universidad de león

Escuela de Ingenierías Industrial, Informática y Aeroespacial

GRADO EN INGENIERÍA INFORMÁTICA

PRÁCTICAS EXTERNAS

Autor: Iván Alexis Castro Martínez

Febrero, 2019

Índice

1. Introducción

Sabemos que hoy día tenemos la posibilidad de viajar a casi cualquier lugar gracias a los avances de los medios de transporte.
El gran inconveniente de los amantes de los viajes, o los que tienen un espíritu aventurero es que no saben que visitar o como, sin tener que formar parte de un grupo turístico con guía.

"TurismoGO" es una aplicación inspirada en la famosa App PokémonGO, que ofrece la solución a esto. Vayas donde vayas (ya sea por trabajo y tengas que ir cada vez a un sitio distinto, o simplemente por unas vacaciones en las que decidiste coger un avión), te dice lo que no te puedes perder en tu viaje, para que puedas viajar con mayor libertad y disfrutar de tus vacaciones de la mejor manera posible.

En el momento que tú desees, podrás abrir TurismoGO y conocer tu ubicación e informarte de cualquier sitio de interés cercano.

2. Objetivos

El objetivo primordial de TurismoGO (como ya hemos introducido) es satisfacer las necesidades de un Turista, viajante o persona que necesite conocer su ubicación y todo lo que le rodea en ese preciso momento.

Para ello, el usuario contará con un mapa en la App, que mediante geolocalización, le ubicará su posición y un montón de sitios de interés a su alrededor tanto monumentos, museos, playas, asi como restaurantes. Sí, es lo más parecido a un guía turístico personal.

Ejemplo: Un Turista que haya viajado durante muchas horas de New York a Tokyo, puede loggear con su cuenta y rápidamente saber donde está el restaurante más cerca, o cual es la parada de bus que más le conviene.

Además del mapa, el usuario tiene a su disposición una Lista con los sitios de interés que le rodean (y su distancia hasta ellos) ordenados de más cercano a más lejano.

En un futuro (aprovechando el sistema de cuentas mediante Auntenticación ya implementado) se pretende añadir una funcionalidad extra que te permita formar grupo con otros usuarios (amigos, compañeros de trabajo, etc). Y mientras no te salgas del grupo, podrás conocer la posición de cada miembro del grupo, facilitándoos una reunión, un reencuentro, o simplemente la tranquilidad y seguridad de saber donde se encuentra alguien querido).

3. Planificación

Dado que no hemos llevado a cabo muchas reuniones, cuando teníamos reunión delimitábamos el trabajo a largo plazo, organizándolo por diferentes etapas en función del desarrollo.

3.1 Qué queremos hacer

Desarrollar una App de Turismo que use geolocalización y te enseñe los sitios de interés más cercanos a nuestra posición. Además cada usuario tendrá su cuenta y que haya una clasificación Online de logros, registrando los sitios por los que has pasado

¿Pudimos cumplir los objetivos? Salvo el sistema Online de logros, sí.
Sin embargo, al inicio acordamos usar librerías de código abierto para los mapas como Leaflet y OpenStreetMaps para personalizar los mapas. Desafortunadamente se tuvo que desarrollar en Google Maps (motivo en las Conclusiones del final).

3.2 Que medios se usarán e instalación

Poner todo lo necesario a punto para proseguir con el desarrollo de la App.
Instalar Ionic3, Cordova, npm, nodejs, GoogleMaps, IOS y Android.
Usaremos el host que nos proporciona Ionic para alojar el server.

Este proceso se finalizó a mediados de Noviembre.

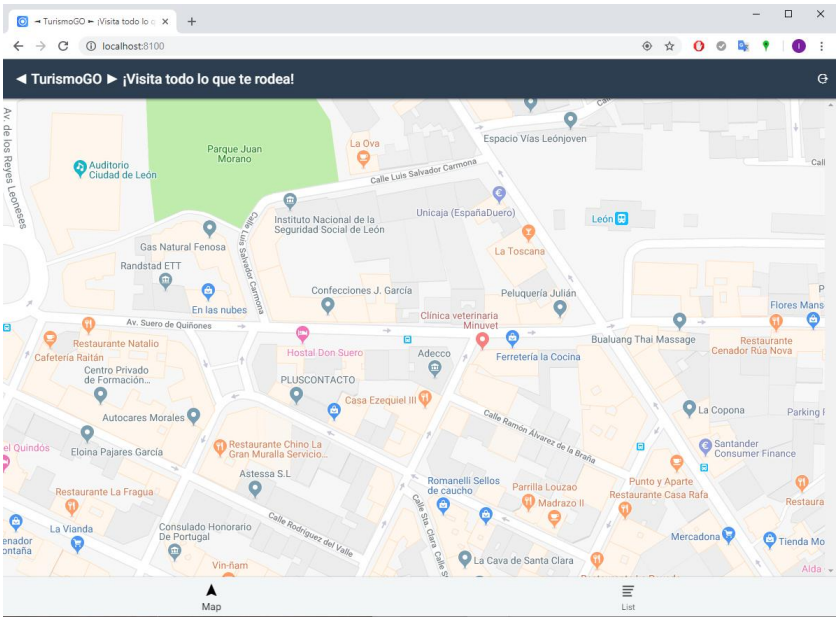
3.3 Aprendizaje del lenguaje, creación de páginas y desarrollo

Sin duda, esta es la etapa que más tiempo ha llevado dado que primero fué necesario aprender como se desarrollan las App, conocer su estructura, etc.
Importante: Este periodo también abarca lo que está detrás de cada página: los proveedores, sus archivos js, css y html donde se implementarán las funciones que desarrollarán estas páginas, etc.

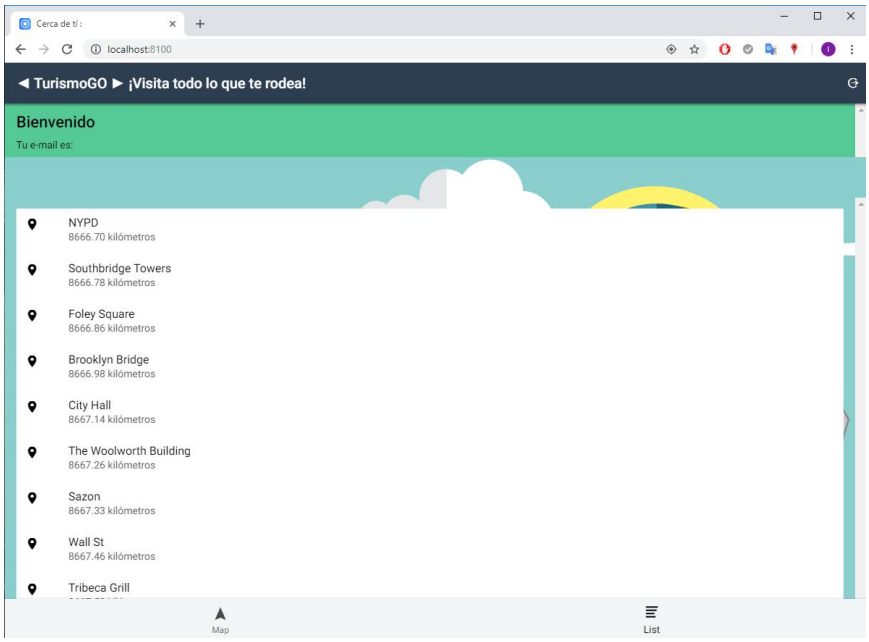
Teniendo en cuenta las funcionalidades que requerimos que tenga la App, para iniciar creamos 2 páginas primero y luego otras 2:

3.4 Páginas a desarrollar

-Página *Map*: Es nuestra página principal, donde irá el mapa y se verán los marcadores de sitios de interés cercanos.



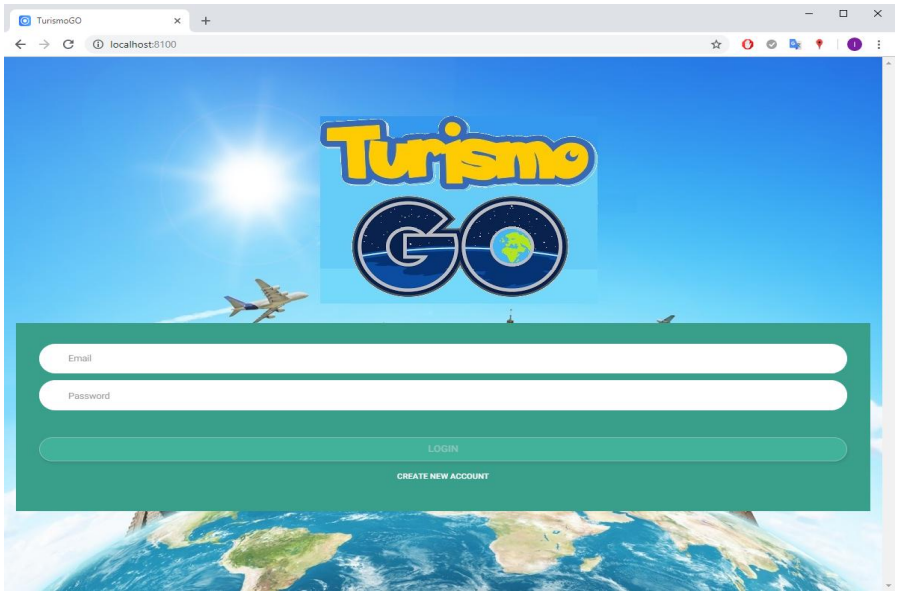
-Página *List*: En esta página podremos a disposición del usuario una Lista con los sitios de interés más cercanos detallando la distancia hacia ellos desde el más cercano al más lejano.



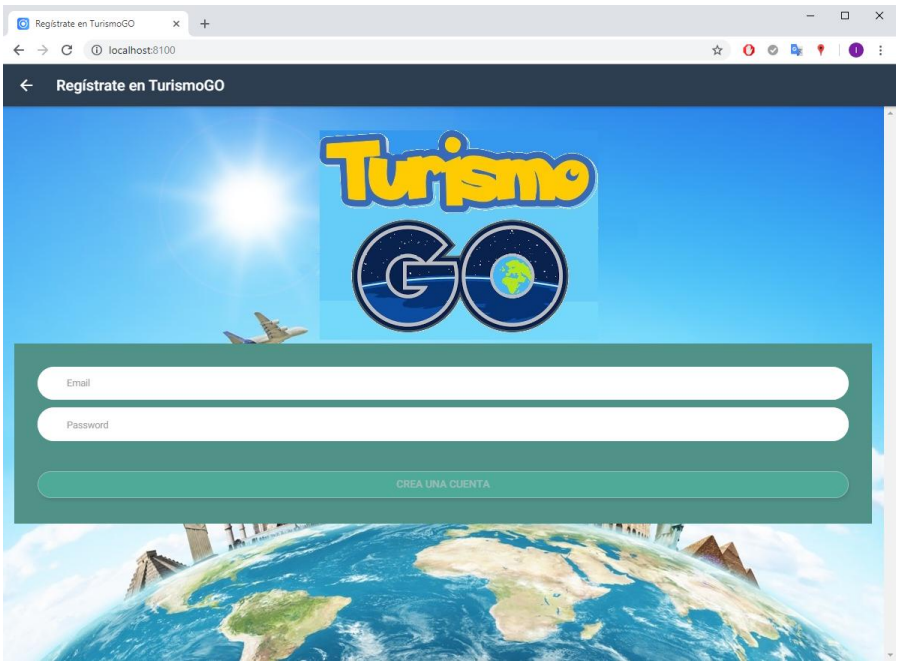
Este proceso (junto con su desarrollo) se finalizó a finales de Diciembre.

Más tarde, una vez implementado todo, crearíamos otras dos páginas para la Autenticación del usuario.

-Página *Login*: Página en la que logearemos con nuestra cuenta y podremos acceder a las funcionalidades de la App.



-Página *Register*: Página en la que crearemos una cuenta. Una vez creada redirige a Login.



Este proceso (junto con su desarrollo) se finalizó a mediados de Enero.

3.5 Informe final

Aunque hay partes que se fueron haciendo a medida que avanzaba el proyecto, hay cosas que necesitaban esperar hasta el final para poder reflejarlas según el resultado obtenido.
Dado que no se finalizó el Informe a tiempo, es entregado en Segunda Convocatoria.

3.6 Calendario con duración de etapas (aproximado)

4. Herramientas/tecnologías utilizadas

Dado que ya hemos mencionado brevemente los medios usados, explicaremos para que los usaremos:

IONIC 3: Es un framework nativo muy útil que te permite crear aplicaciones híbridas (si programamos sobre este software, podremos programar para iOS y Android a la vez).
De manera que nuestra App abastecería casi todo el mercado de telefonía móvil.

Angular: Es muy útil pues está incorporado en ionic y sino tendríamos que recurrir a funcionalidades de la Web. Básicamente nos ayudará a crear el contenido.

Cordova: Al igual que Angular, ayudan a la creación del App. Ayuda a estilizar y dar forma a la interfaz de la aplicación.

Firebase: Herramienta fundamental para la creación usuarios y contraseñas y almacenamiento en la base de datos. Muy completa y dedicada a aplicaciones.

NodeJS y npm: NodeJS es un código abierto de JavaScript para generar apps de forma óptima. Nos permite usar JavaScript del lado del server, y funciona realmente rápido y eficaz. Npm se usa para instalar los componentes necesarios para la App mediante NodeJS.
GoogleMaps: Es increíble la cantidad de cosas que podemos hacer con la conocida librería de mapas de Google. Podemos personalizar de manera sencilla lo que debe llevar nuestro mapa. Por desgracia es gratuita sólo si la App al subirla a la nube no genera un monto de beneficio máximo establecido.

Android y iOS: Para poder construir nuestra App en ambos lenguajes debemos tenerlos instalados en nuestro PC.

5. Núcleo del trabajo

5.1 Resumen del código

A continuación, desarrollaremos y explicaremos el código fundamental, tramo a tramo.
Nos centraremos en tocar puntos como:

Visualización de múltiples marcadores en Google Maps

Calcular la distancia entre dos puntos y mapear esa información en una matriz

Ordenar datos y mostrarlos en una lista (ordenados de menor a mayor)

Creación de Autenticación con Usuario y Contraseña con Firebase

Creación de Registro con Usuario y Contraseña con Firebase

Enlace entre pestañas

5.2 Desarrollo paso a paso

5.2.1 Instalación e iniciación de medios

La base de todo es IONIC, por lo cual comenzaremos creando las páginas necesarias para cada apartado.

Ionic g page Map, List, Register, Login

Además de los correspondientes proveedores, 3 para la gestión de la App:
-*Ionic g provider Connectivity*: Se usará junto con el proveedor de GoogleMaps para ayudar a cargar el SDK de Javascript de Google Maps.
-*Ionic g provider GoogleMaps.*
-*Ionic g provider Locations*: Será el responsable de cargar los datos de la ubicación en la aplicación, y también se encargará de calcular las distancias desde la ubicación del usuario y de ordenar los datos en consecuencia.

Y este para la la Autenticación:
Ionic g provider users.

Una vez hecho esto, para poner todo en marcha el fichero app.module.ts quedaria así:

```
import { MyApp } from './app.component';
import { HomePage } from '../pages/home/home';
import { MapPage } from '../pages/map/map';
import { ListPage } from '../pages/list/list';
import { LoginPage } from '../pages/login/login';
import { RegisterPage } from '../pages/register/register';
import { ConnectivityProvider } from '../providers/connectivity/connectivity';
import { GoogleMapsProvider } from '../providers/google-maps/google-maps';
import { LocationsProvider } from '../providers/locations/locations';
import { AuthService } from '../providers/users/auth-service';

//Creamos 4 páginas:

//'Map' -> Muestra el Mapa de Google con markers que indican la ubicación de lugares cerca.
/* 'List' -> Muestra los lugares cercanos en formato de lista,
  'Login' y 'Register' -> se encargarán de la autenticación de los usuarios.
 además muestra la distancia al lugar y está ordenada por distancia de menor a mayor) */

 @NgModule({
  declarations: [
    MyApp,
    HomePage,
    MapPage,
    ListPage,
    LoginPage,
    RegisterPage
  ],
  imports: [
    BrowserModule,
    HttpModule,
    IonicModule.forRoot(MyApp)
  ],
  bootstrap: [IonicApp],
  entryComponents: [
    MyApp,
    HomePage,
    MapPage,
    ListPage,
    LoginPage,
    RegisterPage
  ],
  providers: [
    GoogleMaps, StatusBar, SplashScreen, Network, Geolocation,
    AuthService, ConnectivityProvider, GoogleMapsProvider, LocationsProvider,

    {provide: ErrorHandler, useClass: IonicErrorHandler}
  ]
})
export class AppModule {}
```

5.2.2 Creación de Map y List

Ahora crearemos las pestañas Mapa y Lista en el *home.ts*.
Importamos las dos páginas y configuramos referencias a ellas. Además al constructor le pasaremos el nombre e e-mail con el que se haya registrado el usuario en el proceso de Autenticación.

```
import { Component } from '@angular/core';
import { NavController, IonicPage } from 'ionic-angular';
import { MapPage } from '../map/map';
import { ListPage } from '../list/list';
import { AuthService } from '../../providers/users/auth-service';

@Component({
  selector: 'page-home',
  templateUrl: 'home.html'
})
export class HomePage {

  username = '';
  email = '';

  //Creamos las dos páginas y las referenciamos
  tab1Root: any = MapPage;
  tab2Root: any = ListPage;

  constructor(private nav: NavController, private auth: AuthService) {
    let info = this.auth.getUserInfo();
    console.log('info', info)
    this.username = info['name'];
    this.email = info['email'];
  }

  public logout() {
    this.auth.logout().subscribe(succ => {
      this.nav.setRoot('HomePage')
    });
  }
}
```

Y configuramos las referencias a las pestañas y botones y mensajes del login en home.html:

```
<ion-header>
  <ion-navbar color="dark">
    <ion-title>
      ◄ TurismoGO ► ¡Visita todo lo que te rodea!
    </ion-title>
    <ion-buttons end>
      <button ion-button (click)="logout()">
        <ion-icon name="log-out"></ion-icon>
      </button>
    </ion-buttons>
  </ion-navbar>
</ion-header>

<ion-content class="home" padding>
  <h3>Bienvenido {{username}}</h3>
  Tu e-mail es: {{email}}
</ion-content>

<ion-tabs color="light">
  <ion-tab [root]="tab1Root" tabTitle="Map" tabIcon="navigate"></ion-tab>
  <ion-tab [root]="tab2Root" tabTitle="List" tabIcon="list"></ion-tab>
</ion-tabs>

<!-- Aquí gestionamos la plantilla y estilo de las pestañas Map y List -->
```

En ***map.ts*** tomamos una referencia al mapa y conectamos los elementos. Luego hacemos una llamada para inicializar el mapa usando el proveedor de Google Maps cuando platform esté listo. Además agregaremos los marcadores y esperaremos la carga de datos una vez que llamemos al proveedor de locations.

```
6   //Referenciamos el mapa y conectamos los elementos
7   @Component({
8     selector: 'page-map',
9     templateUrl: 'map.html',
10  })
11  export class MapPage {
12
13    @ViewChild('map') mapElement: ElementRef;
14    @ViewChild('pleaseConnect') pleaseConnect: ElementRef;
15
16    constructor(public navCtrl: NavController,
17      public maps: GoogleMapsProvider,
18      public platform: Platform,
19      public locations: LocationsProvider) {
20
21    }
22    //Hacemos una llamada para iniciar el mapa usando el proveedor de Google Maps
23    ionViewDidLoad() {
24
25      this.platform.ready().then(() => {
26
27        let mapLoaded = this.maps.init(this.mapElement.nativeElement, this.pleaseConne
28        let locationsLoaded = this.locations.load();
29
30        Promise.all([
31          mapLoaded,
32          locationsLoaded
33        ]).then((result) => {
34
35          let locations = result[1];
36
37          for (let location of locations) {
38            this.maps.addMarker(location.latitude, location.longitude);
39          }
40        });
41      });
42    }
43  }
```

En ***list.ts*** sólo mostrarremos los datos obtenidos más adelante con el cálculo de las distancias de las ubicaciones (método Havershine).

```
import { Component } from '@angular/core';
import { IonicPage, NavController } from 'ionic-angular';
import { LocationsProvider } from '../../providers/locations/locations';

@Component({
  selector: 'page-list',
  templateUrl: 'list.html',
})
export class ListPage {

  constructor(public navCtrl: NavController, public locations: LocationsProvider) {
  }

  ionViewDidLoad() {
    console.log('ionViewDidLoad ListPage');
  }
}
```

5.2.3 Configuración de Proveedores

5.2.3.1Conectivity

Para saber si estamos ejecutando en iOS o Android inyectaremos el servicio de Platform de Ionic. Lo utilizaremos para saber si el usuario está en línea en connectivity.ts:

```
import { Injectable } from '@angular/core';
import { Network } from '@ionic-native/network';
import { Platform } from 'ionic-angular';

// Clase generada para ayudar al proveedor de Google Maps a cargar el SDK de JavaScri
//Además detectará si tenemos internet o no
//'Platform' es el servicio de Ionic que permite saber si ejecutamos en Android o iO

declare var Connection;

@Injectable()
export class ConnectivityProvider {

  onDevice: boolean;

  constructor(public platform: Platform,
              public network: Network,){
    this.onDevice = this.platform.is('cordova');
  }

  isOnline(): boolean {
    if(this.onDevice && this.network.type){
      return this.network.type !== Connection.NONE;
    } else {
      return navigator.onLine;
    }
  }

  isOffline(): boolean {
    if(this.onDevice && this.network.type){
      return this.network.type === Connection.NONE;
    } else {
      return !navigator.onLine;
    }
  }
```

Y ahora configuraremos la sección fundamental de la aplicación, configuramos la implementación de GoogleMaps.
Lo veremos por funciones:

5.2.3.2 GoogleMaps

***loadGoogleMaps**:* Verifica si el objeto de Google está disponible, lo que significaría que el SDK ya se ha cargado. Si el SDK no se ha cargado, verificamos si el usuario está en línea y luego lo cargamos inyectando el script en el documento json (podemos darle ubicaciones personalizadas que nosotros queramos tener indicadas en la Lista, o directamente las que más cerca tengamos de nuestra posición por geolocalización.).

```
loadGoogleMaps(): Promise<any> {

  return new Promise((resolve) => {

    if (typeof google == "undefined" || typeof google.maps == "undefined") {
      console.log("Google maps JavaScript needs to be loaded.");
      this.disableMap();

      if (this.connectivityService.isOnline()) {
        window['mapInit'] = () => {

          this.initMap().then(() => {
            resolve(true);
          });

          this.enableMap();
        }

        let script = document.createElement("script");
        script.id = "googleMaps";

        if (this.apiKey) {
          script.src = 'http://maps.google.com/maps/api/js?key=' + this.apiKey + '&
        } else {
          script.src = 'http://maps.google.com/maps/api/js?callback=mapInit';
        }
        document.body.appendChild(script);
      }
    }
    else {

      if (this.connectivityService.isOnline()) {
        this.initMap();
        this.enableMap();   //cuando se recupere la conexión saldrá un mensaje en c
      }
      else {
        this.disableMap();  //cuando se pierda la conexión saldrá un mensaje en cor
      }
    }

    this.addConnectivityListeners();
  });
}
```

Lo siguiente que hacemos (asumiendo que la conexión a Internet está disponible) es ejecutar la función ***initMap*** . Esto simplemente controla la creación de un nuevo mapa utilizando el SDK de Google Maps cargado y establece las coordenadas en la ubicación actual de los usuarios mediante la API de geolocalización.

```
initMap(): Promise<any> {

  this.mapInitialised = true;

  return new Promise((resolve) => {

    this.geolocation.getCurrentPosition().then((position) => {

      let latLng = new google.maps.LatLng(position.coords.latitude, position.coords

      let mapOptions = {
        center: latLng,
        zoom: 18,
        mapTypeId: google.maps.MapTypeId.ROADMAP
      }

      this.map = new google.maps.Map(this.mapElement, mapOptions);
      resolve(true);
    });
  });
}
```

Otra función importante aquí es ***addConnectivityListeners*** . Esto estará escuchando constantemente cuando el usuario vuelva a estar en línea, y cuando lo haga activará todo el proceso anterior. También llamará a las funciones ***enableMap*** y ***disableMap*** cada vez que se pierda o recupere la conexión a Internet. Y registra un mensaje en consola.

```
addConnectivityListeners(): void {

  document.addEventListener('online', ()=> {

    console.log("online");

    setTimeout(() => {

      if (typeof google == "undefined" || typeof google.maps == "undefined") {
        this.loadGoogleMaps();
      }
      else {
        if (!this.mapInitialised) {
          this.initMap();
        }

        this.enableMap();
      }

    }, 2000);

  }, false);

  document.addEventListener('offline', () => {

    console.log("offline");

    this.disableMap();

  }, false);

}
```

Finalmente la función ***addMarker*** administrará el agregar marcadores en nuestro mapa.

```
  addMarker(lat: number, lng: number): void {

    let usersLocation = new google.maps.usersLocation(lat, lng);

    let marker = new google.maps.Marker({
      map: this.map,
      animation: google.maps.Animation.DROP,
      position: usersLocation
    });
    this.markers.push(marker);
  }
}
```

Y aplicamos algo de CSS tanto para el mapa como para la lista, además de ajustar el tamaño del mapa al ancho de pantalla, configurar estilos del "Por favor, conéctese a internet", etc.

5.2.3.3 Location

Y finalmente en nuestro poveedor location.ts usaremos la Fórmula de Haversine, que permite calcular la distancia entre dos puntos dadas las coordenadas de latitud y longitud.

applyHaversine define la ubicación del usuario real a través de geolocalización, y desde ahí trazará las distancias a las ubicaciones elegidas (como ya mencioné antes, podemos trazar distancias a los sitios más cercanos o personalizar los sitios que queramos a través del fichero *locations.json*.

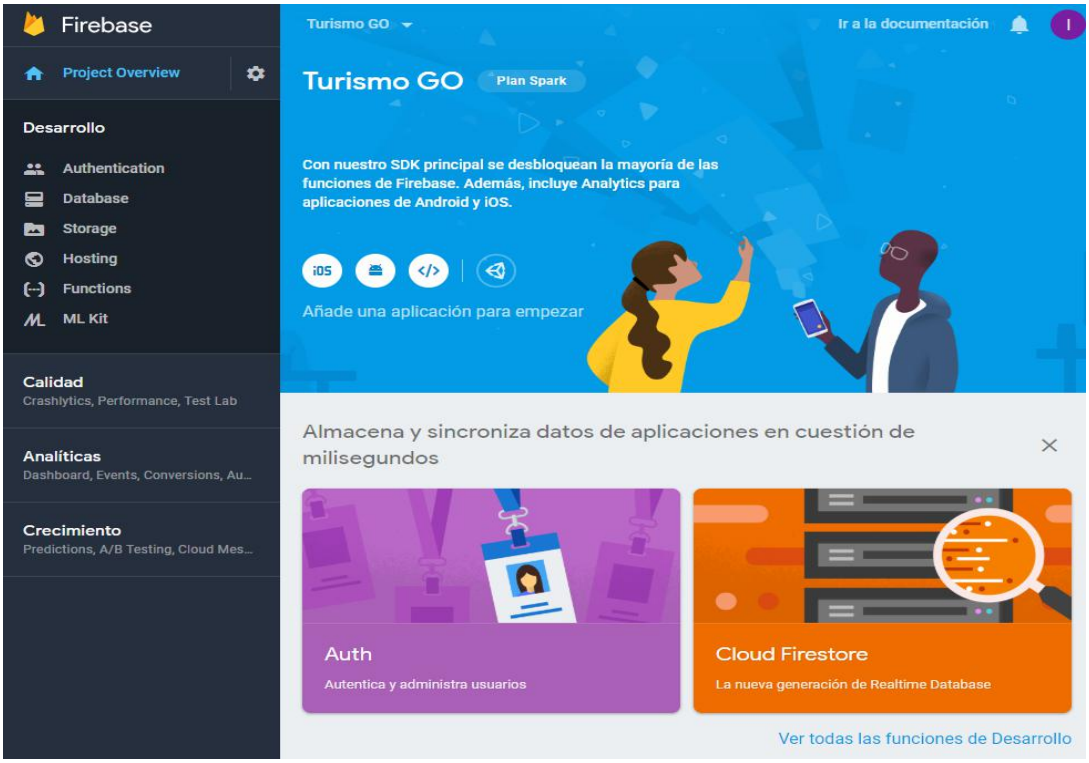
```
applyHaversine(locations) {
  console.log(locations)

  let usersLocation = { lat: 0, lng: 0 };
  var options = { enableHighAccuracy: true, timeout:10000 };

    this.geolocation.getCurrentPosition(options).then((position) => {

    usersLocation = { lat: position.coords.latitude, lng: position.coords.longitud

    }).catch((error) => {
    console.log('Error getting location', JSON.stringify(error));
    });

  locations.map((location) => {

    let placeLocation = { lat: location.latitude, lng: location.longitude };

    location.distance = this.getDistanceBetweenPoints(
      usersLocation,
      placeLocation,
      'km'
    ).toFixed(1);
  });
  return locations;
}
```

5.2.4 Iniciación de Firebase

Firebase nos proporciona un proveedor de autenticación, ahí guardaremos los usuarios que se vayan registrando (sólo debemos configurar el mdominio y URL que nos asigna Firebase a la App). Así que sólo necesitaremos personalizar las pestañas de login y register para que todo funcione a la perfección.

Se instala Firebase y importamos los módulos de AngularFire2.

Y entonces creamos las páginas del login y register
Firebase (entre otras cosas) nos muestra una tabla con los usuarios registrados en nuestra App.



5.2.5 Login con Firebase

En ***login.ts*** tendremos loginUser(), goToSignup() y goToResetPassword().
Cuando el usuario envíe el formulario (Form) desde nuestra App, la info será registrada en Firebase. Usamos el método de login proveido por Firebase. Y goToSignup(), goToResetPassword() las usaremos para la navegación entre páginas.

```
export class LoginPage {

  myForm: FormGroup;
  user: Observable<firebase.User>;
  public loading:Loading;

  constructor(
    public navCtrl: NavController,
    public formBuilder: FormBuilder,
    public afAuth: AngularFireAuth,
    public alertCtrl: AlertController,
    public loadingCtrl: LoadingController
  ) {
    this.myForm = this.formBuilder.group({
      email: ['', Validators.required],
      password: ['', Validators.required]
    });
    this.user = afAuth.authState;
  }

  loginUser(){
    console.log("Email:" + this.myForm.value.email);
    console.log("Password:" + this.myForm.value.password);
    this.afAuth.auth.signInWithEmailAndPassword(this.myForm.value.email, this.myForm.value.password).then(() => {
      console.log("User logging");
      this.navCtrl.setRoot('HomePage');
    }, (err) => {
      this.loading.dismiss().then( () => {
        let alert = this.alertCtrl.create({
          message: err.message,
          buttons: [
            {
              text: "Ok",
              role: 'cancel'
            }
          ]
        });
        alert.present();
      });
    });

    this.loading = this.loadingCtrl.create({
      dismissOnPageChange: true,
    });
    this.loading.present();
  }

  goToSignup(){
    this.navCtrl.push('SignupPage');
  }
  goToResetPassword(){
```

Acorde a la construcción de la página del login, está el html correspondiente para dar formato y crear los botones de “Email”, “Password”.

5.2.6 Register con Firebase

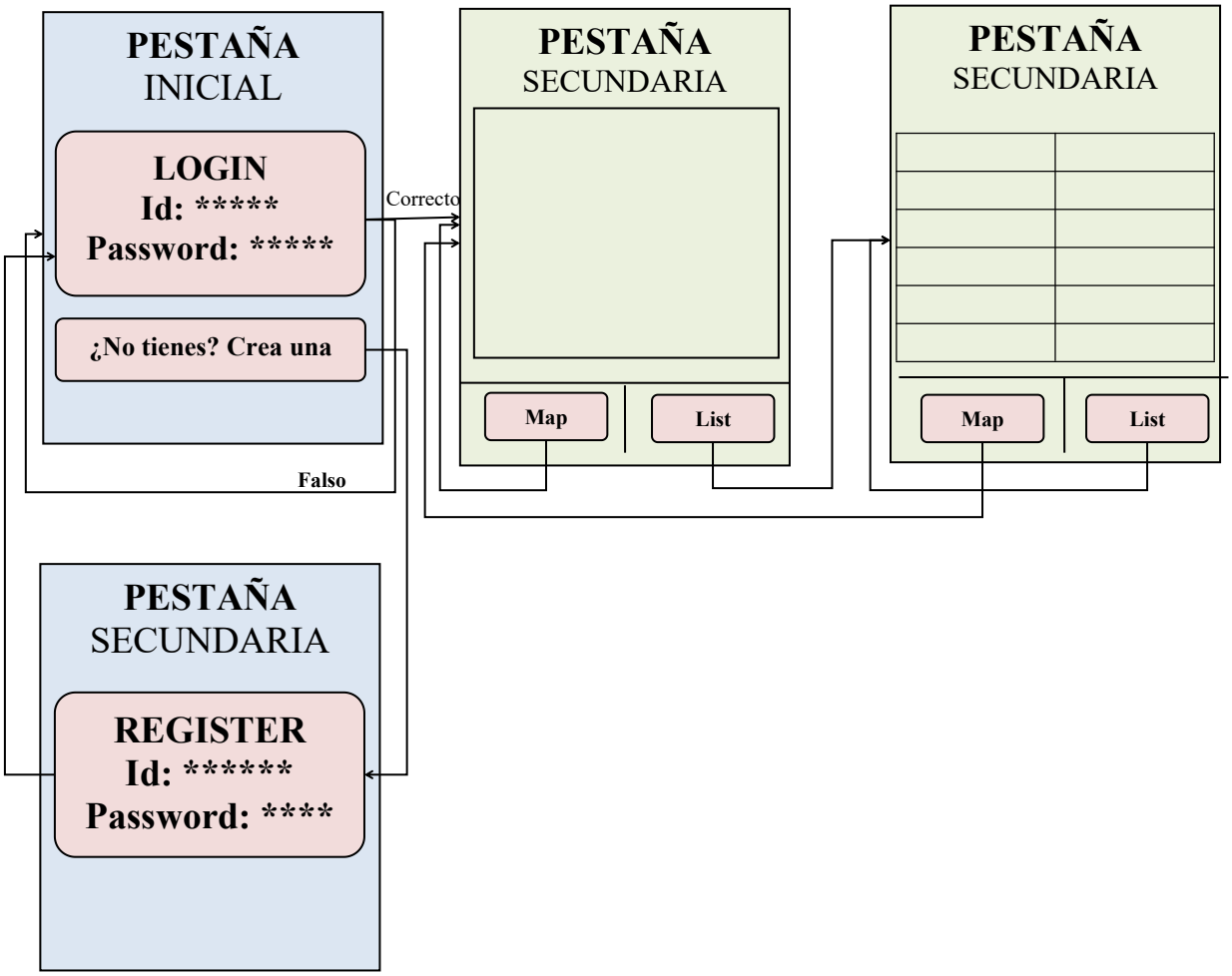
En cuanto al register (***signup.ts***) creamos la función signup() donde aplicamos el método createUserWithEmailAndPassword desde Firebase.

```
@Component({
  selector: 'page-signup',
  templateUrl: 'signup.html',
})
export class SignupPage {

  myForm: FormGroup;
  public loading:Loading;
  constructor(
    public navCtrl: NavController,
    public formBuilder: FormBuilder,
    public afAuth: AngularFireAuth,
    public alertCtrl: AlertController,
    public loadingCtrl: LoadingController
  ) {
    this.myForm = this.formBuilder.group({
      email: ['', Validators.required],
      password: ['', Validators.required]
    });
  }

  signup(){

    console.log("Email:" + this.myForm.value.email);
    console.log("Password:" + this.myForm.value.password);


    this.afAuth.auth.createUserWithEmailAndPassword(this.myForm.value.email, this.myForm.value.pass
    .then(
      res => {
        this.navCtrl.setRoot('HomePage');
      }, error => {
        this.loading.dismiss().then( () => {
          let alert = this.alertCtrl.create({
            message: error.message,
            buttons: [
              {
                text: "Ok",
                role: 'cancel'
              }
            ]
          });
          alert.present();
        });
      });

      this.loading = this.loadingCtrl.create({
        dismissOnPageChange: true,
      });
      this.loading.present();
```

Y con esto, acaba el desarrollo fundamental de TurismoGO. En app.component.ts configuramos que la página inicial al acceder a la App sea la del Login y desde ahí tendremos el árbol de la App montado.

A continuación vamos a ver un esquema final de los resultados de la App.

5.3 Mockup de TurismoGO



6. Conclusiones

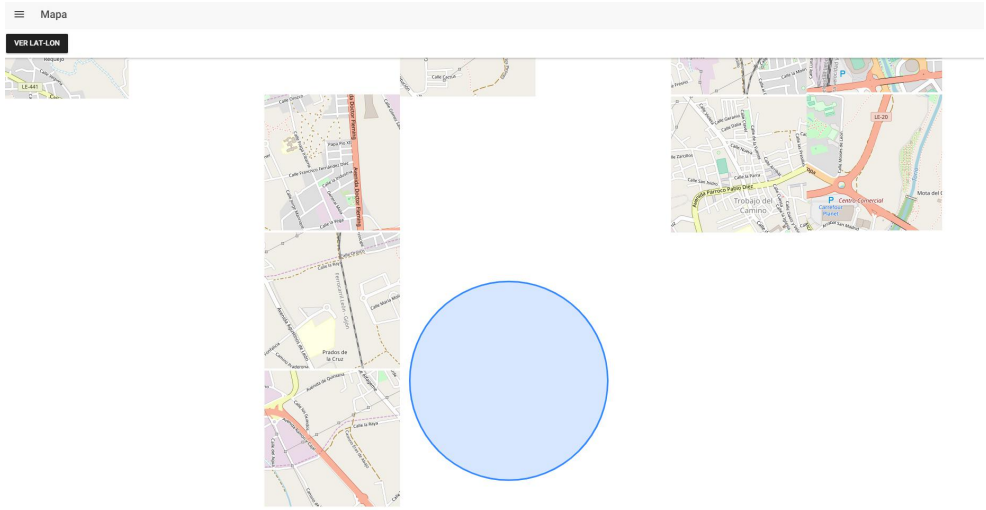
Empezaré por las experiencias obtenidas y luego por los inconvenientes en el desarrollo.

Trabajar con Frameworks que te permiten utilizar CSS HTML y Javascript de manera conjunta y sencilla es genial. Y la ventaja de poder programar para dos lenguajes a la vez con Ionic es mejor aún. Sin duda haré más aplicaciones con esta plataforma.

La tecnología avanza rápido en el campo de las App. Además, crean mucho software especializado al desarrollo de

Agradezco mucho la ayuda de Vexiza orientándome a que herramientas y tecnologías usar, pues he aprendido muchas cosas útiles ya sea para conocimiento personal como salida al mercado laboral.

En cuanto a los inconvenientes, me habría encantado (y de hecho, me arrepiento) haber creado el mapa a través de las librerías gratuitas y más personalizables de *Leaflet*. De hecho, creé la app (sin login y register) en Noviembre con Leaflet. Había personalizado los mapas (estilo PokémonGO) con OpenStreetMaps pero por alguna extraña razón el mapa salía erróneo, los cuadrados mal ubicados y dadas malas fechas. Aquí dejo una captura.



Así que acabé intentándolo con Google Maps (después de indagar que era gratuito hasta cierto beneficio), con lo que me creé una API KEY, apliqué las funciones correspondientes y funcionó a la primera.

7. Referencias

https://ionicframework.com/
https://www.youtube.com/watch?v=zXH4X3mZiik
https://www.uno-de-piera.com/primera-aplicacion-ionic-3-desde-0/
https://guiadev.com/manual-de-ionic-framework-parte-04/
https://www.npmjs.com/package/@ionic-native/google-maps
https://blog.ng-classroom.com/blog/ionic2/google-maps-native/
https://stackoverflow.com/questions/41674770/ionic-v2-how-to-insert-a-logo-in-header
https://firebase.google.com/docs/auth/web/password-auth?hl=es-419