



ARCHELON APP

A data collection app for the volunteers of Archelon, The Sea Turtle Society of Greece

ABSTRACT

A brief report describing the implementation of Archelon android application intended for developers that will carry on working on this project and further develop this application.

Salik Tariq (Student)

Wireless and Mobile Computing

Application Licence

MIT License

Copyright (c) 2020 Salik Tariq

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

PRODUCT AND FEATURE VERSIONS

Feature	Version
minimum SDK	24
compile SDK & target SDK	29
build tools	29.0.2
Kotlin	1.3.50
Gradle	6.1.1
Android Navigation Component	2.3.1
Room database	2.2.5

Package:

The base package of the application is: *com.saliktariq.archelon*

The application is developed in layered approach, benefiting with the Android Architecture Components.

App is structured in four layers:

1. Database Layer
2. Repository Layer
3. View Model Layer
4. User Interface Layer

DATABASE LAYER

The package name for database layer is *com.saliktariq.archelon.datalayer*. The database schema is designed as follows:

Entities

Table (Entities)	Description
Authentication	This table contains user authentication information including usernames, passwords, authentication codes that are used to reset passwords along with other information.
AdultEmergence	This table contains the information related to adult emergence. It contains reference to morning survey. Hence a one-to-many relationship between Morning survey and AdultEmergence can be created. That is 1 morning survey can hold more than one AdultEmergence objects.
ActivitySequence	This table contains the activity sequence of an adult turtle. This activity sequence may be related to a nest if the activity is related to a particular nest. This may also be related to an adult emergence hence a reference to AdultEmergence table. ActivitySequence can have many to one relation with AdultEmergence.
MorningSurvey	This table contains the information related to a particular survey hence includes date, time, name of beach, sector and weather information on the day of particular survey.

Location	Location is saved in a separate table (Location) because more than one location can be attributed to a Nest or an event may be such that it can not be attributed to recorded nest or location. Nests may have one to many relationships with Location.
Nest	This table contains information related to a particular nest. Each nest may contain more than one photo (three as per app requirement). This table contains information such as number of eggs, information related to hatching, dead embryos etc
Photos	This table saves photos along with photo name, and the nestID to which this photo is related to. This table may also be used in future to save photos for other purposes as the field nestID is set as nullable in design.

Data Access Objects

The application makes use of the Data Access Objects of Room library to access the data in the database. Therefore, all the data access objects are stored in package:

com.saliktariq.archelon.datalayer.dao

Data access objects related to each entity are contained in separate files.

Database

The application database is placed in package *com.saliktariq.archelon.datalayer.database*. The name of the database is **ArchelonDB**, which is a singleton. Room android library is used to construct the database.

Parcelables

This sub package of datalayer contains objects that are to be passed between fragment in order to pass the data between fragments.

REPOSITORY LAYER

This layer contains the repository for the application placed in package *com.saliktariq.archelon.repository*. The repository contains functions that will be used to access the database tables. This layer of abstraction helps separate the upper layers from the datalayer. All functions in the repository are set to run on Dispatchers.IO by default. Repository functions are created for all the functions defined in entity classes.

VIEWMODEL LAYER

View Models are created for all the UI classes that handle any kind of data. These view models are generated using relevant view model factory classes. Their operations in this application are further explained in the User Interface Layer section of the document where it is more relevant.

USER INTERFACE LAYER

Atom UI library is utilised to create the user interface of the Archelon application. The user interface classes are placed in package *com.saliktariq.archelon.ui*. The application makes use of the Navigation

Components of android. The MainActivity of the application references to the navigation graph (nav_graph.xml) which initialises the first fragment and corresponding layout file. An overview of navigation graph is shown in the figure 1:

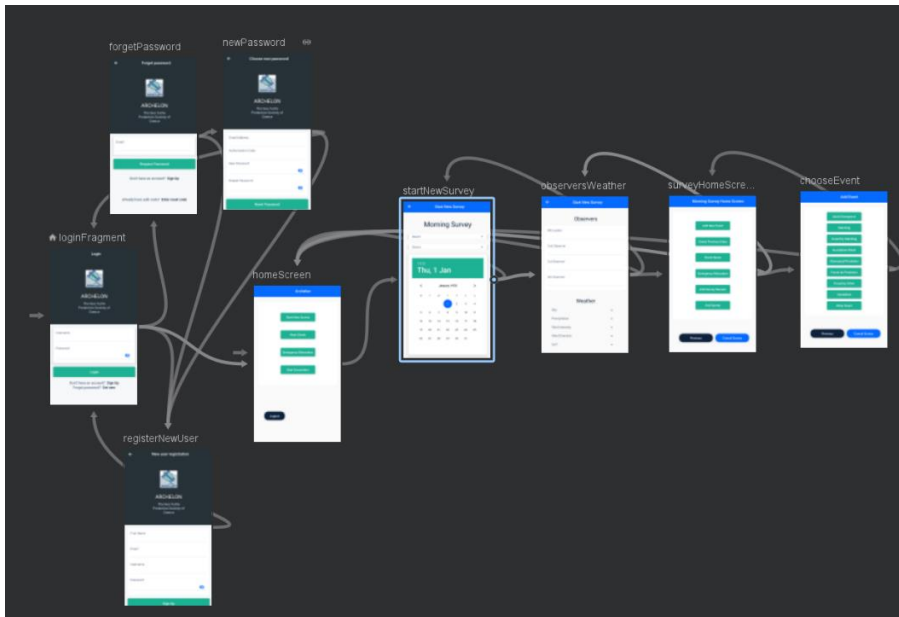


Figure 1: Archelon Navigation Graph

The first fragment is the LoginFragment. It allows the user to either log in the system, sign up for a new account or retrieve password. Figure 2 shows a screenshot of LoginFragment displayed on an android system:

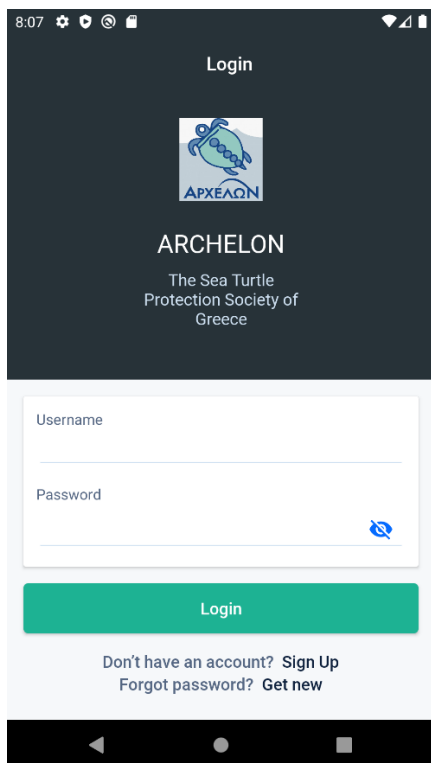


Figure 2: LoginFragment showing login screen

Should the user choose to create a new account, they can do so by clicking Sign Up clickable text at the bottom, which takes to the RegisterNewUser fragment as shown in figure 3:

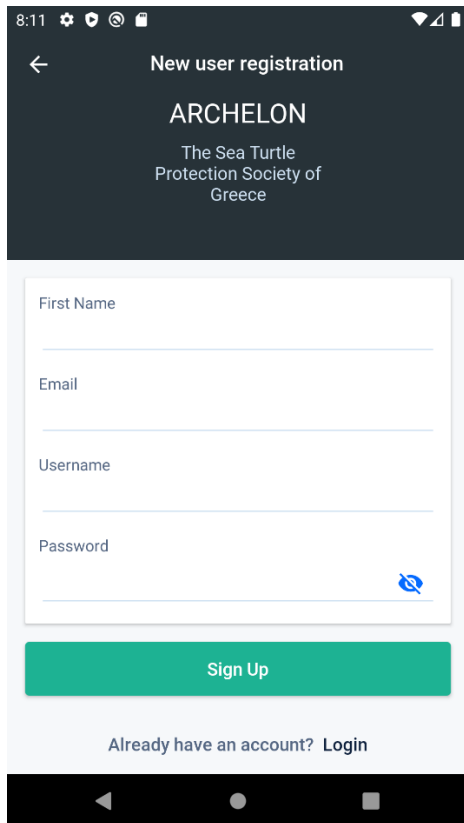


Figure 3: RegisterNewUser Fragment

Once user enters their details to register a new account, the view model associated with RegisterNewUser fragment make query to checks if this username or email address already exists in the system, if it does then it gives the user a warning as a toast as shown in figure 4:

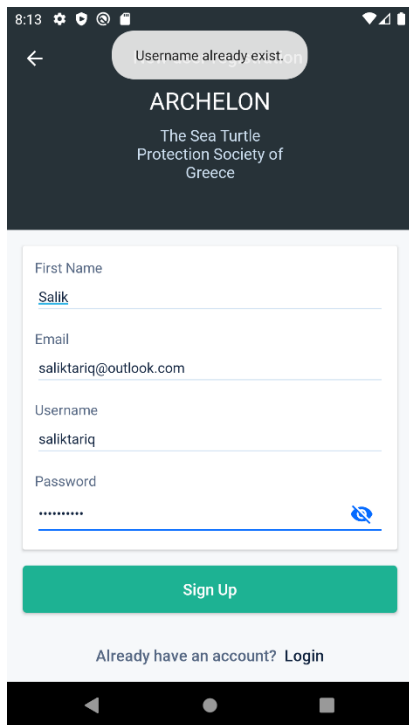


Figure 4: Username already exists Toast

Whereas, if all the checks are made and there is no data conflict a new user is created and the user of the app is navigated to the LoginFragment.

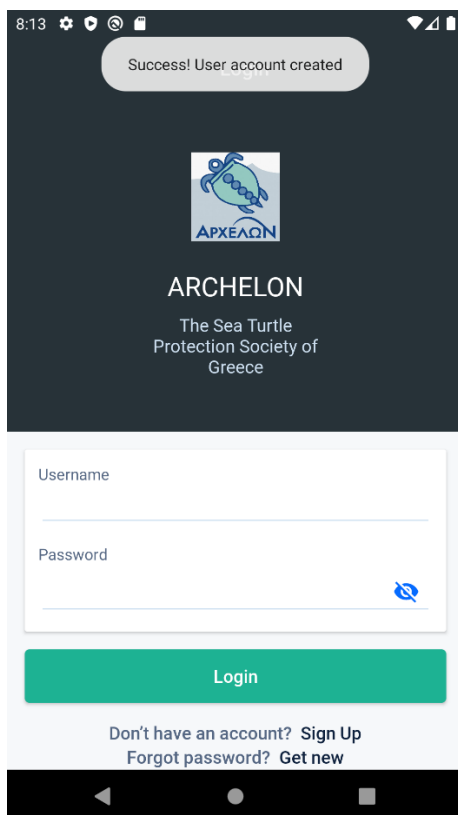


Figure 5: Success message and user sent to LoginFragment

Note: Plain text password is not saved in the database. The hash of the password is generated and saved in the database and every time a user initiates login sequence, the provided password is converted into its respective hash code and this hash code is compared with the saved hash code of the password. Upon successful creation of account, user is also emailed with their account details.

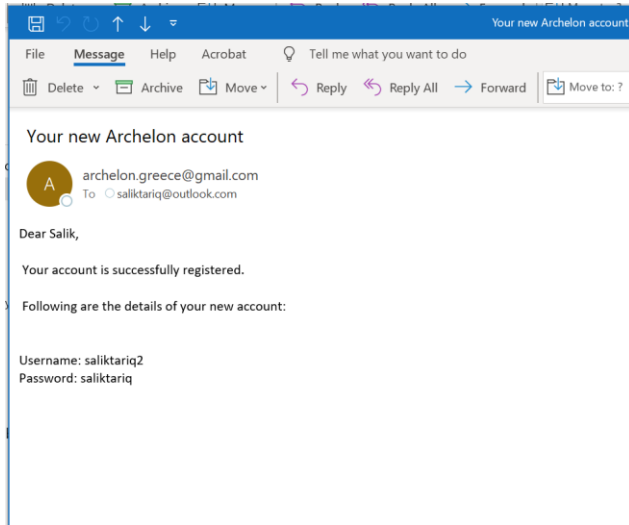


Figure 6: Email received upon successful signup

Should the user choose to reset password, they can do so by clicking the Forget password? **Get new** clickable text, which takes them to ForgetPassword fragment. When user clicks Request Password button, system checks if an account is associated with provided email address, in case of an existing account, an 'auth code' is sent to the user's registered email address and user is navigated to NewPassword fragment.

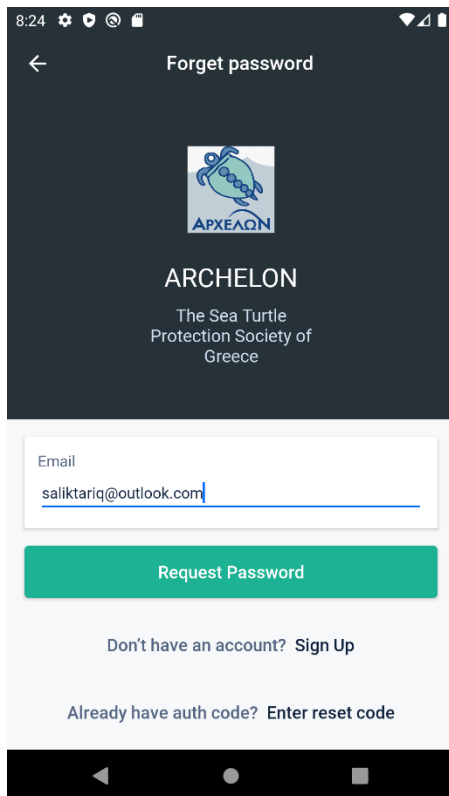


Figure 7: ForgetPassword Fragment

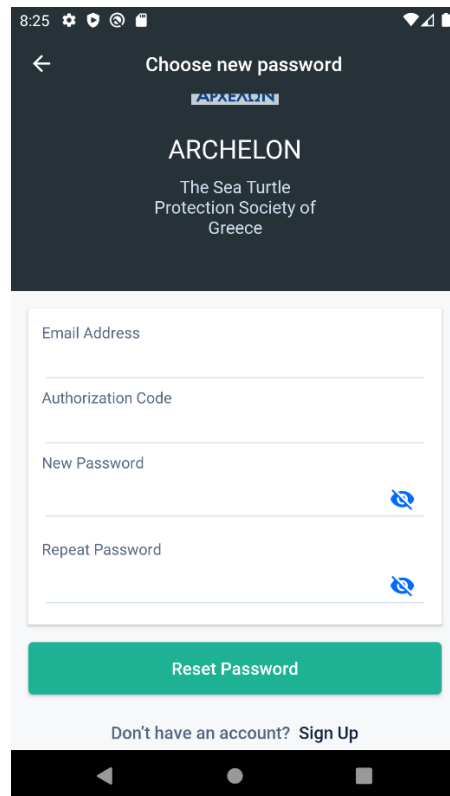


Figure 8: NewPassword Fragment

NewPassword fragment can also be accessed through ForgetPassword fragment following the '**Enter reset code**' link at the bottom of the screen. App emails the user with reset code and it is to be entered in the Authorization Code field of the NewPassword fragment.

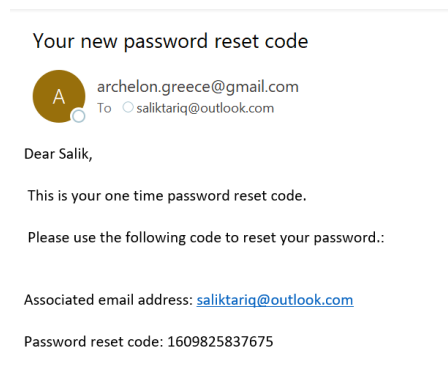


Figure 9: Password reset code email

Once the user successfully reset the password, the application recalculates a new auth code and associates it with the user. User is also emailed with their new password upon successful password reset process described above.

Upon successful login, user is navigated to `ApplicationHomeScreenFragment` which presents user with four options as in Figure 10:

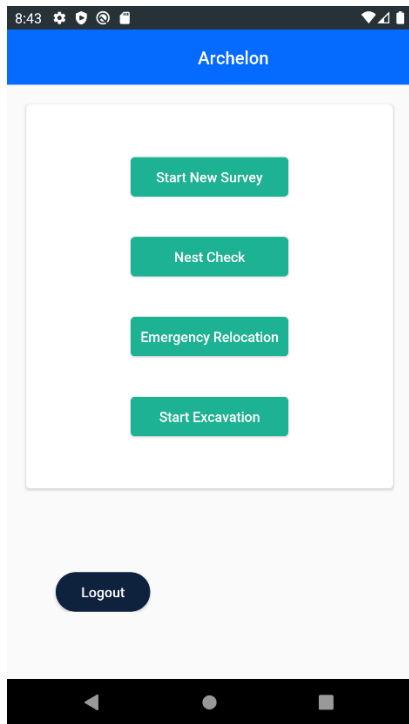


Figure 10: `ApplicationHomeScreenFragment` UI display

The 'Start New Survey' button takes user to StartNewSurvey fragment. The other three buttons are not implemented and will present a toast upon clicking stating that 'Feature not implemented'.

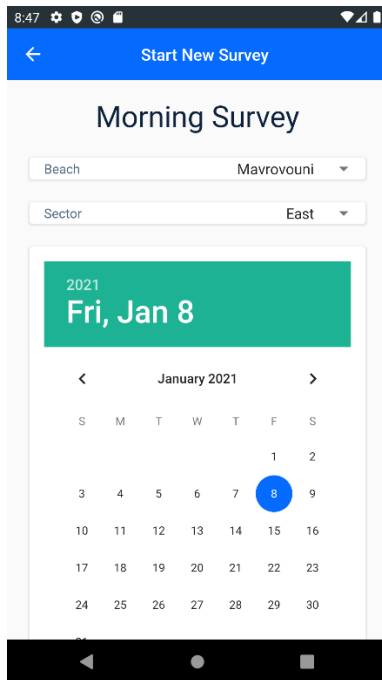


Figure 11: StartNewSurvey Fragment UI display

Here in this screen, the user chooses the beach, sector, date and time. And then clicks '+' icon to start a new morning survey. At this point, the data saved from StartNewSurvey is converted into a parcelable object and passed to the next fragment as SafeArgs. This safeargs data is passed to the ObserversWeather fragment as shown in figure 12:

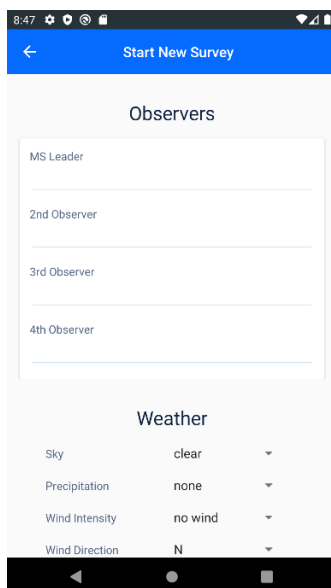


Figure 12: ObserversWeather fragment UI display

On ObserversWeather fragment, user has option either to go back, cancel survey or proceed to next screen. In case the user goes back, the received parcelable object is sent back to the StartNewSurvey

fragment to repopulate the UI fields. Kindly note, the date and time are not sent as they are updated constantly on the UI to represent the current date and time and forcing them to be updated with a previous entry might cause errors in collecting correct date and time in the application. Upon pressing next button, user is navigated to SurveyHomeScreen fragment which presents user with six further choices. Please see figure 13 for SurveyHomeScreen:

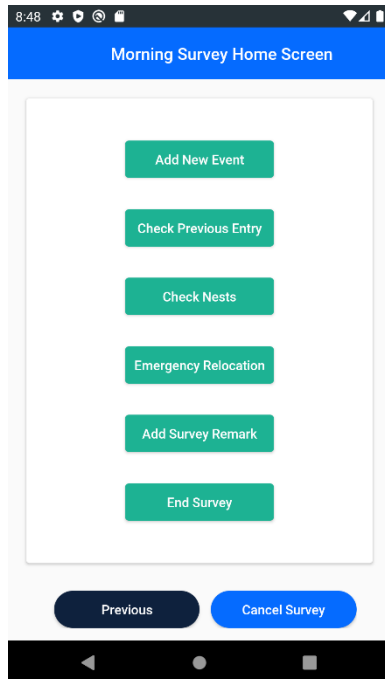


Figure 13: SurveyHomeScreen fragment UI design

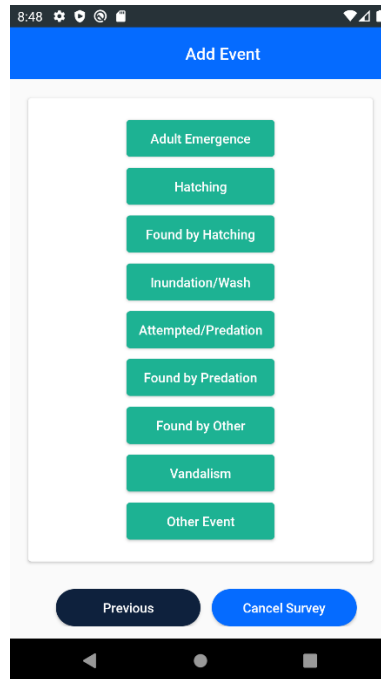


Figure 14: ChooseEvent Fragment UI design

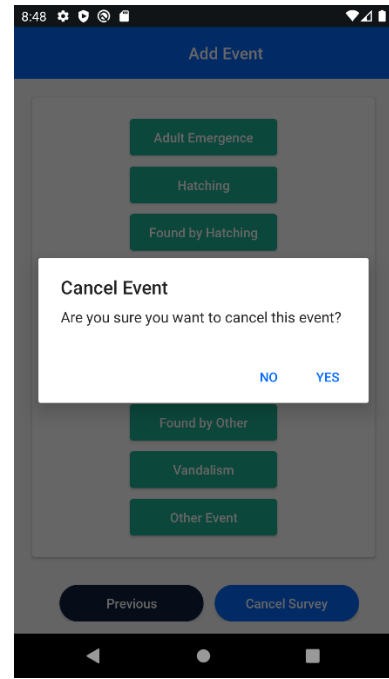


Figure 15: Cancel Dialogbox to take user back to SurveyHomeScreen fragment

When user clicks add new event, it takes them to ChooseEvent fragment (Figure 14) which presents a choice of entering nine type of events in the system, these functionalities are not implemented and therefore each of these buttons present with a toast stating, 'Feature not implemented'.

Alternatively, the user can end survey at that point by clicking 'End Survey' button which will save the Morning Survey information collected up till that point and save it in the database, and direct user to the SurveyHomeScreen fragment, see figure 15.

USER INTERFACE LAYER

Project contains package named **helper** at *com.saliktariq.archelon.helper* which contains helper classes that are created for convenience and carrying on repetitive tasks.

AppContext: This class is registered with the AndroidManifest and is loaded when the application starts. This class provides with Application context where necessary, it also initialises the Repository class so that a single instance of Repository class is available from the start of the application.

AuxiliaryFunctions: This class contains the function to generate authentication code for user. The algorithm to generate authentication code is as follows:

- 1) Generate hash code for username, password, email address, first name.
- 2) Add the sum of hash codes generated in step 1 to current time in milliseconds to create randomness.

- 3) Trim the length of the hash code to 13 characters.

GmailAuthDetails: This class contains the credentials for the email address that is used to send emails to users.

MyToasts: This class contains two implementations of Toasts to simplify the toast code usage in the application.

SendMail: This class is used to send email messages to the user.

ANDROID TESTS

Application contains comprehensive unit tests to test the successful functionality of the seven entity classes. These are instrumented tests that create in-memory database and destroy the database upon completion of the tests.

This app also contains Repository tests that create an instance of ArchelonDB and make the repository calls. This database is not an in-memory database, hence the tests are written to account for this, so that no data integrity errors are generated upon re-running the tests repeatedly as same data would be inserted in the database.

CAVEAT

The application is tested on real devices running Android API 28 and API 29. The email functionality of the application is restricted to all devices running up to API 29, which is Android Q (Android 10). Some type safety checks are added to Repository that seem redundant, they are done