GitHub Username: marcotf-git

Learning App

Description

The app is for helping teachers and students in having a short class with video, text and useful links about the content. The teachers can use the Android device, intended to be a tablet, for making a mini video and a small text with some useful links for further reading. The students can load the content in their devices, phones or tablets, for viewing anytime.

The user can view and create the content. The app will have a local database storage (for viewing offline) and external database for uploading or downloading.

Intended User

This is an app for anyone interested in teaching short classes. The app can also be used in other areas, as its structure is of a general purpose knowledge sharing app.

Features

The app will have the main features:

- The app will help the user (teacher) in creating the class, by making the movie with the camera of the device, by making the text with the virtual keyboard, and storing the content in an accessible external database.
- Storing data in the device is useful in situations when there is not wireless connection or it can be interrupted frequently, like when travelling.
- The app will retrieve the data stored in the external database to the local database, when the user or student want to attend new classes.
- Having the external database for uploading is useful for sharing the content with the students. No matter where they are, they can download and have the content for helping with their studies.

User Interface Mocks

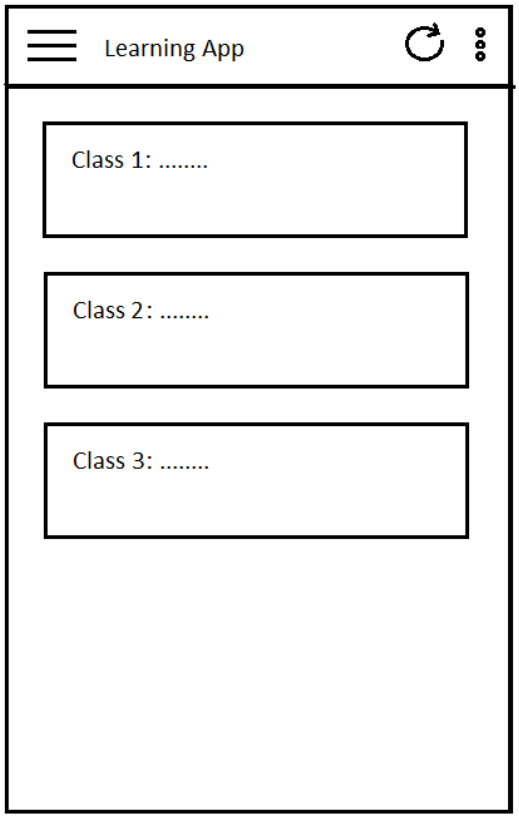

Fig. 1 - Screen 1

The main screen will show an app bar and some cards for the classes that are stored in the local database. The navigation menu icon will handle the login account. The settings icon will handle two main uses for the app: “create”, when the user can create the content, and “view”, when the user only views the classes. There will be a refresh icon, for downloading the latest content from the external database.

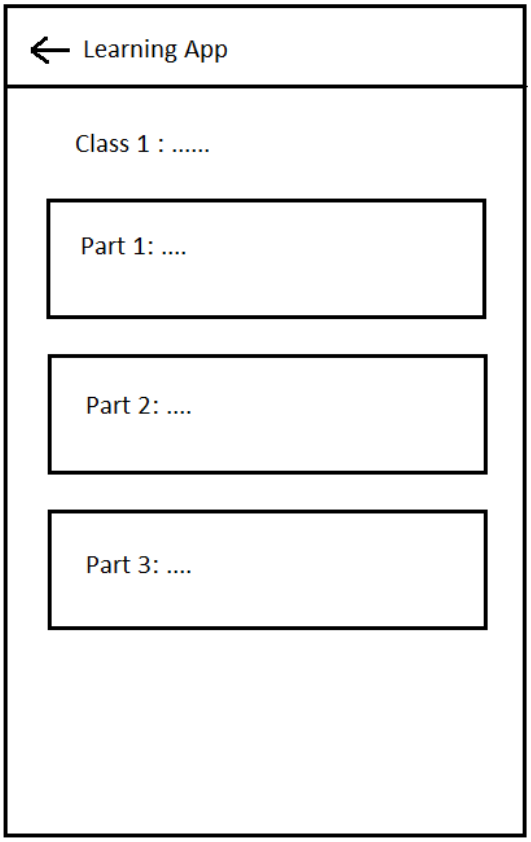

Fig. 2 - Screen 2

When the user clicks into one of the cards on the main screen (screen 1), another screen (screen 2) will open showing some cards with a short explanation of the content of each part of the class.

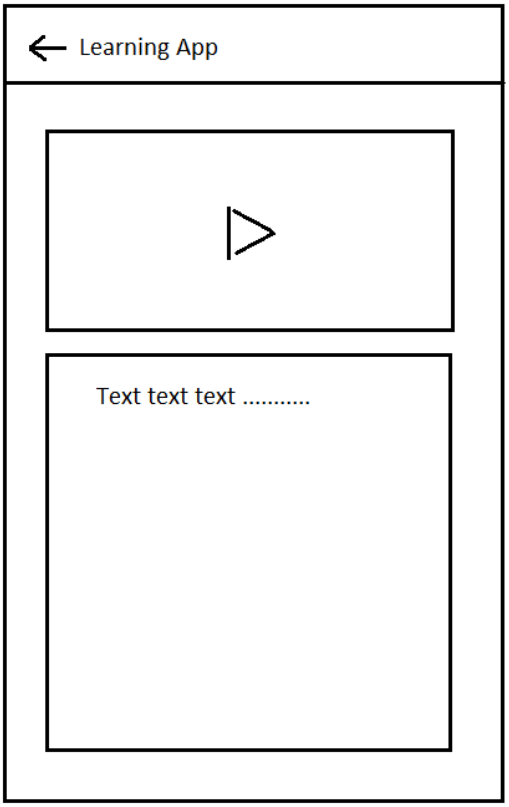

Fig. 3 - Screen 3

If the user clicks again into one of these cards, it will open another screen (screen 3: on the right) with the video or image, and the text and links.

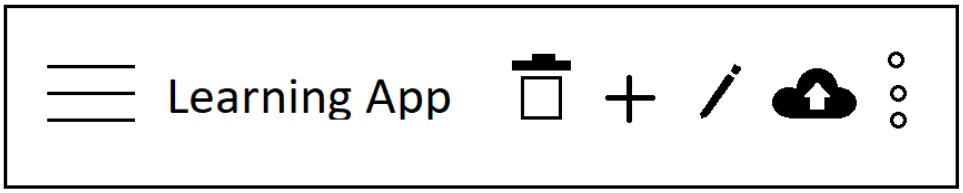

Fig. 4 - App bar

The app bar will be contextual. When the option to “create” is selected, the icons for adding, editing, deleting and uploading content will appear.

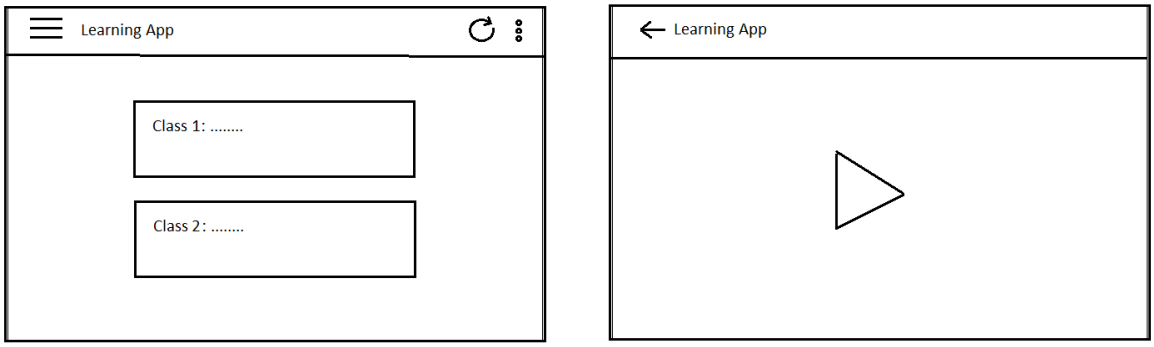

Fig. 5 - Phone in portrait mode

For a phone in portrait mode, it will use one column layout with a wide margin, in case of text (screens 1 and 2), and it will show only video on screen 3.

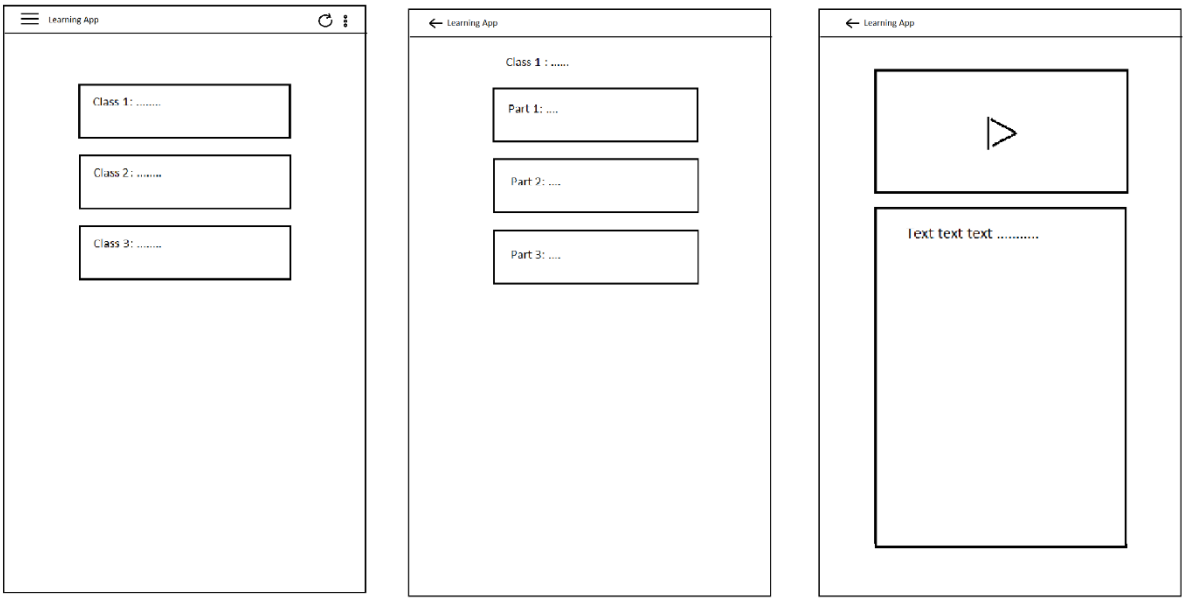

Fig. 6 - Tablet in portrait

For a tablet in portrait mode, it will have greater text sizes and wide margins and video view sizes.

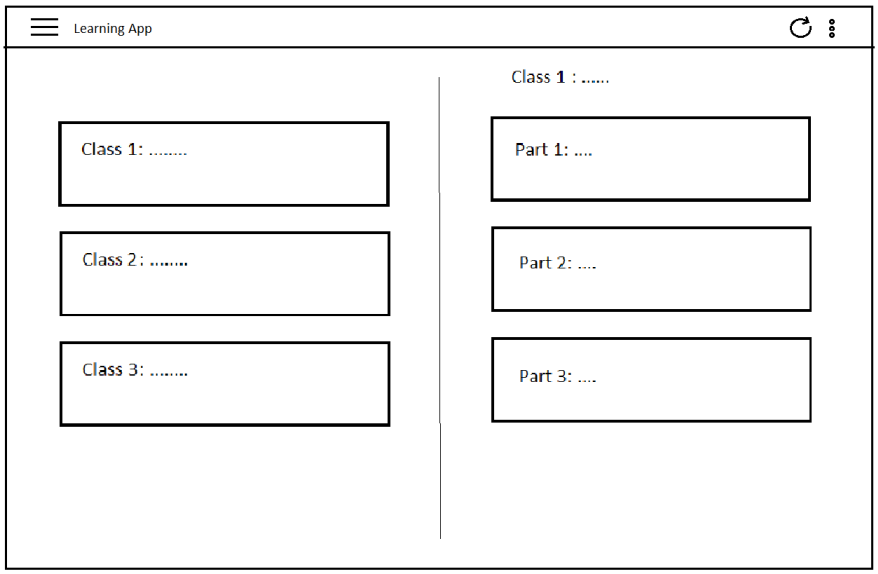

Fig. 7 - Tablet in landscape

For tablet on landscape mode, it will use two column layout, showing the class parts on the right, of the selected class on the left.

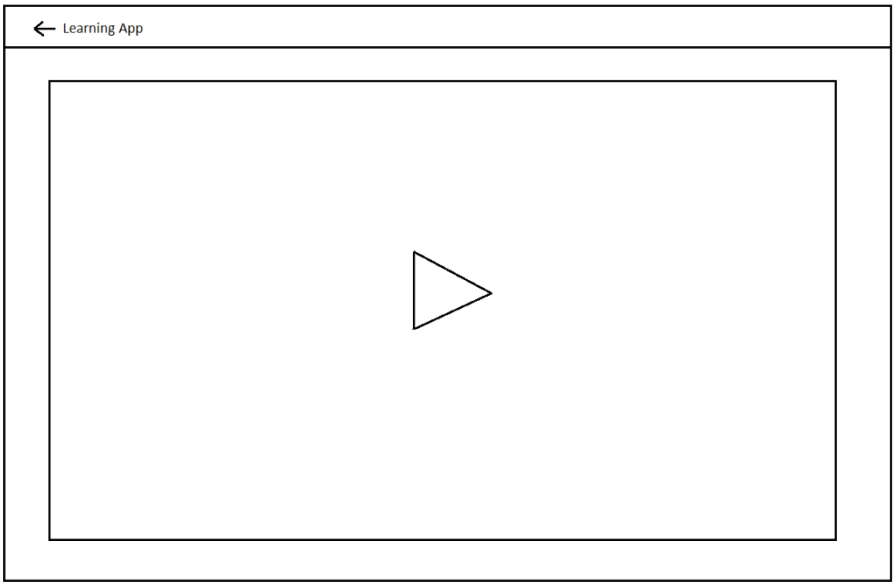

Fig. 8 - Tablet in landscape: video screen

For tablet on landscape mode, the video will fill the entire screen

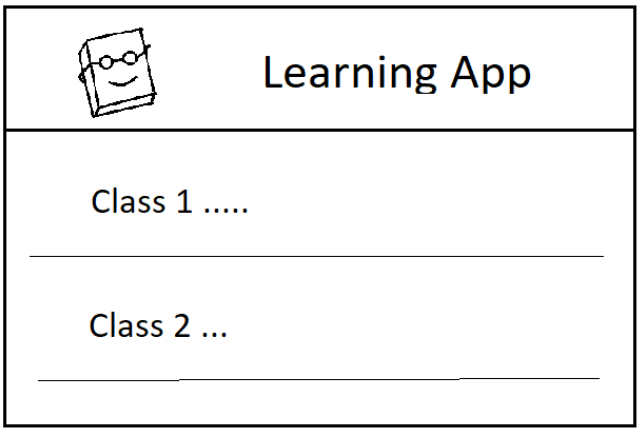

Fig. 9 – App widget

The app widget will show the class titles. Clicking on them will open that class.

Key Considerations

Programming Language

The app will be written solely in the Java Programming Language (https://docs.oracle.com/javase/8/docs/api/).

Libraries, Gradle and Android Studio

The app will be developed on Android Studio 3.1.3 IDE, and will use the Gradle for the build process, with the Android Plugin for Gradle version 3.1.3 and Gradle 4.4.

The app will also use, in a first approach, the following libraries:

- **Android v4 Support Libraries** for application components, user interface features, accessibility, data handling, network connectivity, and programming utilities (1)
- **Android v7 appcompat library** for Action bar user interface (1)
- **Android v7 recyclerview library** for the RecyclerView class (1)
- **Android v7 cardview library** for the CardView widget (1)
- **Design Support Library** for material design components (snackbar) (1)
- **Butter Kinfe** for UI data binding (10)
- **Gson** for converting Java Objects into their JSON representation (8)
- **Firebase Authentication** for securing the data (4)
- **FirebaseUI Auth** for the log in UI (5)

- **Firebase Realtime Database** for storing the text data on a database in JSON format (4)
- **Firebase Storage** for storing the images or videos on Firebase and getting the url (4)
- **Picasso** to handle the loading and caching of images based on the url of the image (6)
- **ExoPlayer** for fetching playing the videos based on the url of the video (7)
- **Android testing Support Library** for Espresso tests (9)

ref.	Developer site reference
(1)	https://developer.android.com/topic/libraries/support-library/packages
(2)	https://developer.android.com/studio/releases/gradle-plugin
(3)	https://developers.google.com/android/guides/google-services-plugin
(4)	https://firebase.google.com/docs/android/setup
(5)	https://firebase.google.com/docs/auth/android/firebaseui
(6)	http://square.github.io/picasso/
(7)	https://github.com/google/ExoPlayer
(8)	https://github.com/google/gson
(9)	https://developer.android.com/training/testing/espresso/setup
(10)	https://github.com/JakeWharton/butterknife

The setup details and library versions are in the tables below:

Library Name	Android app dependency		ref.
v4 compat library	com.android.support:support-compat:	27.1.1	(1)
v4 core-utils library	com.android.support:support-core-utils:	27.1.1	(1)
v4 core-ui library	com.android.support:support-core-ui:	27.1.1	(1)
v4 media-compat library	com.android.support:support-media-compat:	27.1.1	(1)
v4 fragment library	com.android.support:support-fragment:	27.1.1	(1)
v7 appcompat library	com.android.support:appcompat-v7:	27.1.1	(1)
v7 recyclerview library	com.android.support:recyclerview-v7:	27.1.1	(1)
v7 cardview library	com.android.support:cardview-v7:	27.1.1	(1)
Design Support Library	com.android.support:design:	27.1.1	(1)
Gson	com.google.code.gson:gson:	2.8.5	(8)
Firebase Analytics	com.google.firebase:firebase-core:	16.0.1	(4)
Firebase Realtime Database	com.google.firebase:firebase-database:	16.0.1	(4)
Firebase Authentication	com.google.firebase:firebase-auth:	16.0.1	(4)
Firebase Storage	com.google.firebase:firebase-storage:	16.0.1	(4)
FirebaseUI Auth	com.firebaseui:firebase-ui-auth:	4.0.1	(5)
Picasso	com.squareup.picasso:picasso:	2.71828	(6)
ExoPlayer	com.google.android.exoplayer:exoplayer:	2.8.1	(7)
Butter Knife	com.jakewharton:butterknife:8.8.1	8.8.1	(10)
Butter Knife - Compiler	com.jakewharton:butterknife-compiler:8.8.1	8.8.1	(10)
Android Testing Support Library - Espresso tests	com.android.support.test.espresso:espresso-core:	3.0.1	(9)
Android Testing Support Library - Junit 4 rules	com.android.support.test:rules:	1.0.1	(9)
Android Testing Support Library - Espresso Idling Resource	com.android.support.test.espresso:espresso-idling-resource:	3.0.1	(9)
Android Testing Support Library - Espresso Contrib	com.android.support.test.espresso:espresso-contrib:	3.0.1	(9)

Plugin Name	Gradle build script (itself) dependency		ref.
Android Plugin for Gradle	com.android.tools.build:gradle:	3.1.3	(2)
Google Services Gradle Plugin	com.google.gms:google-services:	4.0.1	(3)

Android defaultConfig dependency		ref.
testInstrumentationRunner	android.support.test.runner.AndroidJUnitRunner	(9)

Data persistence

The data structure will be in the form of JSON string, storing the classes titles, part titles and part detail text, and uri links of the videos or images. The videos or images will be stored as files.

There will be a local SQLite (https://www.sqlite.org/index.htm) database, to store the JSON string, and a Content Provider to query and write to it. The app will use a Loader class to (asynchronously) retrieve the data from the local database and update the views. It will use Picasso library to fetch the locally stored images and ExoPlayer for the videos.

There will be also an external database, on a remote server, for uploading and retrieving data. The app will use the Loader for inserting the data in the local database, after the data has been retrieved. The data will be fetched with the Firebase methods.

When the user creates new data, the data will be first inserted into the local database and folder. After that, it will be possible to upload to the remote server.

App operation modes

The app will have two main settings: the “view” mode and the “create” mode, which will be selected on the overflow menu.

On “view” mode, the app will have three main screens, each one showing the data in a more specific way (fig. 1: Class Titles; fig. 2: Class Parts; fig. 3: Part Detail, with text and video or image).

On “create” mode, the app will have the same screens, but the app bar (fig. 4) will show the options in the form of action items to create new content, edit or delete, according to the view shown. Clicking on the “create” action item, it will open a text box in the same screen for creating that content.

External services

The app will use the **Firebase** (https://firebase.google.com) for handling the external database and for the authentication and protecting the data.

Editing and deleting data

Only the user that has created the content will be able to edit or delete it.

Viewing data

All users will be able to view the data uploaded.

Widget

The app will have a widget that will show the class titles. The app also will have an Intent Service for providing data to the widget.

Resources management

The resources like images, strings, themes, colors, layouts, will be stored into the ‘res’ folder, in a subfolder according with the title of the resource, and some qualifier to handle the screen position and size (device type). The colors, strings and styles will be stored into the ‘values’ folder. With the resource qualifiers, when the device switch to landscape, the specific layout will be loaded. It also will adapt according to the device size, when the smallest width was above 600dp.

Accessibility

The app will apply Material Design guidelines. The App Theme will extend the AppCompat. The interfaces will be planned as it was material surfaces, setting the elevations and spaces accordingly the guidelines. The colors will be chosen to match great accessibility, and will be configured into the app style (styles.xml). It will use high quality images (full bleed dimensions) and the Android font Roboto for the text (great visibility). The line height and column width will assure good readability.

Error handling

If the app encounters an error, like when there is not internet or no data, the app will inform the user with a *snackbar*. While the app is waiting the load, it will also show a progress bar, so it will try to inform the user about this edge cases.

Required Tasks

Task 1: Project Setup

Setup the *build.gradle* for the project (**Gradle, Google Services, etc**.)**.**

Setup the *build.gradle* for the app and add the dependencies (**Android v4 Support Libraries**, **Android v7, Picasso, ExoPlayer, etc.).**

Setup the **Firebase** account and add the libraries to the app:

- Create a **Firebase** project on the **Firebase** Console (https://console.firebase.google.com)
- In the options of the project, select “Add **Firebase** to your **Android** app” and follow the instructions (register the app; download the JSON configuration file in the app directory; add the SDK dependencies to the *build.gradle*)
- Configure **Firebase Database** and **Auth** dependencies (https://github.com/firebase/FirebaseUI-Android)

Task 2: Implement the simpler version

- Build UI for MainActivity and MainFragment, showing only one text box
- Build the account log in screen as part of the navigation drawer
- Build the setup with options “view” and “create”, as part of overflow menu

Task 3: Implement simple local database

- Implement a contract and a content provider, with only one field for testing

Task 4: Implement external database on Firebase

- Create Firebase account for the project
- Implement a simple one field database on Firebase
- Upload content to the database when user clicks the upload icon
- Download content when user clicks the refresh

Task 5: Test the simple version

- Simulate three users with their own data (create and view “classes” with the only one text phrase)
- See if the security rules are working (users can’t write on other user data)
- See what happens when the app is offline

Task 6: Implement the simple version with a video

- Build UI for MainActivity and MainFragment, for also playing video

Task 7: Extend simple local database

- Extend the contract and the content provider, with field for video url

Task 8: Extend external database

- Implement the Firebase Storage
- Upload video content to the Storage when user clicks the upload icon, and saves the (external) url in the Firebase database and in the local database
- Download content when user clicks the refresh, saving it in the assets folder as a file, and its local (internal) url in the local database

Task 9: Test the simple version with video

- Simulate three users with their own data (create and view “classes” with the only one text phrase and one video)
- See if the security rules are working (users can’t write on other user data)
- Test also the “delete” command
- See what happens when the app is offline

Task 10: Extend the app: implement UI for each Activity and Fragment

Implement a complete version of the app, for mobile and tablet.

- Build UI for MainActivity and MainFragment (fig. 1, fig. 7)
- Build UI for DetailActivity and DetailFragment (fig. 2, fig. 7)
- Build UI for PartActivity (fig. 3, fig. 8)

Task 11: Extend the database

- Implement the contract and the content provider, with all the fields

Task 12: Extend the external database

- Implement all the fields in the external database

Task 13: Implement the widget

- Implement the widget layout file
- Implement the intent service

Task 14: Test the complete version

- Simulate three users with their own data (upload and download text and videos)
- Verify and rotate screens on mobile and tablet
- See if the security rules are working (users can't write on other users data)
- See what happens when the app is offline

Task 15: Verify Material Design

Verify if the app attend the Material Design specs.

- Try to make improvements to the layout, according to Material Design
- Verify the spaces between elements and the sizes of the views
- Check the colors
- Check the text font and visibility

Task 16: Test again

- Simulate three users with their own data (upload and download text and videos)
- Verify and rotate screens on mobile and tablet
- See if the security rules are working (users can't write on other user data)
- See what happens when the app is offline

Task 17: Make the instructions for installing

- Make the readme with instructions for installing and configuring the app
- Verify and correct any security issue with eventual app private keys
- Clean the project and upload