Capstone Project Design Document

In this document I am going to give a general description and overview over what the custom application is and what it does, I will also go through all the minor and major features the custom capstone app can offer, alongside this I am going to show how these features meet the capstone project rubric in order for it to pass, I will also show all the available screens that can be seen throughout the app.

Application general description and overview:

The application is simply a Meter readings tracking application designed specifically to track and monitor electricity, water and gas meter readings in Egypt using useful tables and graphs, it also calculates the costs of consumption on a daily, weekly or monthly bases based on a set of rules provided publicly by the corporates responsible of collecting electricity, water and gas money from the Egyptian citizens. The users can also create accounts and save their readings online using Firebase.

Application Features: Users are able to:

1. **Sign up for an account, Sign in using an existing account:**

- In case of Signing up: The users are taken through a set of screens to finish signing up for their account using Firebase.
- Users can also sign in using an existing account.

2. Create a tracking record and start tracking a certain meter:

- By clicking the floating button on the home screen, the users are able to start creating a tracking record.
- The users are then taken to a separate screen where they have to enter the following: **The name** of the tracking record for future reference, **the type** of the tracking record whether they are tracking electricity, gas or water meter readings (this is important so that the appropriate corresponding billing rules are applied later to calculate the cost), **the meter reading the last time the electricity, gas or water bills collector passed by to collect the bill** (this is important in order to calculate the appropriate multiplier needed to calculate the cost, more on the business logic later on the document), **the current meter reading**(the previous field minus this field gives us a kilowatt hour number that helps us determine the starting meter multiplier of the meter reading record in order to calculate the cost).
- After confirming the entered data, the users are then taken to the home screen to view an entry of the tracking record they just created.

3. View the recorded readings, and details of a certain tracking record:

- By clicking the meter itself in the main screen, the user is able to view the meter collections recorded for this meter, a meter collection starts in a certain start date and ends when the user records that the collector arrived to collect the meter bill, in that case a new meter collection is created with fresh data.

- The user is able to see the start and end dates, current meter slice (which determines the billing rules), the total cost up until this point (Cost of each and every meter reading plus the customer service for each slice), and the total consumption to this point of the meter collection.
- The user is able to click on a certain meter collection to view the meter readings recorded throughout the month, each meter reading entry shows the date the reading was taken, the actual reading, the consumption from the last meter reading and the cost of consumption.

4. Add a reading to the already existing record:

- The user is able to enter a reading to an already existing meter record, this can happen from the home screen by clicking the main floating button then selecting add meter reading, the user then chooses a certain meter record to add the reading to and then enters the meter reading, the meter reading can't be less than the last recorded meter reading for this meter, has to be numbers only and can’t exceed 5 digits.
- The user is able to enter a meter reading from the meter details screen itself.

5. Reset the current cost multiplier of the record by confirming that the bill collector showed up today:

- By clicking the button on the meter details screen and by entering the required fields, the user can end the current meter collection and start a new meter collection with fresh data.

6. Log out and be brought back to the Sign in and sign-up screens
7. Sync their record data with the cloud:

- Users are able to sync their data with the firebase real-time database by clicking the sync button in the home screen, the syncing rules are as follows:

1- Meters not found locally are fully downloaded alongside their meter collections and meter readings.

2- Meters not found remotely are fully uploaded alongside their meter collections and meter readings.

3- When some meters are found locally and remotely, the user is then asked to resolve the conflict by choosing to keep the local data which in turn uploads these meters, or accept the remote meters which in turn downloads these meters and overwrite the already existing local meters.

8. Upload a meter photo which can be viewed again when the user is connected to the internet

- By Clicking the meter photo, a new photo of the meter can be captured and uploaded that can be viewed later when the user is connected to the internet.

How does the application meet the rubric?

Build a navigable interface consisting of multiple screens of functionality and data.	Application includes at least three screens with distinct features using either the Android Navigation Controller or Explicit Intents. The Navigation Controller is used for Fragment-based navigation and intents are utilized for Activity-based navigation. An application bundle is built to store data passed between Fragments and Activities.

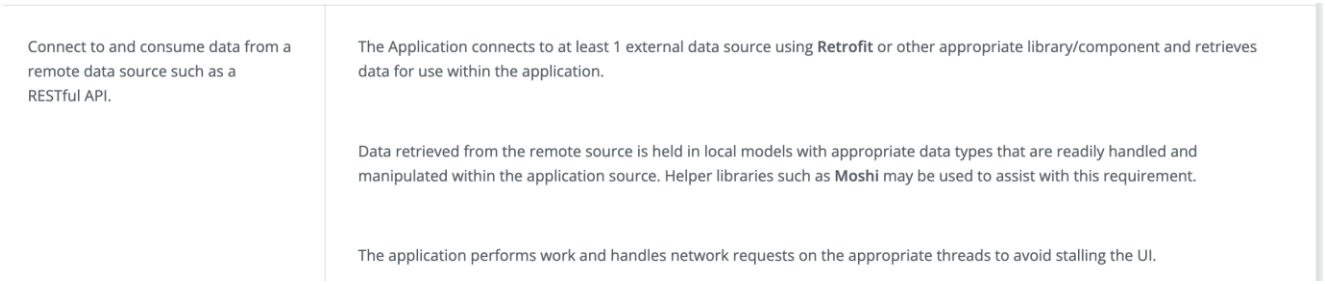
- The application contains two main activities with their fragments with two navigation graphs and controllers, data is passed between these screens and fragments successfully using a bundle and a shared view model in the second activity and its fragments.

Construct interfaces that adhere to Android standards and display appropriately on screens of different size and resolution.	Application UI effectively utilizes `ConstraintLayout` to arrange UI elements effectively and efficiently across application features, avoiding nesting layouts and maintaining a flat UI structure where possible. Data collections are displayed effectively, taking advantage of visual hierarchy and arrangement to display data in an easily consumable format. Resources are stored appropriately using the internal `res` directory to store data in appropriate locations including `string` * values, `drawables`, `colors`, `dimensions`, and more. Every element within `ConstraintLayout` should include the `id` field and at least 1 vertical constraint. Data collections should be loaded into the application using ViewHolder pattern and appropriate View, such as `RecyclerView`.

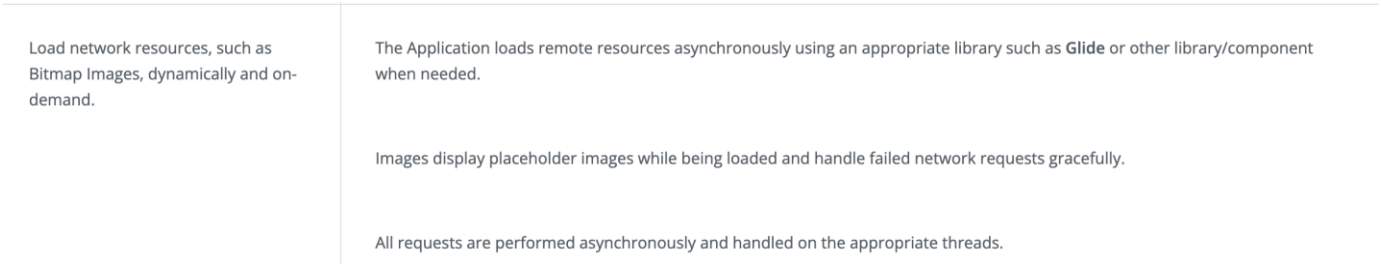
- The app uses the constraint layout in almost every layout xml file, and has multiple recycler views all follow the view holder design pattern, string, drawable, colors and dimensions are all stored in their dedicated files.
-

Animate UI components to better utilize screen real estate and create engaging content.	Application contains at least 1 feature utilizing *MotionLayout* to adapt UI elements to a given function. This could include animating control elements onto and off screen, displaying and hiding a form, or animation of complex UI transitions. *MotionLayout* behaviors are defined in a *MotionScene* using one or more *Transition* nodes and *ConstraintSet* blocks. *Constraints* are defined within the scenes and house all layout params for the animation.

- Motion layout is used in the second screen to animate the meter image with the meter collections and readings with its dedicated motion scene.



- Data is consumed from the firebase database in case of data syncing, where the user can download data they uploaded from a different device and upload new ones, the application stores this data in the local database as a local cache that can been be viewed while being offline, also all of that happens on different threads using Koltin coroutines.



- The application uses firebase storage to store meter photos and they are loaded dynamically and on demand when the user views the meter detail and has previously uploaded a meter photo, it used Glide to download the image and place it into the image view, a placeholder is there and all requests are done on a different thread.

Store data locally on the device for use between application sessions and/or offline use.	The application utilizes storage mechanisms that best fit the data stored to store data locally on the device. Example: `SharedPreferences` for user settings or an internal database for data persistence for application data. Libraries such as **Room** may be utilized to achieve this functionality. Data stored is accessible across user sessions. Data storage operations are performed on the appropriate threads as to not stall the UI thread. Data is structured with appropriate data types and scope as required by application functionality.

- The application stores data and retrieves it from a Room local database, data stored is accessible across user sessions, operations are performed on appropriate threads, data is structured with appropriate data types and scope as required by application functionality.

Architect application functionality using *MVVM*.	Application separates responsibilities amongst classes and structures using the MVVM Pattern: • *Fragments*/*Activities* control the *Views* • Models houses the data structures, • *ViewModel* controls business logic. Application adheres to architecture best practices, such as the observer pattern, to prevent leaking components, such as Activity Contexts, and efficiently utilize system resources.

- MVVM is used to manage data flow and separate responsibilities.

Implement logic to handle and respond to hardware and system events that impact the Android Lifecycle.	Beyond MVVM, the application handles system events, such as orientation changes, application switching, notifications, and similar events gracefully including, but not limited to: • Storing and restoring state and information • Properly handling lifecycle events in regards to behavior and functionality ◦ Implement bundles to restore and save data • Handling interaction to and from the application via *Intents* • Handling Android Permissions

- The app handles orientation changes gracefully as it stores typed info as it uses live data that withstand activity and fragment lifecycles and permissions are handled correctly.

Utilize system hardware to provide the user with advanced functionality and features.	Application utilizes at least 1 hardware component to provide meaningful functionality to the application as a whole. Suggestion options include: • Camera • Location • Accelerometer • Microphone • Gesture Capture • Notifications Permissions to access hardware features are requested at the time of use for the feature. Behaviors are accessed only after permissions are granted.

- Application uses the Camera to take the meter photo and upload it, it handles the required permissions well.

- **I have created a google test account that contains data to test Syncing, you can download stored data by clicking the sync button.**

User Name: udacitycaptest@gmail.com

Password: Welcome@2022