

APLICACIONES MÓVILES MULTIPLATAFORMA

LABORATORIO N° 10

Integración con Maps



Alumno(s):					Nota	
Grupo:			Ciclo:V			
Criterio de Evaluación	Excelente (4pts)	Bueno (3pts)	Requiere mejora (2pts)	No accept. (0pts)	Puntaje Logrado	
Agregar react-native-maps						
Personalizar Google Maps						
Consumir APIs nativas de Geolocation						
Realiza con éxito lo propuesto en el laboratorio						
Es puntual y redacta el informe adecuadamente						

Laboratorio 10: Integración con Maps

Objetivos:

Al finalizar el laboratorio el estudiante será capaz de:

- Entender el funcionamiento del componente react-native-maps
- Desarrollar aplicaciones web enfocadas a componentes
- Manejar las APIs nativas del dispositivo, como Geolocation

Seguridad:

- Ubicar maletines y/o mochilas en el gabinete del aula de Laboratorio.
- No ingresar con líquidos, ni comida al aula de Laboratorio.
- Al culminar la sesión de laboratorio apagar correctamente la computadora y la pantalla, y ordenar las sillas utilizadas.

Equipos y Materiales:

- Una computadora con:
 - Windows 7 o superior
 - VMware Workstation 10+ o VMware Player 7+
 - Conexión a la red del laboratorio
- Máquinas virtuales:
 - Windows 7 Pro 64bits Español - Plantilla
- Instalador de node.js

Procedimiento:

Lab Setup

1. Configuración de proyecto

- 1.1. Copie el contenido del laboratorio 9 (la clase anterior) a excepción de la carpeta node_modules en una nueva carpeta llamada lab10 y reinstale todas las dependencias:

```
>npm install
```

- 1.2. En la nueva carpeta lab10, instalaremos las siguientes dependencias:

```
>npm install --save react-native-floating-action
```

```
>npm install --save react-native-maps
```

```
>react-native link react-native-maps
```

- 1.3. Modificaremos AndroidManifest.xml ubicado en la ruta "android\app\src\main" para agregar dos cosas importantes: solicitaremos al celular acceso a la geo localización y también estableceremos el token de Google Maps a utilizar.

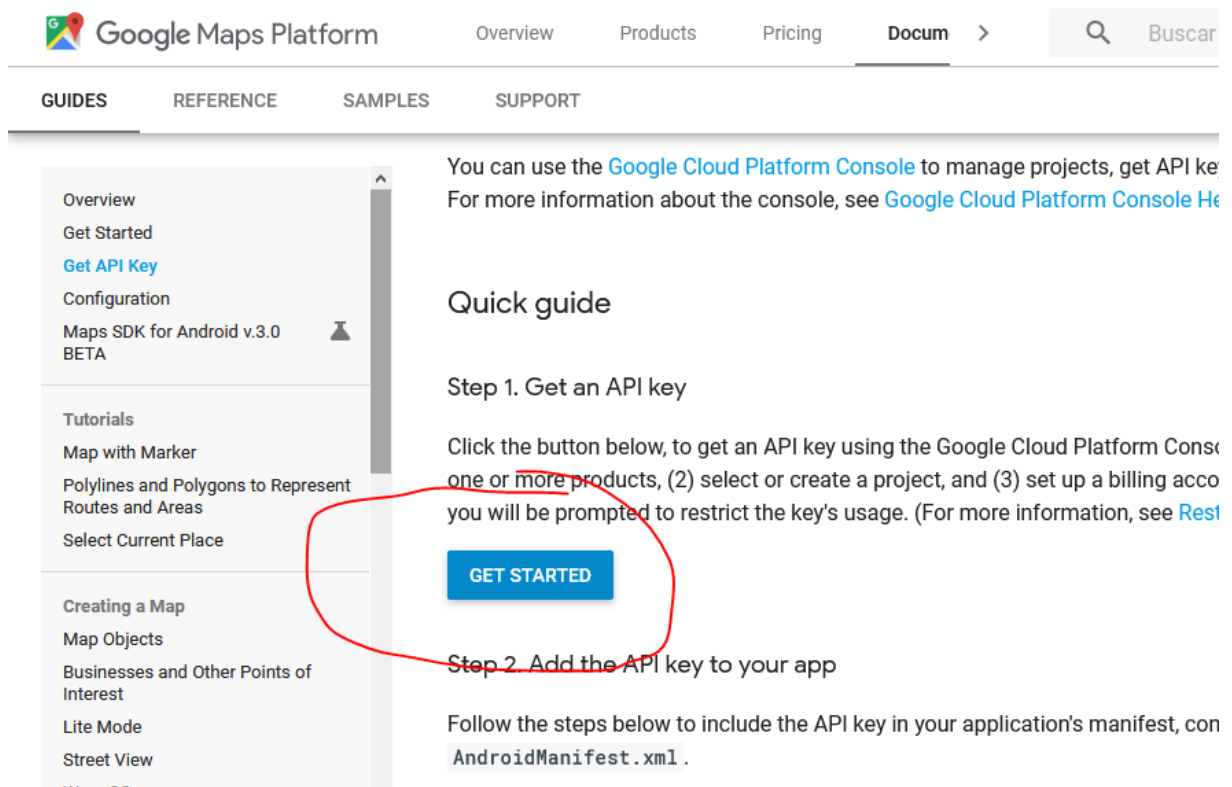
```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.lab07">

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

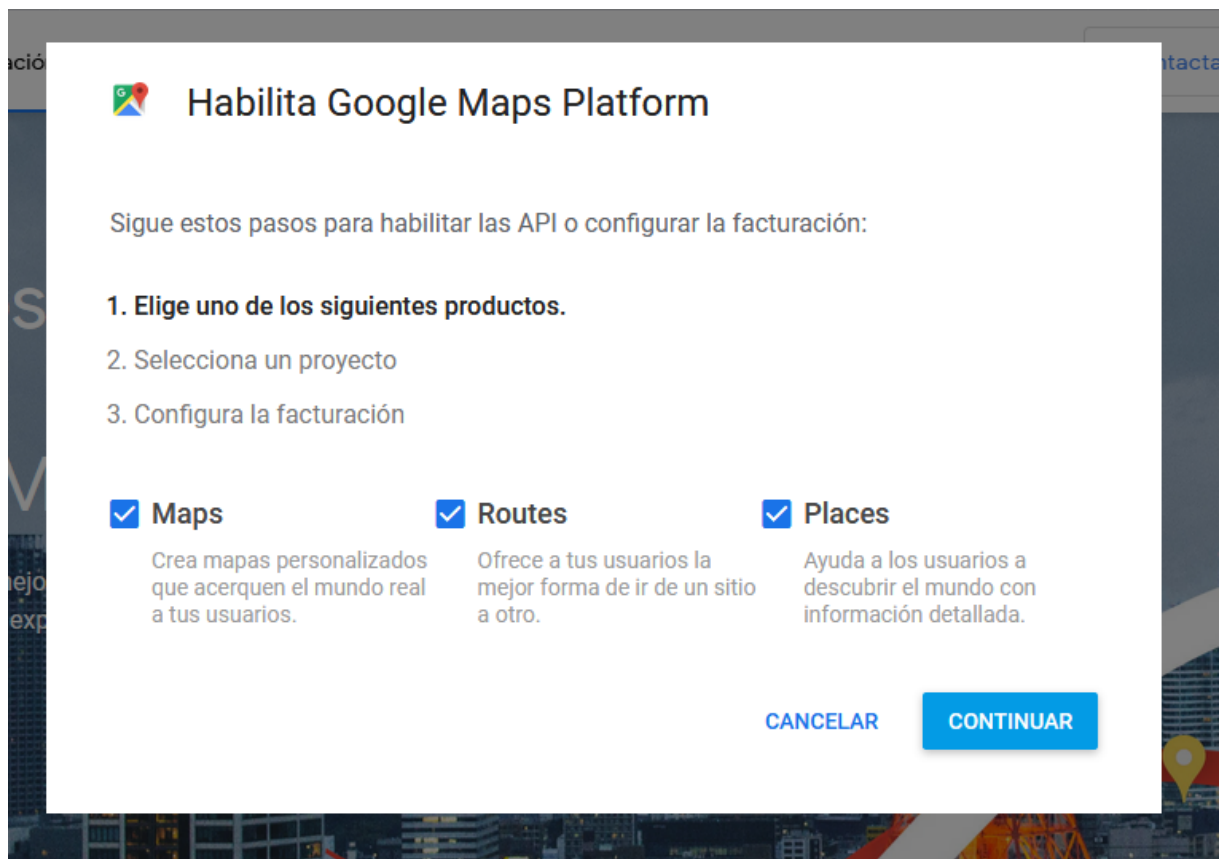
    <application
        android:name=".MainApplication"
        android:label="@string/app_name"
        android:icon="@mipmap/ic_launcher"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:allowBackup="false"
        android:theme="@style/AppTheme">
        <meta-data
            android:name="com.google.android.geo.API_KEY"
            android:value="AIzaSyBn3IR0zbtK91PzKu8tzFaViNArBRO4weweTsS" />
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name"
            android:configChanges="keyboard|keyboardHidden|orientation|screenSize"
            android:windowSoftInputMode="adjustResize">
```

El token mostrado aquí es un token de ejemplo. Usted debe utilizar uno que ya tenga o en todo caso, obtener uno nuevo desde la siguiente página:

<https://developers.google.com/maps/documentation/android-api/signup>



The screenshot shows the Google Maps Platform documentation page. The left sidebar contains a navigation menu with sections like Overview, Get Started, Get API Key, Configuration, Maps SDK for Android v.3.0 BETA, Tutorials, and Creating a Map. The main content area is titled 'Quick guide' and includes 'Step 1. Get an API key' and 'Step 2. Add the API key to your app'. A red circle highlights the 'GET STARTED' button in Step 1. The text in Step 1 mentions using the Google Cloud Platform Console to manage projects and get API keys, and Step 2 mentions adding the API key to the application's manifest.



2. Agregar opción de vista Mapa en Chat

2.1. Agregaremos Map a nuestro App.js para que el router tenga acceso a dicha vista,

```
import HomeScreen from './src/screens/Home/Home';
import ChatScreen from './src/screens/Chat/Chat';
import Location from './src/screens/Location/Location';
import Camera from './src/screens/Camera/Camera';
import Map from './src/screens/Map/Map';

const AppStack = createDrawerNavigator({
  Home: HomeScreen,
  Chat: ChatScreen,
  Location: Location,
  Camera: Camera,
  Map: Map
});

const AuthStack = createBottomTabNavigator({
  SignIn: SignInScreen,
  SignUp: SignUpScreen
});
```

- 2.2. Modificaremos el método render dentro del archivo Chat.js para agregar un botón que invoque a la vista de mapa.

```
render() {
  const user = { _id: this.state.userId || -1 };
  return (
    <Fragment>
      <Modal
        animationType="slide"
        transparent={false}
        visible={this.state.modalVisible}
      >
        <View>
          <Button
            onPress={this.chatHandler}
            title="Regresar al Chat"
            color="#841584"
          />
          <Button
            onPress={this.cameraHandler}
            title="Tomar foto"
            color="green"
          />
          <Button
            onPress={this.mapHandler}
            title="Compartir Ubicación"
            color="yellow"
          />
          <Button
            onPress={this.backHandler}
            title="Regresar al Inicio"
            color="red"
          />
        </View>
      </Modal>
      <GiftedChat
        placeholder="Escribe algo..."
        renderActions={() => {
          return (

```

- 2.3. En el mismo archivo, agregaremos un método mapHandler para navegar a dicha vista de mapa. No se olvide de que antes de navegar entre vistas, debemos desmontar el modal, por eso que primero modificaremos el state.

```
cameraHandler = () => {
  this.setState({ modalVisible: false }, () => {
    console.log('si hace click');
    this.props.navigation.navigate('Camera');
    //CameraRoll.saveToCameraRoll('file:///sdcard/img.png', 'photo');
  });
};

mapHandler = () => {
  this.setState({ modalVisible: false }, () => {
    this.props.navigation.navigate('Map');
  });
};

render() {
  const user = { _id: this.state.userId || -1 };
  return (
    <Fragment>
      <Modal
        animationType="slide"

```

this.setState recibe como segundo argumento una función de callback, es decir, una función que se ejecutará solamente cuando el estado se haya actualizado. Así nos aseguramos que nuestro modal estará oculto para el momento de la navegación.

3. Obtención de permisos de Geo Localización

- 3.1. Crearemos la carpeta Map dentro de screens y a su vez, el archivo Map.js con el siguiente contenido. (toda esta sección es un solo archivo, pero se harán comentarios en cada parte para detallar lo que acontece)

```
import React, { Component } from 'react';
import {
  View,
  Text,
  PermissionsAndroid,
  StyleSheet,
  Dimensions
} from 'react-native';
import MapView from 'react-native-maps';

const { width, height } = Dimensions.get('window');

const ASPECT_RATIO = width / height;
const LATITUDE_DELTA = 0.0922;
const LONGITUDE_DELTA = LATITUDE_DELTA * ASPECT_RATIO;

class MapStyle extends Component {
  state = {
    granted: false,
    data: null,
    latitude: null,
    longitude: null
  };
  componentDidMount() {
    this.requestGpsPermission();
  }
}
```

Hemos declarado nuestro componente y hacemos uso de Dimensions, que es un objeto nativo de React Native que nos permite acceder a las dimensiones del dispositivo. Gracias a él, hacemos una pequeña fórmula entre ancho y alto (**width/height**) para poder obtener un ratio de aspecto acorde a la pantalla.

```
requestGpsPermission = async () => {
  try {
    const granted = await PermissionsAndroid.request(
      PermissionsAndroid.PERMISSIONS.ACCESS_FINE_LOCATION,
      {
        title: 'Permiso para geolocalización',
        message: 'Necesitamos tu permiso para mostrar tu posición.',
        buttonNeutral: 'Preguntarme luego',
        buttonNegative: 'Cancelar',
        buttonPositive: 'OK'
      }
    );
    if (granted === PermissionsAndroid.RESULTS.GRANTED) {
      console.log('You can use the geolocation');
      this.setState({ granted: true }, this.getPosition);
    } else {
      console.log('Geolocation permission denied');
      this.setState({ granted: false });
    }
  } catch (err) {
    console.warn(err);
  }
};
```

Luego declaramos la función **requestGpsPermission**, que ha sido invocada en el **componentDidMount**. Esta solicitará permiso al usuario para acceder a su geolocalización. Existen APIs del celular que no estarán disponibles si no se solicita permiso antes. El listado de ellas está en el siguiente enlace:

<https://facebook.github.io/react-native/docs/permissionsandroid>

```

getPosition = () => {
  navigator.geolocation.getCurrentPosition(
    data => {
      console.log('data', data);
      this.setState({
        data: data,
        latitude: data.coords.latitude,
        longitude: data.coords.longitude
      });
    },
    error => {
      console.log('Error', error);
      alert('Hubo un error al obtener la geolocalizacion!');
    }
  );
};

```

La función **getPosition** accede a la variable global navigation que podrá acceder al mismo tiempo a la geolocalización del dispositivo. En esta ocasión solamente pediremos la latitud y longitud al momento de ser ejecutada, pero tenga en cuenta que hay métodos como `watchPosition` que permite crear una función de seguimiento según el usuario se mueve. La documentación de dicha API se encuentra en el siguiente enlace:

<https://facebook.github.io/react-native/docs/geolocation>

```

render() {
  return this.state.granted ? (
    this.state.data && (
      <View style={styles.container}>
        <MapView
          provider={'google'}
          style={styles.map}
          initialRegion={{
            latitude: this.state.latitude,
            longitude: this.state.longitude,
            latitudeDelta: LATITUDE_DELTA,
            longitudeDelta: LONGITUDE_DELTA
          }}
        />
      </View>
    )
  ) : (
    <Text>No obtuvimos permiso :(</Text>
  );
}

```

En el método **render** nos aseguramos de renderizar solamente si tenemos el permiso del usuario otorgado, caso contrario, mostramos un mensaje.

```

const styles = StyleSheet.create({
  container: {
    ...StyleSheet.absoluteFillObject,
    justifyContent: 'flex-end',
    alignItems: 'center'
  },
  map: {
    ...StyleSheet.absoluteFillObject
  }
});

export default MapStyle;

```

Finalmente, declaramos los estilos necesarios para que el mapa llene toda la pantalla. En sus proyectos puede decidir cómo se verá el mapa, en este caso llenaremos toda la pantalla por lo que utilizamos las propiedades dentro de **StyleSheet.absoluteFillObject**. Debemos obtener la siguiente vista renderizada.



4. Agregar marcador para indicar posición

- 4.1. Es muy común en aplicaciones de mapas ver como un marcador resalta nuestra ubicación o hasta nos permite setearla. Para lograr esto, importaremos el componente **Marker** del módulo **react-native-maps**

```
import MapView, { ProviderPropType, Marker } from 'react-native-maps';
```

- 4.2. A continuación, modificaremos el método **render**, específicamente la parte de **MapView** para agregarle un **children**, en este caso, el **Marker** indicando nuestra posición actual. Dicho componente cuenta con el evento **onDragEnd**, que nos entregará las coordenadas de la nueva posición una vez arrastrado a un lugar en el mapa.

```
<View style={styles.container}>
  <MapView
    provider={'google'}
    style={styles.map}
    initialRegion={{
      latitude: this.state.latitude,
      longitude: this.state.longitude,
      latitudeDelta: LATITUDE_DELTA,
      longitudeDelta: LONGITUDE_DELTA
    }}
  >
    <Marker
      draggable
      coordinate={{
        latitude: this.state.latitude,
        longitude: this.state.longitude
      }}
      onDragEnd={e => {
        console.log(e.nativeEvent.coordinate);
        this.setState({
          latitude: e.nativeEvent.coordinate.latitude,
          longitude: e.nativeEvent.coordinate.longitude
        });
      }}
    />
  </MapView>
</View>
```


5. Agregando estilos personalizados
 - 5.1. Descargue el archivo customStyles.js que se encuentra adjunto en este módulo, copie su contenido (verá que es una constante con muchas propiedades) y péguela en la declaración de constantes de su archivo.

```
const ASPECT_RATIO = width / height;
const LATITUDE_DELTA = 0.0922;
const LONGITUDE_DELTA = LATITUDE_DELTA * ASPECT_RATIO;

const customStyle = [
  {
    elementType: 'geometry',
    stylers: [
      {
        color: '#242f3e'
      }
    ]
  },
]
```

- 5.2. Agregaremos la propiedad **customMapStyle** a nuestro **MapView**.

```
<MapView
  provider={'google'}
  style={styles.map}
  initialRegion={{
    latitude: this.state.latitude,
    longitude: this.state.longitude,
    latitudeDelta: LATITUDE_DELTA,
    longitudeDelta: LONGITUDE_DELTA
  }}
  customMapStyle={customStyle}
```

Lo que hemos hecho nos permite controlar el fondo del mapa, color de las calles, avenidas, tráfico, etc, creando un resultado personalizado para nuestra aplicación.

6. Botón flotante
 - 6.1. Nuestro mapa no permite regresar a la vista anterior de una forma explícita en la aplicación (obviamente regresaremos presionando el botón regresar de la aplicación, pero esto no es una buena experiencia de usuario). Agregaremos las siguientes importaciones en nuestra clase **Map**

```
import {
  Platform,
  View,
  Text,
  PermissionsAndroid,
  StyleSheet,
  Dimensions
} from 'react-native';
import MapView, { ProviderPropType, Marker } from 'react-native-maps';
import { FloatingAction } from 'react-native-floating-action';
import Icon from 'react-native-ionicons';
```

- 6.2. Declaramos la constante actions, que nos permitirá declarar las opciones que deseamos mostrar al usuario.

```

cancelHandler = () => this.props.navigation.navigate('Chat');
shareHandler = () => this.props.navigation.navigate('Chat');
render() {
  const actions = [
    {
      text: 'Compartir ubicación',
      name: 'bt_share',
      color: 'green',
      icon: (
        <Icon
          name={Platform.OS === 'ios' ? 'ios-share' : 'md-share'}
          color="#fff"
          size={25}
        />
      ),
      position: 1
    },
    {
      text: 'Regresar',
      name: 'bt_cancel',
      color: 'red',
      icon: (
        <Icon
          name={Platform.OS === 'ios' ? 'ios-close' : 'md-close'}
          color="#fff"
          size={25}
        />
      ),
      position: 3
    }
  ];
  return this.state.granted ? (
    this.state.data && (
      <View style={styles.container}>
        <MapView

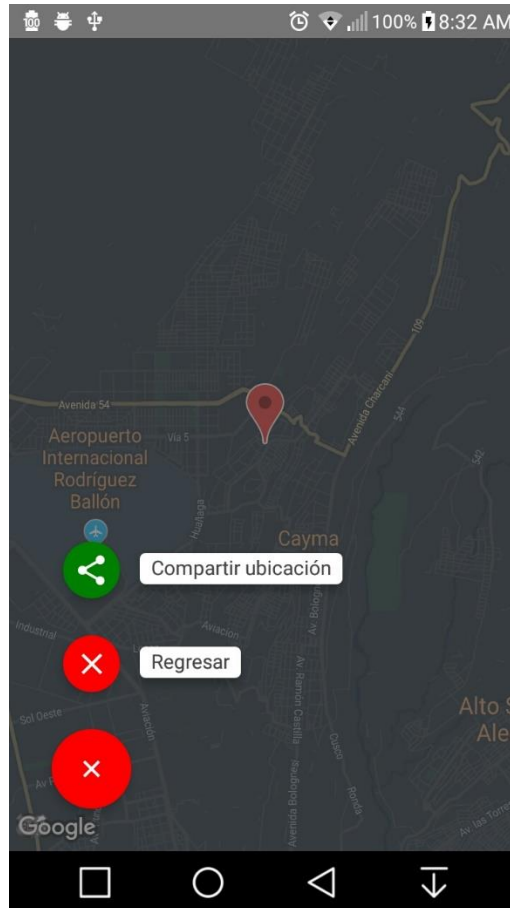
```

- 6.3. Finalmente, agregaremos el componente FloatingAction, que agregará un botón flotante a la vista con las opciones ocultas hasta que el usuario decide interactuar con ellas.

```

      latitude: e.nativeEvent.coordinate.latitude,
      longitude: e.nativeEvent.coordinate.longitude
    });
  });
}
/>
</MapView>
<FloatingAction
  position="left"
  color="red"
  actions={actions}
  onPressItem={name => {
    if (name === 'bt_share') {
      this.shareHandler();
    } else if (name === 'bt_cancel') {
      this.cancelHandler();
    }
  }}
/>
</View>
)

```



7. Finalizar la sesión

- 7.1. Apagar el equipo virtual
- 7.2. Apagar el equipo

Conclusiones:

Indicar las conclusiones que llegó después de los temas tratados de manera práctica en este laboratorio.