Desarrollo de una Aplicacion para monitoreo de sensor DHT11

Ing. Alfredo Valdés Cárdenas

12 de mayo de 2018

Como instalar Ionic

NOTA: Para poder utilizar Ionic se debe de tener Node.Js instalado
Abrimos una terminal con permisos de administrador y ejecutamos el siguiente comando:

```
npm install -g ionic@latest cordova@latest
```

Una vez terminado, Ionic queda instalado. Para crear una aplicación corremos el siguiente comando:

```
ionic start appDiplomado blank
```

Nos aparecerá una serie de preguntas, la primera:

```
Would you like to integrate your new app with Cordova
to target native iOS and Android? (y/N)
```

Presionamos N y después Enter para no incluir Cordova, la siguiente pregunta:

```
Install the free Ionic Pro SDK and connect your app? (Y/n)
```

Respondemos que No y con esto la plantilla de aplicación queda generada dentro de una carpeta con el nombre de la misma aplicación, en este caso «appDiplomado». Entramos a la carpeta de la aplicación con:

```
cd appDiplomado
```

y ejecutamos la aplicación con el siguiente comando:

```
ionic serve --lab
```

A continuacion, se abrirá la aplicación en una ventana del navegador

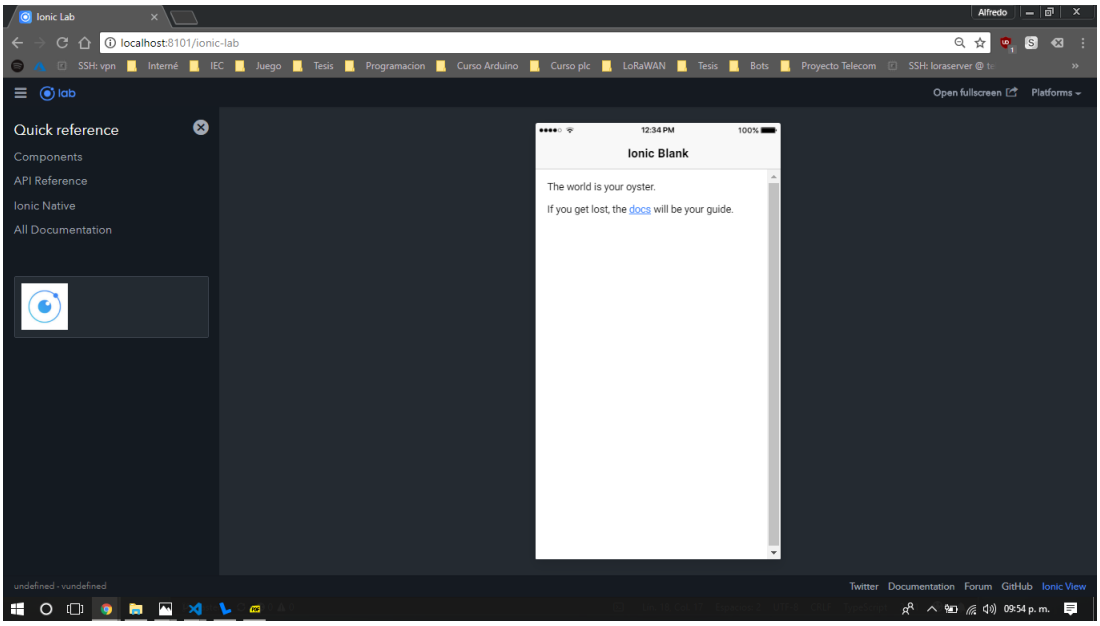

Figura 1: Vista en vivo de la aplicación

Desarrollo de la aplicación

Sin cerrar la ventana de la vista previa, abrimos en visual studio code la carpeta de la aplicación generada. Se nos presenta la siguiente ventana:

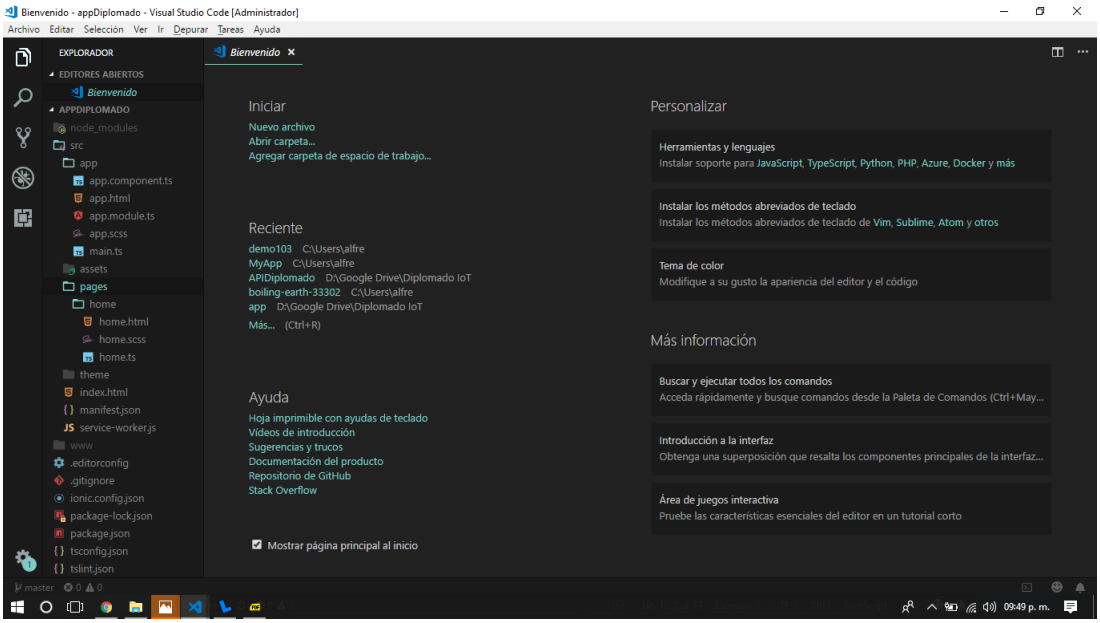

Figura 2: Ventana inicial de VS Code

En la barra de la izquierda podremos ver la estructura del proyecto, que ya contiene los archivos generados en la sección anterior. Ubicamos dentro de la carpeta «src» la carpeta «pages» y dentro de esta la carpeta «home». Abrimos el archivo «home.html» y podemos encontrar el código HTML de nuestra app. Editamos el titulo de la app que originalmente dice «Ionic Blank» por el texto «Hola Diplomado!» y guardamos el archivo. Sin hacer nada mas la ventana del navegador refleja los cambios al código de la aplicación. Esta es la función de recarga en vivo de Ionic, que funcionara siempre que el comando «ionic serve» se este ejecutando.

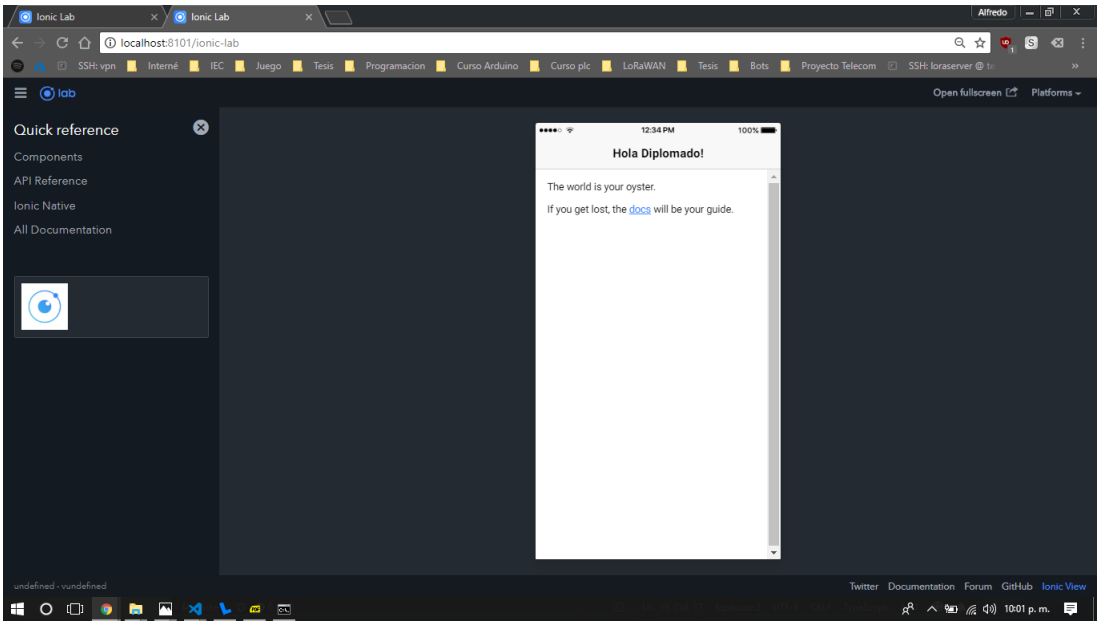

Figura 3: Recarga en vivo de Ionic

Importar el modulo HTTPClientModule

Ahora debemos agregar HttpClientModule en nuestro archivo app.module.ts, esto nos permitirá realizar peticiones a un servidor.
Dentro de app.module.ts:

```
...
import { HttpClientModule } from '@angular/common/http';

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

Crear un proveedor de datos

Para manejar la conexión con el servidor datos vamos a utilizar un provider. Los providers son proveedores que se encargan del manejo de datos, bien extraídos de la base de datos, desde una API REST, etc.
Detenemos la ejecucion de la recarga en vivo y ejecutamos el siguiente comando para generar un nuevo proveedor:

```
ionic g provider servidor
```

Ionic creará un archivo para nuestro servicio que estará en «src/providers/servidor.ts». A su vez, en app.module.ts se importo el nuevo proveedor y se agrego a las declaraciones del modulo.

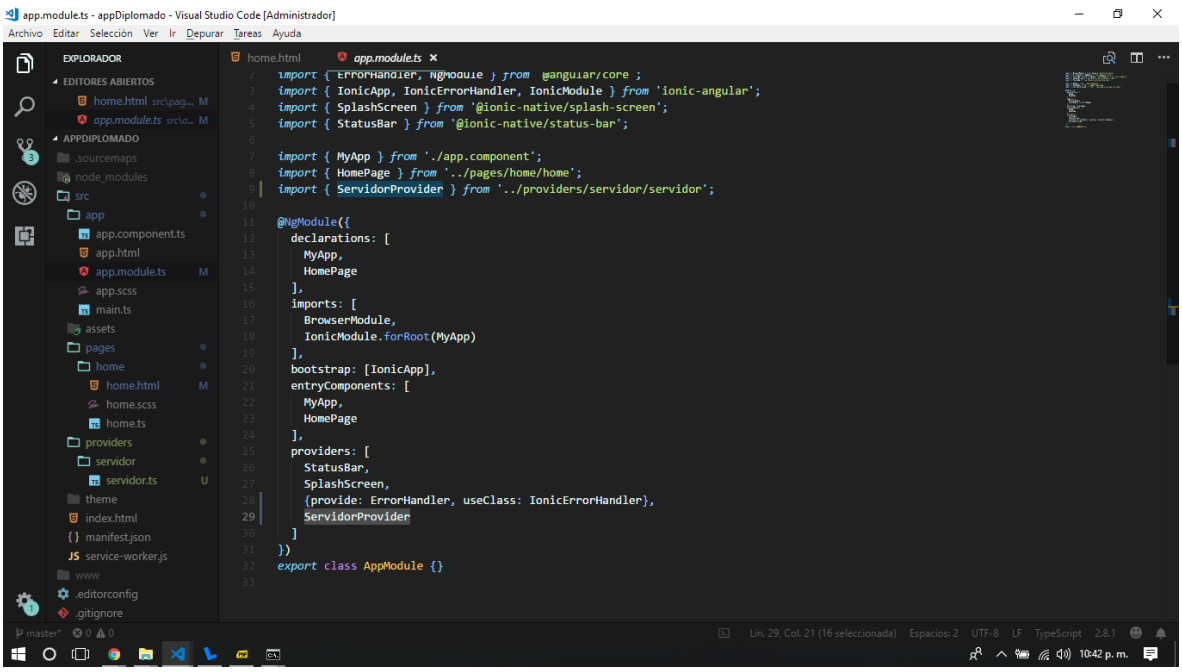

Figura 4: Proveedor instalado

Agregar la llamada a la API del servidor

Nuestra API expone los datos del servidor a través de una petición GET a la dirección:
`https://apidiplomado.herokuapp.com/getdata?device=Device&results=1`
Donde el parámetro *Device* corresponde al nombre del dispositivo y el parámetro *results* entrega el numero de registros que se quiere obtener, en este caso solo es uno. Esta llamada responde con un objeto JSON:

```
{"1526091822":{"hin":"22","hum":"10","temp":"21"}}
```

El numero identificador del arreglo corresponde a la fecha y hora en la que se registro el dato, el parámetro «hin» indica la sensación térmica, «hum» el porcentaje de humedad en el aire y «temp» la temperatura.
Para realizar la llamada a la API crearemos un nuevo método en la clase del proveedor:

```
getData(){
  return this.http.get('https://apidiplomado.herokuapp.com/getdata?device=Device&results=1');
}
```

De manera que el archivo «servidor.ts» quedaría de la siguiente manera:

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';

@Injectable()
export class ServidorProvider {

  constructor(public http: HttpClient) {
  }
  getData() {
    return this.http.get(
                'https://apidiplomado.herokuapp.com/getdata?device=Device&results=1'
        );
  }
}
```

Inyectando los datos en la pantalla

Ahora, abrimos el archivo «app/pages/home/home.ts». En este archivo programaremos la lógica de nuestra pantalla «home.html».
Primero importamos el proveedor y creamos una instancia dentro del constructor de la clase.

```
import { Component } from '@angular/core';
import { NavController } from 'ionic-angular';
import { ServidorProvider } from '../../providers/servidor/servidor';
@Component({
  selector: 'page-home',
  templateUrl: 'home.html'
})
export class HomePage {
  constructor(public navCtrl: NavController, public servidor: ServidorProvider) {
  }
   sensors: any[] = [];
}
```

Ahora, las llamadas a la API se realizan en dos situaciones: Cuando abrimos la aplicación por primera vez y cuando actualizamos la pantalla, por lo que debemos crear dos métodos. Cuando abrimos la aplicación, Ionic nos permite ejecutar código mandando a llamar el método ionViewDidLoad():

```
ionViewDidLoad() {//cuando se abra la aplicacion
    this.servidor.getData()//Llamar el metodo de la clase servidor
      .subscribe(//Las peticiones al servidor son asincronas, por
                 //lo que hay que esperar a que el servidor responda
        (data) => { // Si el servidor contesta, almacena la respuesta en la variable data
          var results = [];//Un arreglo nuevo para pasarle los datos a la aplicacion
          for (var i in data) {//iteramos sobre la respuesta
            results.push(data[i]); // y lo guardamos en la respuesta
          }
          this.sensors = results.reverse(); //giramos el arreglo y lo pasamos a la aplicacion
        },
        (error) => { //Si el servidor no responde
          console.error(error); //imprime el error en consola
        }
      )
  }
```

La lógica para actualizar los datos en pantalla es similar a la del método anterior, pero se le agrega un componente llamado *refresher* que le indica a la aplicación cuando debe de mostrar los datos nuevos en pantalla:

```
  doRefresh(refresher) {
    this.servidor.getData()//Llamar el metodo de la clase servidor
      .subscribe(//Las peticiones al servidor son asincronas, por
        //lo que hay que esperar a que el servidor responda
        (data) => { // Si el servidor contesta, almacena la respuesta en la variable data
          var results = [];//Un arreglo nuevo para pasarle los datos a la aplicacion
          for (var i in data) {//iteramos sobre la respuesta
            results.push(data[i]); // y lo guardamos en la respuesta
          }
          this.sensors = results.reverse(); //giramos el arreglo y lo pasamos a la aplicacion
          refresher.complete(); // le indicamos a la aplicacion que termine la actualizacion
        },
        (error) => { //Si el servidor no responde
          console.error(error); //imprime el error en consola
          refresher.complete();// le indicamos a la aplicacion que termine la actualizacion
        }
      )
  }
```

El código completo de la clase «home.ts» queda asi:

```
import { Component } from '@angular/core';
import { NavController } from 'ionic-angular';
import { ServidorProvider } from '../../providers/servidor/servidor';

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

  constructor(public navCtrl: NavController, public servidor: ServidorProvider) {

  }
  sensors: any[] = [];

  doRefresh(refresher) {
    this.servidor.getData()//Llamar el metodo de la clase servidor
      .subscribe(//Las peticiones al servidor son asincronas, por
```

```
        //lo que hay que esperar a que el servidor responda
        (data) => { // Si el servidor contesta, almacena la respuesta en la variable data
          var results = [];//Un arreglo nuevo para pasarle los datos a la aplicacion
          for (var i in data) {//iteramos sobre la respuesta
            results.push(data[i]); // y lo guardamos en la respuesta
          }
          this.sensors = results.reverse(); //giramos el arreglo y lo pasamos a la aplicacion
          refresher.complete(); // le indicamos a la aplicacion que termine la actualizacion
        },
        (error) => { //Si el servidor no responde
          console.error(error); //imprime el error en consola
          refresher.complete();// le indicamos a la aplicacion que termine la actualizacion
        }
      )
  }

  ionViewDidLoad() {//cuando se abra la aplicacion
    this.servidor.getData()//Llamar el metodo de la clase servidor
      .subscribe(//Las peticiones al servidor son asincronas, por
        //lo que hay que esperar a que el servidor responda
        (data) => { // Si el servidor contesta, almacena la respuesta en la variable data
          var results = [];//Un arreglo nuevo para pasarle los datos a la aplicacion
          for (var i in data) {//iteramos sobre la respuesta
            results.push(data[i]); // y lo guardamos en la respuesta
          }
          this.sensors = results.reverse(); //giramos el arreglo y lo pasamos a la aplicacion
        },
        (error) => { //Si el servidor no responde
          console.error(error); //imprime el error en consola
        }
      )
  }
}
```

Mostrando los datos

Regresamos a «home.html»

```
<ion-header>
  <ion-navbar>
    <ion-title>
      Monitor de Sensores Diplomado
    </ion-title>
  </ion-navbar>
</ion-header>

<ion-content padding>
  <ion-list>
    <!-- No se cargara la pagina hasta que se le asigne un valor a la variable sensors -->
    <ion-item *ngFor="let sensor of sensors">
      <div id="Temperatura">
        <!-- Creamos un boton con que abarque todo el ancho de la pantalla,
          con el click desactivado -->
        <button ion-button ng-disable="true" block>
          <!-- Le ponemos un icono con forma de termometro -->
          <ion-icon name="thermometer"></ion-icon>
              <!-- AQUI SE INYECTAN LOS DATOS DE LA API -->
```

```
          Temperatura {{ sensor.temp }}C
        </button>
      </div>
    </ion-item>
  </ion-list>
</ion-content>
```

El código es reusable, por lo que solo seria cuestión de cambiar las referencias de las variables de temperatura por las de humedad y sensación térmica. La documentación de Ionic nos muestra las distintas funciones para los objetos de botón, en `https://ionicframework.com/docs/api/components/button/Button/` . Para buscar mas iconos esta el catalogo de ion-icons `https://ionicframework.com/docs/ionicons/`

Agregar la función de actualización a la pagina

Para agregar la función de actualización a la pagina es necesario utilizar el componente *ionRefresher*.

```
<ion-content>
  <ion-refresher (ionRefresh)="doRefresh($event)">
    <ion-refresher-content></ion-refresher-content>
  </ion-refresher>
</ion-content>
```

Al momento de jalar la pantalla hacia abajo se activa el evento de actualización, el cual manda a llamar el método doRefresh que creamos en la clase «home.ts».
El código completo del archivo «home.html» es el siguiente:

```
<ion-header>
  <ion-navbar>
    <ion-title>
      Diplomado
    </ion-title>
  </ion-navbar>
</ion-header>
<ion-content padding>
    <ion-refresher (ionRefresh)="doRefresh($event)">
        <ion-refresher-content></ion-refresher-content>
      </ion-refresher>
  <ion-list>
    <!-- No se cargara la pagina hasta que se le asigne un valor a la variable sensors -->
    <ion-item *ngFor="let sensor of sensors">
      <div id="Temperatura">
        <!-- Creamos un boton con que abarque todo el ancho de la pantalla,
          con el click desactivado -->
        <button color="danger" ion-button ng-disable="true" block>
          <!-- Le ponemos un icono con forma de termometro -->
          <ion-icon name="thermometer"></ion-icon>
          <!-- AQUI SE INYECTAN LOS DATOS DE LA API -->
          Temperatura {{ sensor.temp }}C
        </button>
      </div>
      <div id="Humedad">
        <!-- Creamos un boton con que abarque todo el ancho de la pantalla,
            con el click desactivado -->
        <button color="primary" ion-button ng-disable="true" block>
          <!-- Le ponemos un icono con forma de termometro -->
          <ion-icon name="water"></ion-icon>
          <!-- AQUI SE INYECTAN LOS DATOS DE LA API -->
          Humedad {{ sensor.hum }}%
        </button>
```

```
      </div>
      <div id="SensacionTermica">
        <!-- Creamos un boton con que abarque todo el ancho de la pantalla,
             con el click desactivado -->
        <button color="secondary" ion-button ng-disable="true" block>
          <!-- Le ponemos un icono con forma de termometro -->
          <ion-icon name="sunny"></ion-icon>
          <!-- AQUI SE INYECTAN LOS DATOS DE LA API -->
          Sensacion Termica {{ sensor.hin }}C
        </button>
      </div>
    </ion-item>
  </ion-list>
</ion-content>
```

La aplicación debería de verse de la siguiente manera:

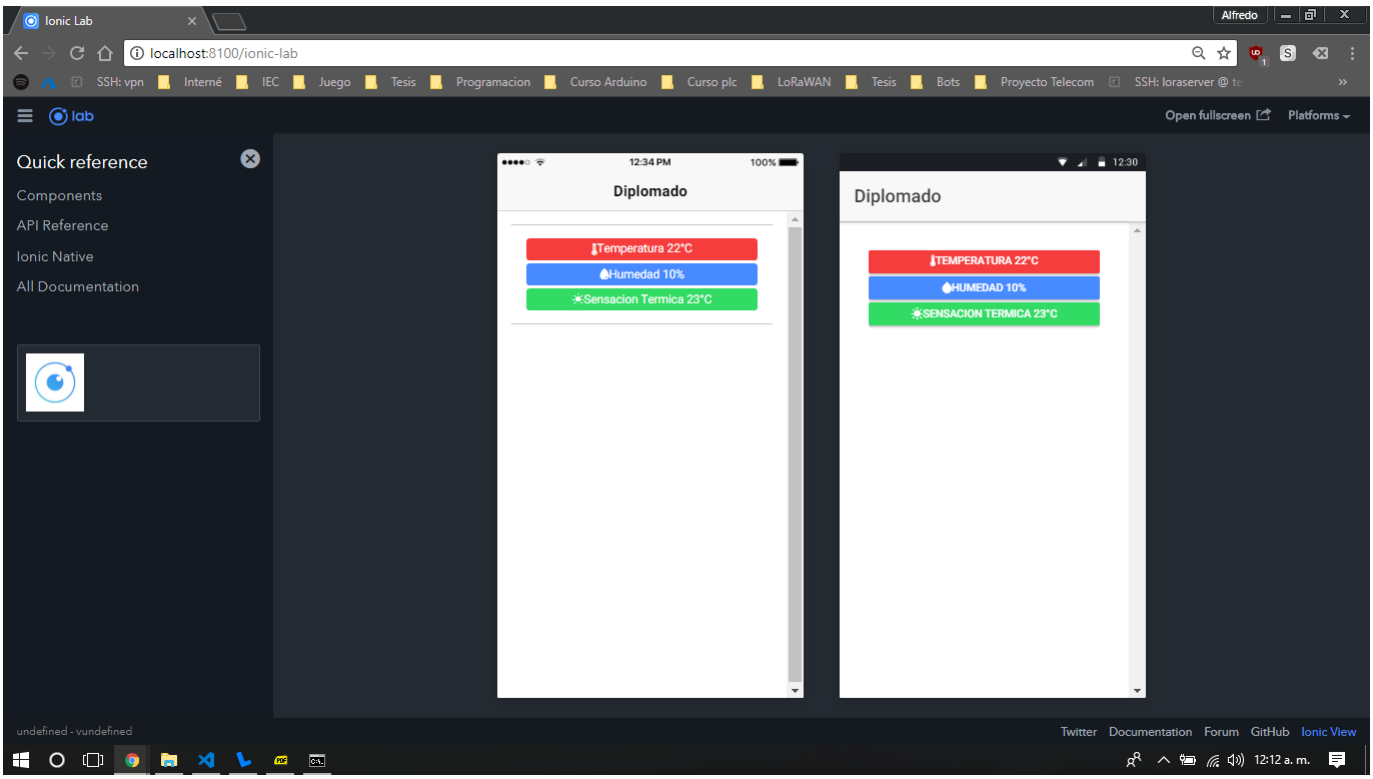

Figura 5: Vista previa de la aplicación

Figura 6: Vista final de la aplicación