos a el interfaz *Card* en card.model la propiedad **favorite** con el tipo *boolean*.
7. Añadimos la función favoriteCard(card) que almacena la carta que es definida como favorita (ver pag. 8)
8. Añadir en la página de estilos de la card-listing los estilos like-icon y favorite del icono flame (ver pag. 42)

Ejercicio 4 (optativo): HearthStoneApp (Cont. 1)

8. Añadimos a la plantilla de card-listing un ion-icon de tipo “flame” para representar si la carta es favorita o no Además, se define la propiedad ngClass para asignar si la carta tiene la clase favorite o like-icon. (ver pag. 41)
9. Instalar el paquete Angular Google Maps en el proyecto (ver Pag. 10)
10. Crear una página llamada Location con el IonicCLI
11. Importar en el modulo de la página location la librería Agm (ver Pag. 11)
12. Definir en la carpeta shared, un fichero llamado mock-cards.ts donde se especifiquen varias cartas con sus direcciones. (ver Pag. 15)
13. Importar el fichero mock-cards.ts en el componente de Location y cargarlo en el constructor asignando las páginas a un array llamado nearCards
14. Representar el mapa en la plantilla de Location (ver pag. 14)

Introducción del icono favorito

- Hacemos uso de uno de los iconos multiplataforma de ionic (https://ionicframework.com/docs/v3/ionicons/) llamado “flame”
- Se usa la propiedad ngClass de angular que permite hacer dinámica la asignación de la class de un elemento. En este caso por la propiedad card.favorite de la carta elige entre la clase ‘favorite’ o la clase ‘like-icon’ (normal)

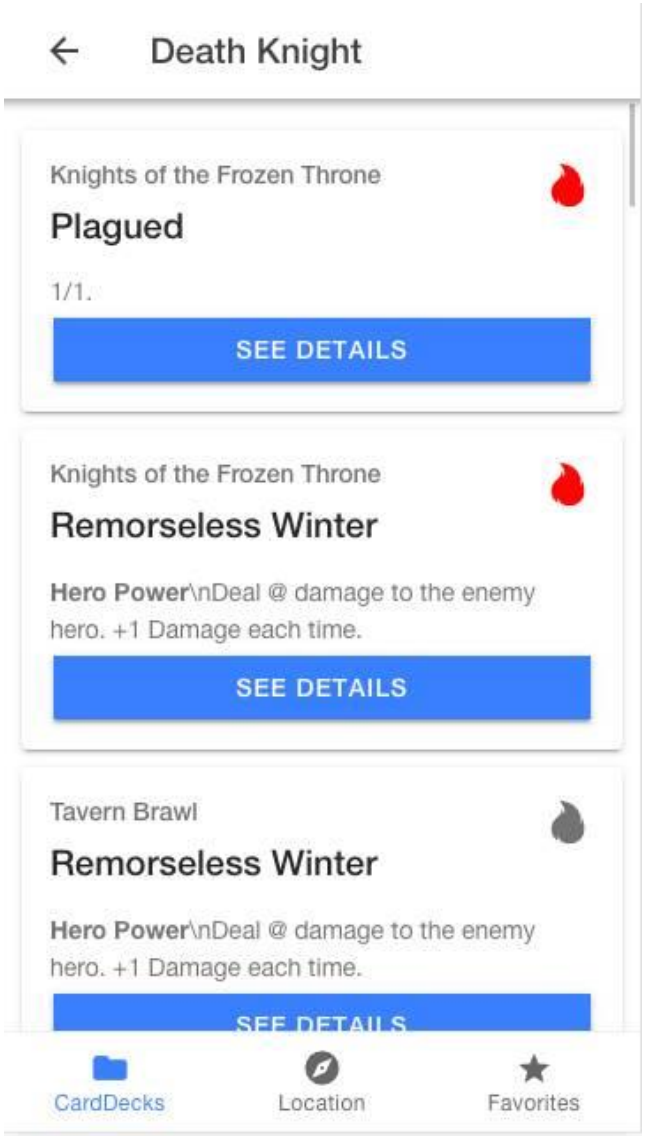
```
<ion-card-subtitle>
  {{card.cardSet}}

  <ion-icon (click)="favoriteCard(card)" [ngClass]="card.favorite ? 'favorite' : 'normal'"
        name="flame" class="like-icon"> </ion-icon>

</ion-card-subtitle>
```

Estilos del icono flame

- Se definen los estilos del icono en la página de estilos de la página card-listing.page.scss para mostrar si es favorito (favorite) o si es normal (like-icon)

```
.like-icon {
  float: right;
  font-size:30px;
}

.favorite {
  color: red;
}
```

Vista final del ejercicio



Documentación

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