

Máster Universitario en Ingeniería Web Front-End para Móviles

Desarrollo de aplicación móvil Android para gestionar un Smart Home: Listo para Madrid

Ramón Morcillo Cascales

Índice

1. Contexto. ¿Para qué sirve?	1
2. Proceso	2
2.1 Elegir la API	2
2.2 Añadir Retro Api	2
2.3 Añadir el Oauth the Firebase	3
2.4 Media y alerta de temperatura	6
2.5 Comunicarse con la lámpara WIFI	8
2.5 Persistir los datos obtenidos (Firebase realtime database)	11
2.6 Actualizar Clima	13
3. Conclusión	15

1. Contexto. ¿Para qué sirve?

Listo para Madrid es una aplicación basada en la sencillez y cuyo objetivo es prepararte a punto para afrontar el día en lo referente a la temperatura y precipitaciones del exterior. El objetivo es que el usuario una vez entre en la aplicación pueda visualizar la temperatura y precipitaciones en las próximas 24h horas por lo que podrá prepararse, abrigarse y coger un paraguas si lo considera necesario.

La aplicación además te permite conectarte a una lámpara inteligente IoT la cual te avisará mediante el color de la misma del estado del clima para el día de hoy y si deberías abrigarte o coger un paraguas.

2. Proceso

Para realizar la aplicación he decidido centrarme en una idea simple pero muy usable como es cada mañana o noche comprobar que ropa debería llevar al día siguiente o si debería llevar paraguas. Lo primero ha sido crear un proyecto en **Android Studio** y crear los primeros layouts.

2.1 Elegir la API

La API elegida ha sido la de <https://openweathermap.org/> la cual te permite saber el tiempo y clima de cualquier lugar del mundo. Tiene muchos endpoints para trabajar con ella pero yo me he centrado en solo uno, el del pronóstico de los próximos 5 días cuya documentación se encuentra en el siguiente enlace: <https://openweathermap.org/forecast5>.

La razón por la que me he centrado en ese endpoint es porque me ofrecía la información necesaria para mi proyecto. Del pronóstico de 5 días recojo las primeras 24 horas para mi aplicación.

2.2 Añadir Retro Api

Una vez decidida la API con la que voy a trabajar necesito un método para conectarme con ella. En este caso he elegido Retro Api para cumplir esta funcionalidad ya que es más sencilla que otras librerías. El objetivo final de esta fase era conseguir mostrar por pantalla la información recibida de la API, en este caso la temperatura y nubes de las siguientes horas.

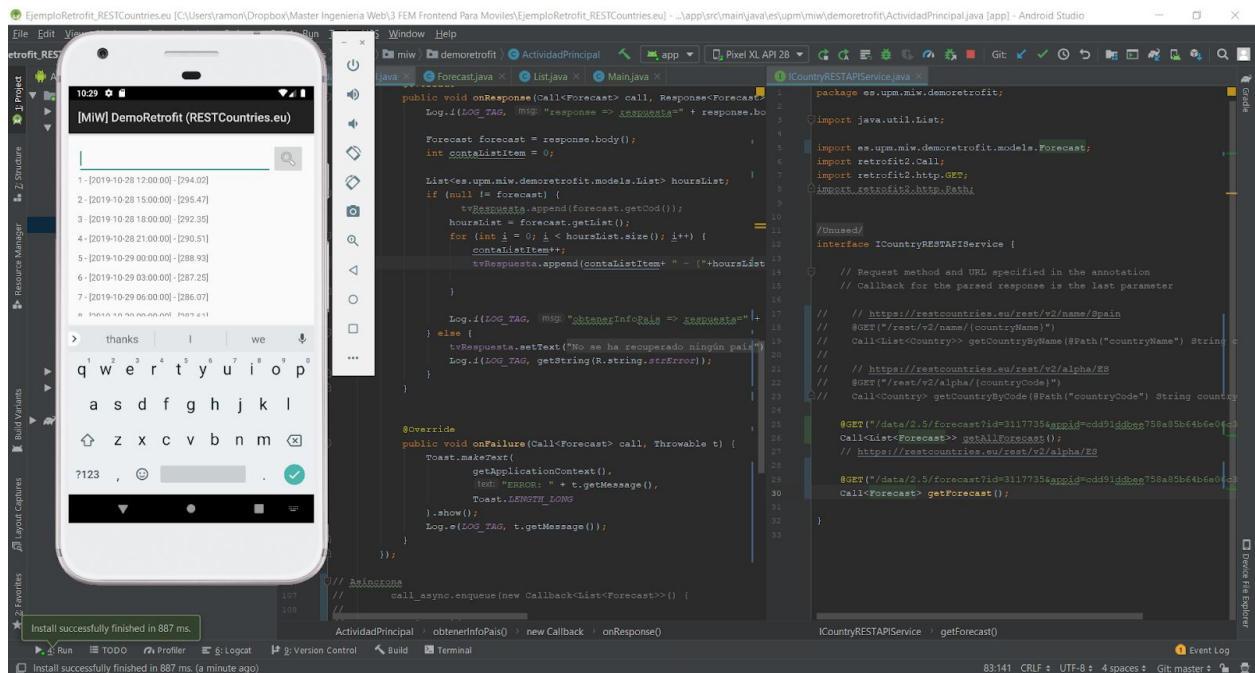


Figura 1. Captura de escritorio de la aplicación recogiendo y mostrando los resultados de la API.

2.3 Añadir el Oauth the Firebase

El siguiente paso fue añadir un sistema de autenticación previo a la interfaz principal de la aplicación usando Firebase. Una vez creado el Proyecto en Firebase habilité el Sign-in mediante Google y Email.

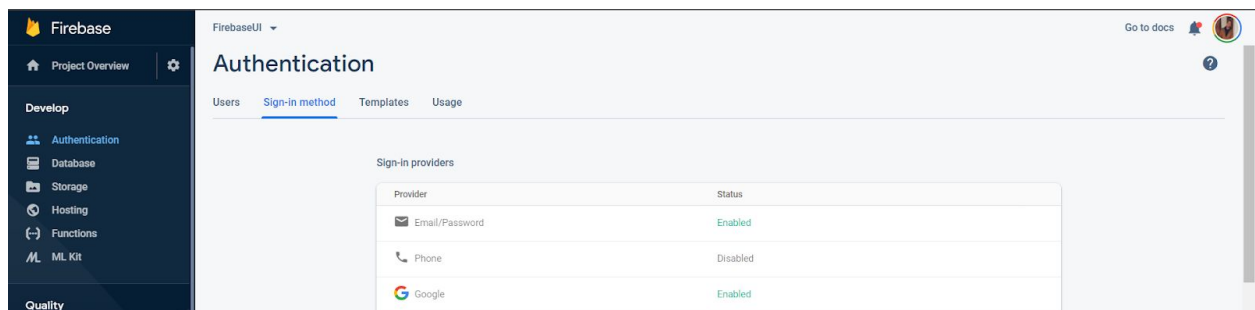


Figura 2. Autenticación por Google y por correo activados en Firebase.

De este modo luego en la aplicación nos aparecen ambos métodos para hacer Login.

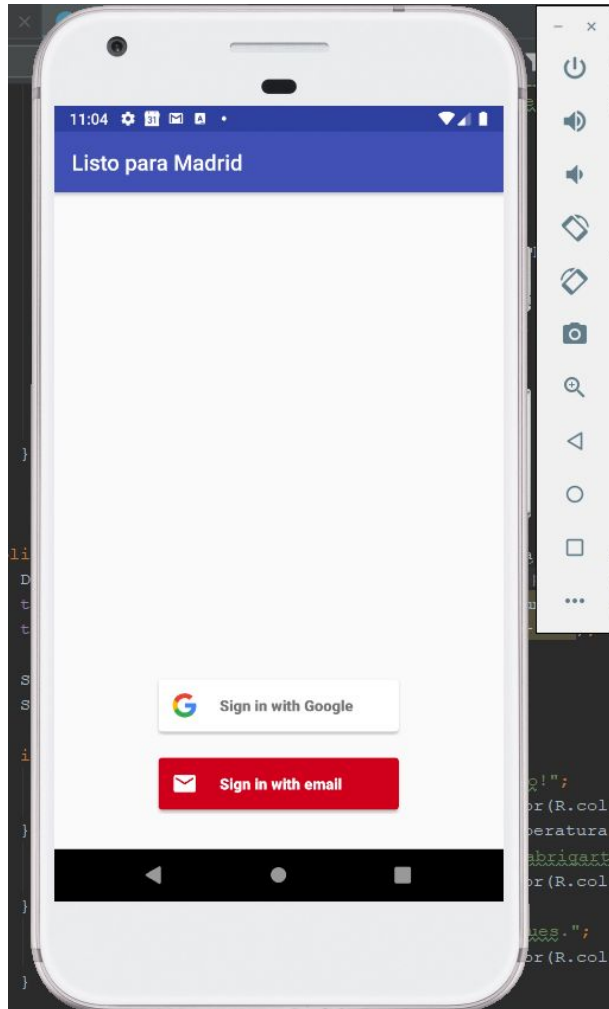


Figura 3. Pantalla de autenticación por Google o por correo dentro de la aplicación.

Una vez hecho login el usuario que ha accedido aparece en Firebase.

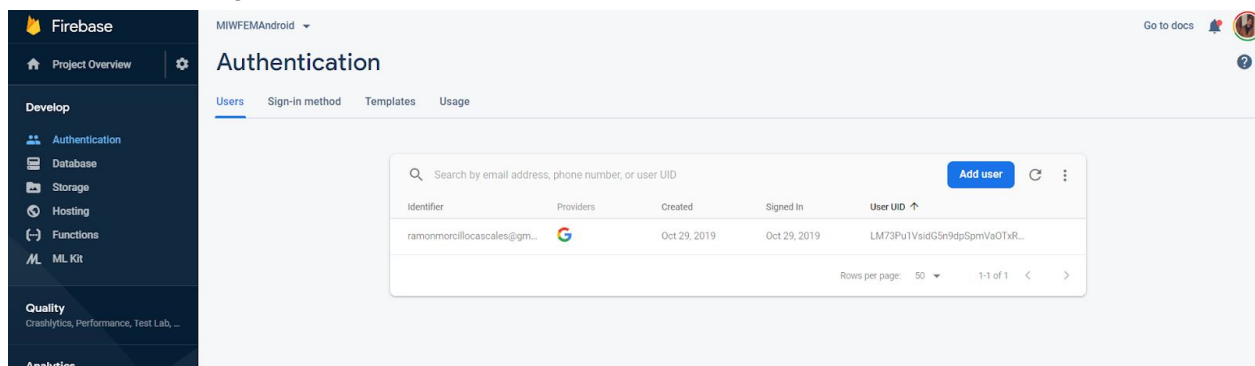


Figura 4. Usuario registrado en Firebase.

Y ya en la aplicación al finalizar la autenticación correctamente aparece la interfaz principal de la aplicación con un botón para hacer logout y salir de la misma.

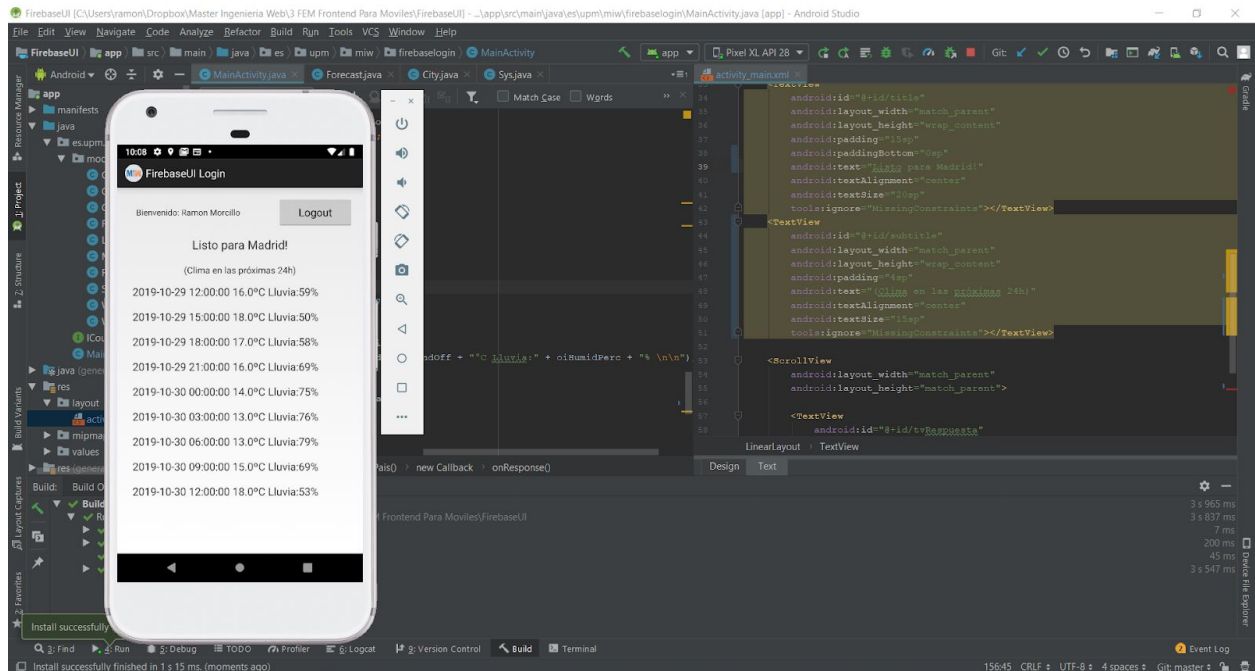


Figura 5. Captura de pantalla de la aplicación con el método de Logout implementado.

Para usar las funcionalidades de Firebase he tenido que añadir a mi proyecto un fichero **google-services.json** en el directorio **app** de la aplicación:

```
{
  "project_info": {
    "project_number": "668692236785",
    "firebase_url": "https://miwfemandroid.firebaseio.com",
    "project_id": "miwfemandroid",
    "storage_bucket": "miwfemandroid.appspot.com"
  },
  "client": [
    {
      "client_info": {
        "mobilesdk_app_id":
"1:668692236785:android:4fdabdb30ed5f94763dc50",
        "android_client_info": {
          "package_name": "es.upm.miw.firbaselogin"
        }
      },
      "oauth_client": [
        {
          "client_id":
"668692236785-v80ccrrumpdk52g41fum24ek4165epum.apps.googleusercontent.com",
```

```

        "client_type": 1,
        "android_info": {
            "package_name": "es.upm.miw.firebaselogin",
            "certificate_hash": "7ffabe78566a802e9276ad19e7e7262643a976e4"
        }
    },
    {
        "client_id":
"668692236785-2qc6ab6fjt4p8haigcqfgd3f1tnvjm0a.apps.googleusercontent.com",
        "client_type": 3
    }
],
"api_key": [
    {
        "current_key": "AIzaSyDCHAXd1yxW9bK6wY7hr7I2hgk6YLFvA0E"
    }
],
"services": {
    "appinvite_service": {
        "other_platform_oauth_client": [
            {
                "client_id":
"668692236785-2qc6ab6fjt4p8haigcqfgd3f1tnvjm0a.apps.googleusercontent.com",
                "client_type": 3
            }
        ]
    }
}
],
"configuration_version": "1"
}

```

2.4 Media y alerta de temperatura

Con esta estructura y recursos he mejorado la interfaz para mostrar la temperatura y nubes media así como unos mensajes de aviso para que el usuario sepa si debe abrigarse o coger el paraguas. El mensaje y color del mismo varía según la temperatura y proporción de nubes.

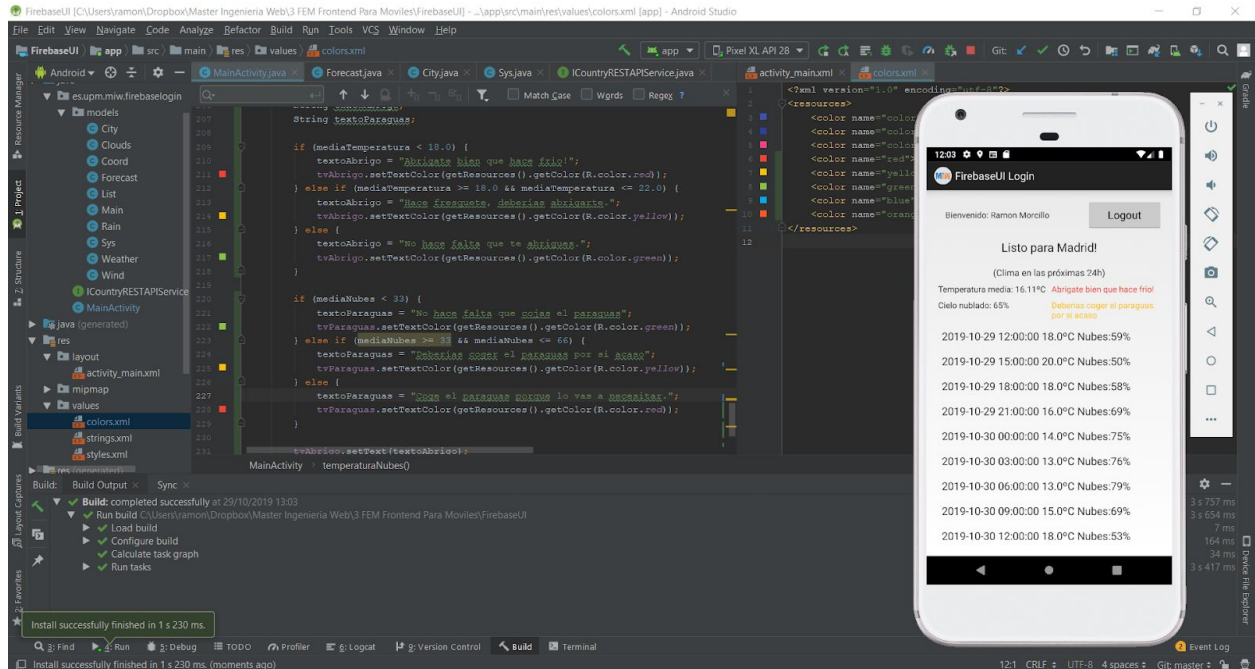


Figura 6. Captura de pantalla de la aplicación con la media de temperatura y nubes y alertas implementados.

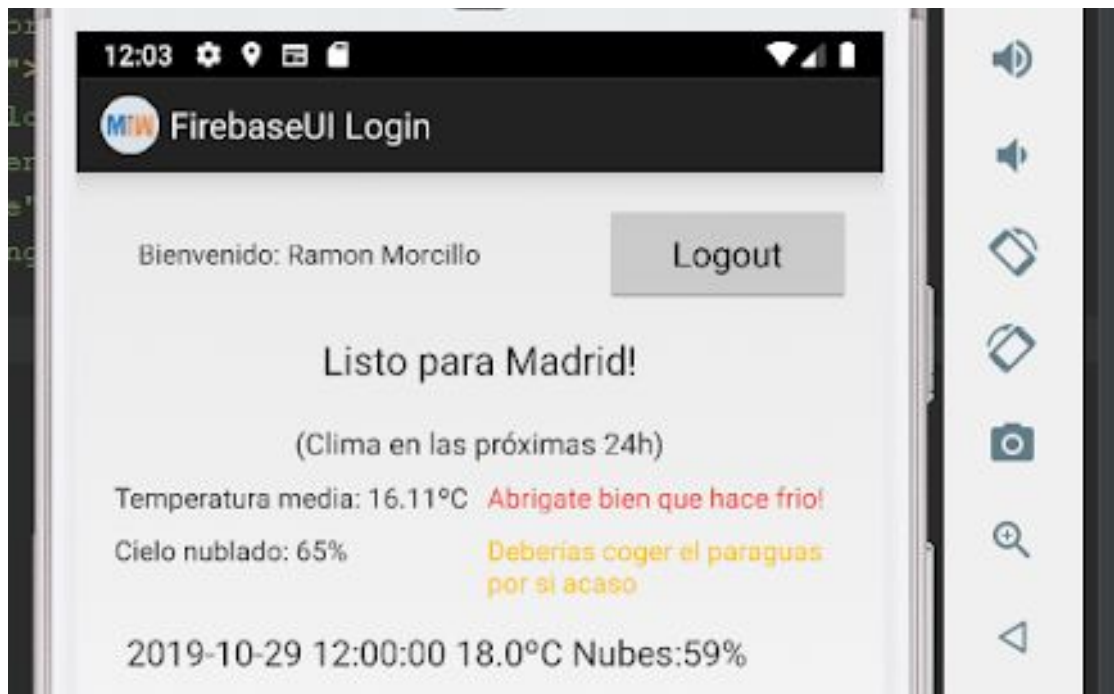


Figura 7. Captura más próxima de la aplicación con la media de temperatura y nubes y alertas implementados.

2.5 Comunicarse con la lámpara WIFI

El siguiente punto a cubrir ha sido la funcionalidad de la lámpara IoT. Para la conexión con la misma he copiado los recursos del ejemplo y adaptado los métodos a mi proyecto de tal modo que cuando el usuario pulsa un botón la lámpara emite un color en función de la temperatura; Si hace frío la lámpara se pondrá azul mientras que si hace calor adquirirá una tonalidad cálida roja.



Figura 8. Botón de activar la luz según la temperatura implementado

A continuación se muestra el código implementado en la clase principal. Una vez se hace click en el botón se ejecuta la función `setCubeLight()`

```
public void setCubeLight() {
    // First encender cubo
    encenderCubo();
    // Si hace frio Cubo azul
    if (mediaTemperatura < 20) {
        sendColor(new FeedbackColor(0, 0, 255));
    } else {
        // Si hace calor cubo rojo
        sendColor(new FeedbackColor(255, 0, 0));
    }
}

public void sendColor(FeedbackColor color) {
    FCColor fcc = new FCColor(sIp, "" + color.getR(), ""
        + color.getG(), "" + color.getB());
    new FeedbackCubeManager().execute(fcc);
}

public void encenderCubo() {
    FCOn f = new FCOn(sIp);
```



```
        new FeedbackCubeManager().execute(f);  
    }  
    public void apagarCubo() {  
        FCOff f = new FCOff(sIp);  
        new FeedbackCubeManager().execute(f);  
    }
```

Ahora por ejemplo que las temperaturas son frías la lámpara adquiere el color Azul:



Figura 9. Lámpara IoT con el color azul tras presionar el botón de Activar Luz según temperatura con una temperatura menor de 20 grados.

También he añadido unos emoticonos a los mensajes para mejorar la experiencia del usuario

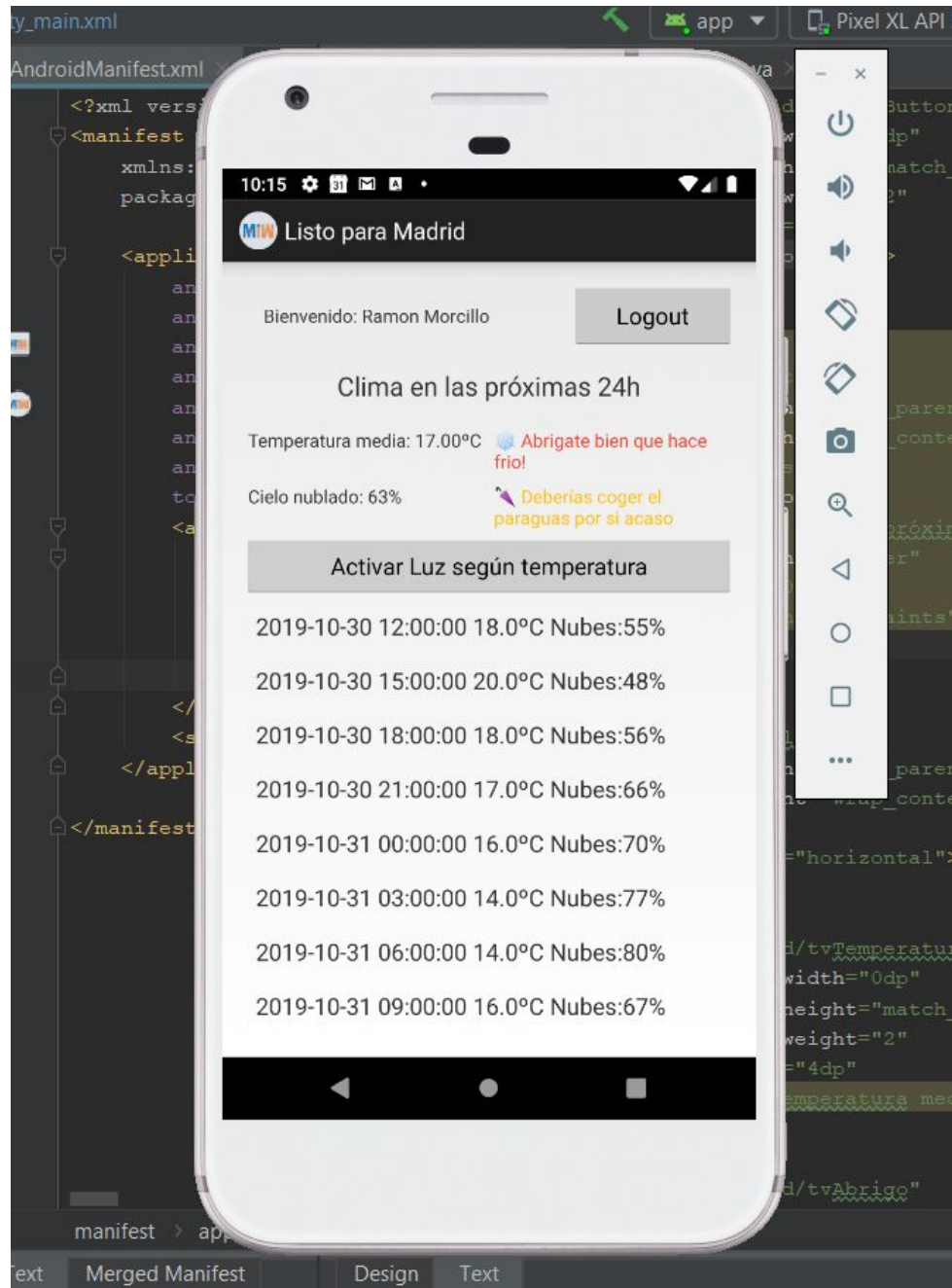


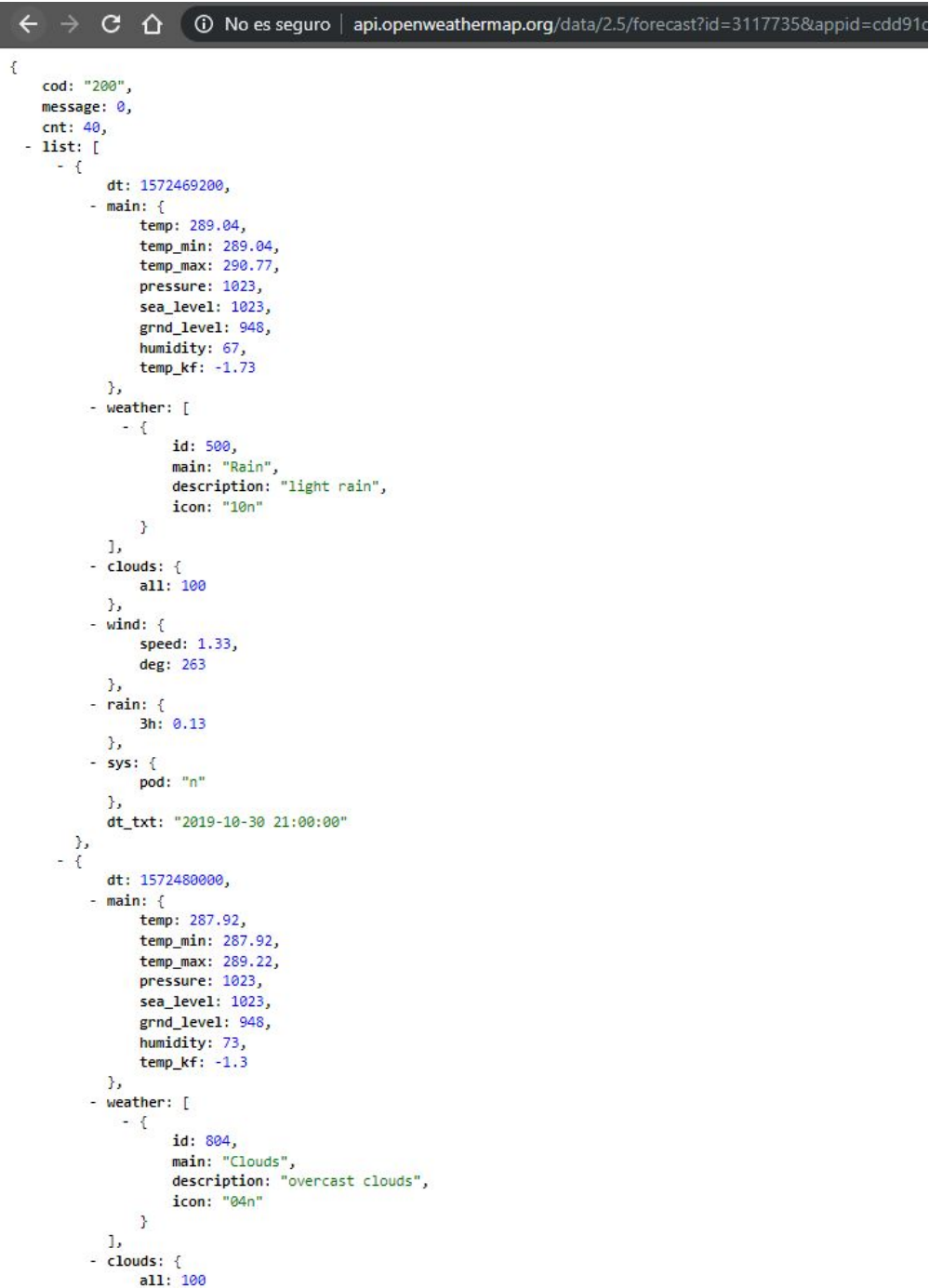
Figura 10. Captura de pantalla de la aplicación con emoticonos en las alertas.

2.5 Persistir los datos obtenidos (Firebase realtime database)

La última parte del proyecto consistía en persistir los datos obtenidos de las sucesivas consultas al API externo. Para ello he usado Firebase Realtime Database:

- Primero he añadido el sistema de realtime database al proyecto creado anteriormente con las reglas para test

- Luego he añadido las dependencias en el proyecto de android y hago que se guarde el objeto forecast una vez se realiza la petición a la API del tiempo.



```
{
  cod: "200",
  message: 0,
  cnt: 40,
  - list: [
    - {
      dt: 1572469200,
      - main: {
        temp: 289.04,
        temp_min: 289.04,
        temp_max: 290.77,
        pressure: 1023,
        sea_level: 1023,
        grnd_level: 948,
        humidity: 67,
        temp_kf: -1.73
      },
      - weather: [
        - {
          id: 500,
          main: "Rain",
          description: "light rain",
          icon: "10n"
        }
      ],
      - clouds: {
        all: 100
      },
      - wind: {
        speed: 1.33,
        deg: 263
      },
      - rain: {
        3h: 0.13
      },
      - sys: {
        pod: "n"
      },
      dt_txt: "2019-10-30 21:00:00"
    },
    - {
      dt: 1572480000,
      - main: {
        temp: 287.92,
        temp_min: 287.92,
        temp_max: 289.22,
        pressure: 1023,
        sea_level: 1023,
        grnd_level: 948,
        humidity: 73,
        temp_kf: -1.3
      },
      - weather: [
        - {
          id: 804,
          main: "Clouds",
          description: "overcast clouds",
          icon: "04n"
        }
      ],
      - clouds: {
        all: 100
      }
    }
  ]
}
```

Figura 11. Json Forecast devuelto en la petición a la API

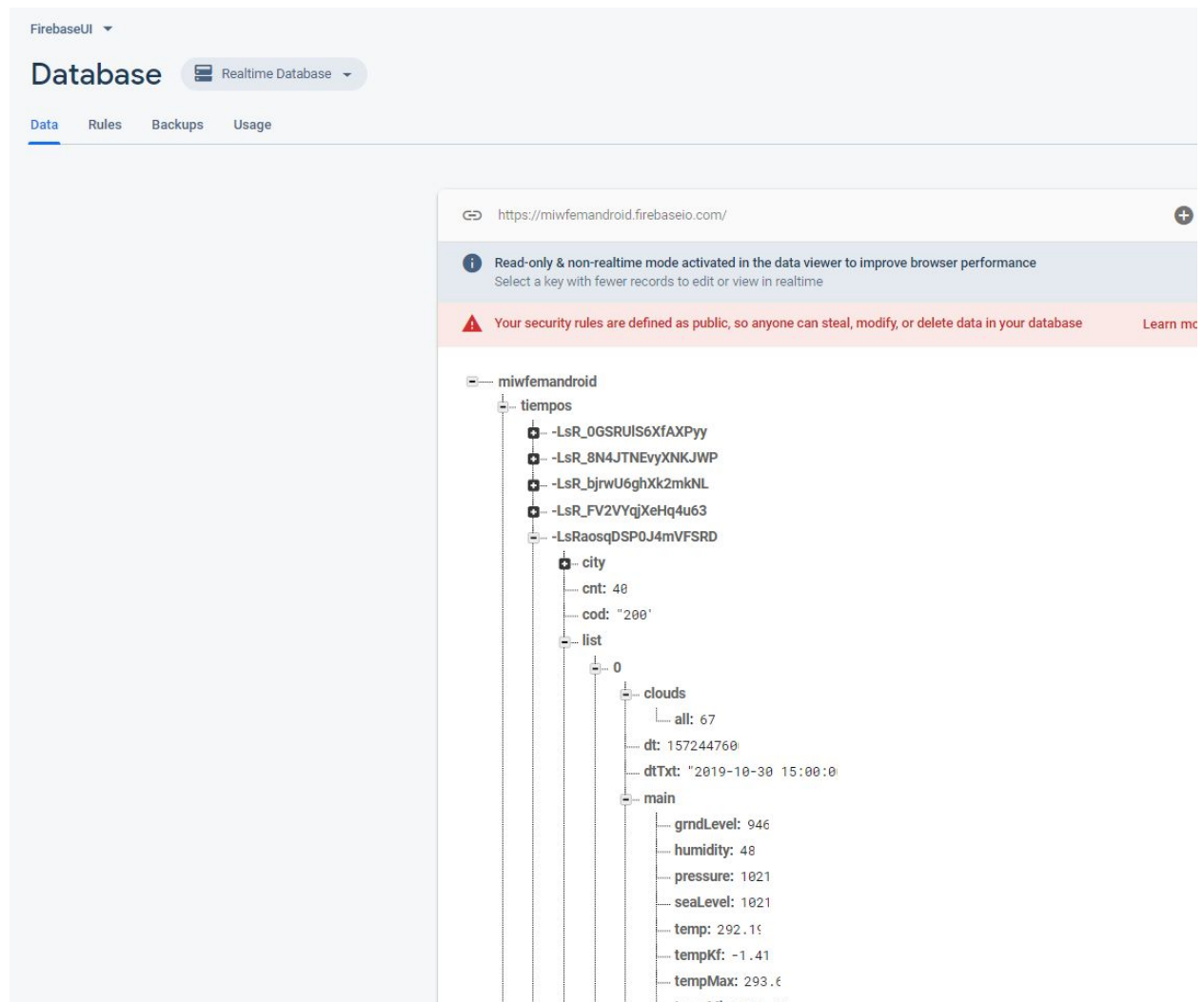


Figura 12. Objetos Forecast guardados en la Realtime Database de Firebase

2.6 Actualizar Clima

Para acabar he añadido un botón para actualizar el clima que vuelve a recoger los datos de la API y recalcular la temperatura y probabilidad de lluvia de nuevo. De este modo el usuario no tiene que abrir y cerrar la aplicación cada vez que quiera actualizar el clima.

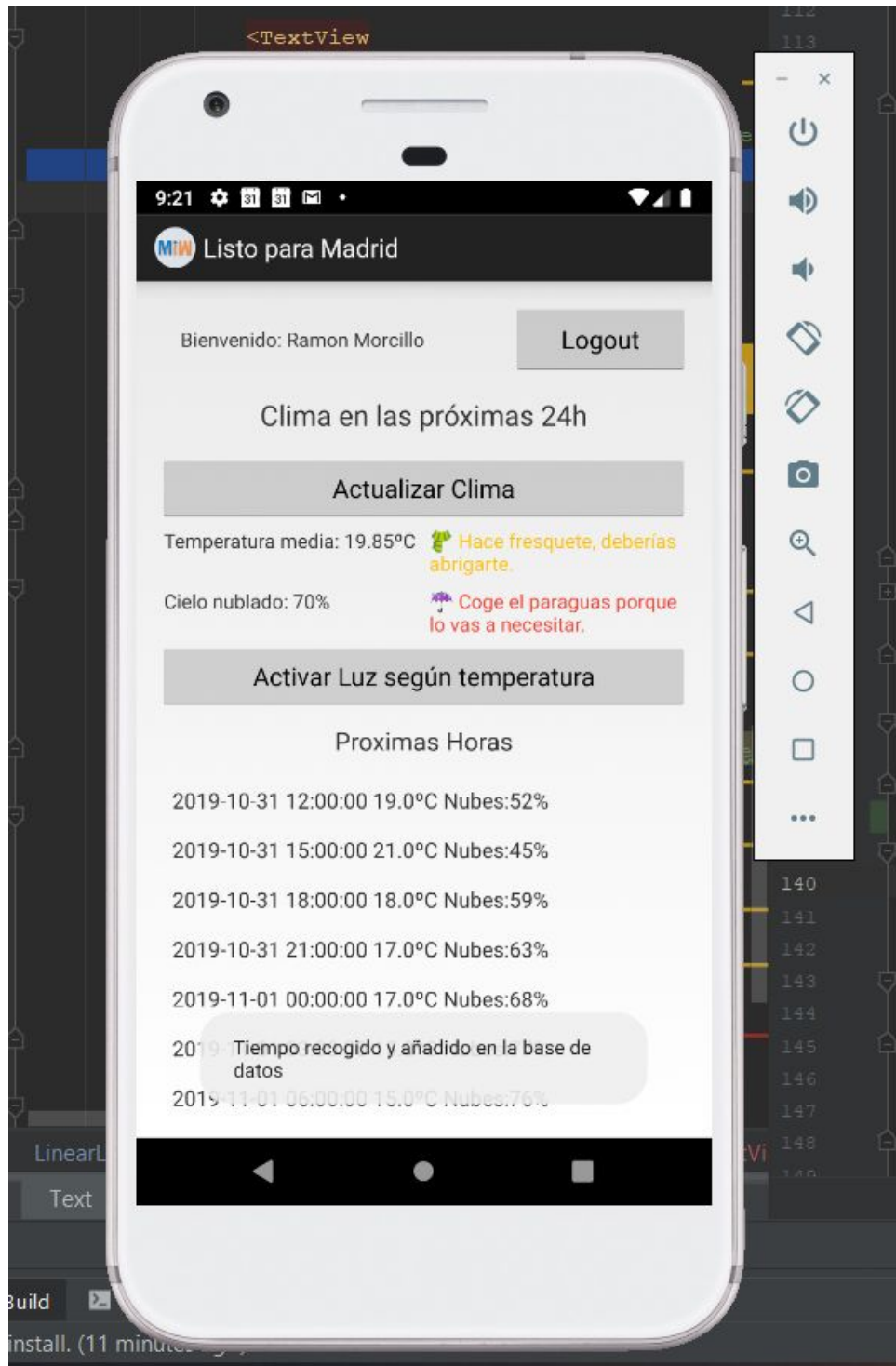


Figura 13. Botón y funcionalidad de actualizar clima implementados en la aplicación.

3. Conclusión

Aunque al principio, al ver en qué consistía, creí que sería más complicada al final y avanzando poco a poco he conseguido realizarla entera. La práctica me ha enseñado a ir más allá con Android implementando conexiones con APIs externas y usando sistemas de Autenticación y Persistencia en la nube.

Por último la inclusión de la lámpara IoT en la práctica permite experimentar con este tipo de tecnologías que cada vez más se encuentran en nuestro día a día.