**Moneytrack**

**Documentation**

**Moneytrack is a personal finances management application designed to allow the user to document income and expense streams and view reports containing aggregates of those streams.**

**Created by Apostolos Boutos as part of his thesis in his undergraduate studies in the Technological Institute of Larissa.**

**Moneytrack is available for free at Google Play:**

**https://play.google.com/store/apps/details?id=com.apboutos.moneytrack**

**Index**

1. Implemented Features…………………………………………………… 3
2. Future Features……………………………………………………………… 5
3. Tools and Libraries…………………………………………………………. 7
4. Application Architecture………………………………………………. 13
5. Activity Flow Diagram…………………………………………………… 16
6. Project Structure……………………………………………………………21
7. Class Diagram………………………………………………………………..23

**Implemented Features**

* **Authentication System**

In order, for the user to use Moneytrack a personal account must be created first. The account is necessary because the application is designed to be accessible from multiple devices and/or by multiple users from the same device. The first time the user installs the application she is prompted to create a new account and upon successful account registration she is prompted to login to her account from her device. The first time a login takes place an authentication request is sent to the server and upon successful authentication the user’s credentials are stored in the local database. This way an internet connection is no longer necessary for further use of the application.

* **Finance logging**

After the authentication has been completed the user is redirected to the application’s main screen. The current day’s log is presented, and the user can proceed to log that day’s entries, containing incomes or expenses. All the entries for the given day are presented to the user in a scrollable list format. The user can edit the contents of the list by clicking on a single entry and/or also delete an entry by swiping left or right on it. The user can move to past or future dates by using the arrows on the top of the screen, or by clicking the calendar menu icon.

* **Aggregate Report**

By clicking the report menu icon, the user is presented by a list of aggregates. The current date is presented on the left side and entries sums the correspond to the current day, month or year are presented on the right side. The day, month and year of the current date are presented inside spinners and the user can move to another date simply by touching the respective spinner and selecting a new value. The aggregates on the right side are automatically recalculated after a date change. On the bottom of the screen the user’s lifetime aggregates are displayed.

* **Customized Search**

By clicking the search menu icon, the user is presented with a search dialog whereas she can specify certain search criteria like a period of time, the type, description and/or category of entry. By clicking the search button, the local database is queried for the appropriate entries and they are displayed in a scrollable list.

* **Data remote storage and synchronization**

The application uses a local SQLite database to store the user’s data. A local database cuts down on the internet usage and also allows the user to use the application without and internet connection. Because the application is designed to be used by multiple devices a copy of each user’s data is also stored in a remote database. This way the user can access her data from any devices and their integrity against accidental loss is further secured. An automatic data synchronization mechanism is implemented. The application upon start requests all new or modified entries from the remote location and upon pause sends to it all new local modifications.

**Future Features**

* **Migration of the remote database to Google’s cloud platform Firebase**

The current version of the application uses a remote nginx server to host the database and the communication scripts. This introduces maintenance issues as well as problems with the data synchronization when the application is used concurrently in many devices. Moving the backend to Google’s Firebase will solve all those problems as Google provides implementation of these features for free.

* **Implementation of a Settings screen**

A settings screen must be implemented so the user can customize the application according to his needs. Some customization options can include the selection of a certain theme, selection of currency, different options to delete an entry etc.

* **A selectable night and day theme**

According to Google’s material design principles an application must have a day and night theme. Depending on the room brightness and other factors like the user’s eyesight ability an ability to select between and a day and night theme can increase the applications accessibility.

* **The report dialog remodeling**

The report dialog must be remodeled to include a better graphical presentation, description tabs for each aggregate, category, type and description filters, as well as more presentation methods like a time graph or a pie chart.

* **A tutorial**

Implementing a short tutorial to be displayed the first time the user installs the application can help to familiarize the user with the core elements of the application.

* **Authentication with Google or Facebook**

Implementing user authentication by using Google’s of Facebook’s automatic authentication saves the user time. Most people dislike creating the process of creating an account and/or remembering their login credentials and prefer login in by using their Google or Facebook account.

* **Recuring income/expenses**

The user should be able create recurring and expenses and incomes like monthly paychecks or rent bills. These entries should be added to their respective dates when that date arrives or in advance if the user chooses so.

* **Quick date summary at the calendar screen**

The calendar screen is currently used only as a way for the user to quickly reach remote dates.

The is more potential for this screen. Upon prolonged touching of each day a quick summary of that day’s incomes and expenses could appear. Upon double touch the day should be selected without the user having to press the Pick Day button, this saves time. Also, the number of each day and each month should be colored red or green to quickly inform the user that there are entries in that specific day. The color of the day can also inform the user if he had a monetary surplus of a deficit that day without having to select it.

* **An about screen for easy contact with the developer**

A quick form should be implanted in an about screen, so the users can leave direct comments and feedback.

**Tools and Libraries**

* **Android Studio**

The application has been developed using Android Studio. Android Studio is an Integrated Development Environment (IDE), an aggregation of different tools that increase the efficiency and productivity of a programmer. Those tools include syntax highlighters for quicker identification of syntax errors, intelligent code completion allowing for higher productivity when writing code, integrated documentation tools to quickly read or produce documentation for your classes, debuggers, database inspectors, device emulator etc.

Android Studio is a customized IDE for the production of Android mobile application. It is provided by Google and is based on the IntelliJ IDE provided by JetBrains.

For more information on Android Studio refer to:

[**https://developer.android.com/studio**](https://developer.android.com/studio)

[**https://en.wikipedia.org/wiki/Android\_Studio**](https://en.wikipedia.org/wiki/Android_Studio)

For more information on IntelliJ refer to:

[**https://www.jetbrains.com/idea/**](https://www.jetbrains.com/idea/)

[**https://en.wikipedia.org/wiki/IntelliJ\_IDEA**](https://en.wikipedia.org/wiki/IntelliJ_IDEA)

* **Git**

Git is a Version Control System created by Linus Torvalds. It allows programmers to save different versions of their code files and quickly search and move to previous versions when they introduce bugs due to changes in the codebase. This increases productivity as it allows many programmers to work on different branches of the same project and allows many bugs to be discovered and eliminated before they find their way to the main branch of the project. Git is integrated in Android Studio for easier use.

For more information on Git refer to:

[**https://git-scm.com/about**](https://git-scm.com/about)

* **GitHub**

GitHub is an online repository for Git projects. By having a backup copy your codebase stored on a remote server you ensure that you can never lose your files due to accidents, and you can also share your code and collaborate with others on the same project much easier.

For more information refer to:

[**https://github.com/**](https://github.com/)

* **Nginx**

Nginx is a free open source software that can be used as a web server. It was created and by Igor Sysoev and released in 2004. Nginx is used to run the backend scripts of the project that facilitate the remote data storage and synchronization.

For more information refer to:

[**https://www.nginx.com/**](https://www.nginx.com/)

* **Gradle**

Gradle is a build automation tool. Compiling your project is not an easy task and it is a very time consuming and repeatable process, especially when your project includes many third party libraries and other dependencies. Gradle simplifies this process by allowing the programmer to write Gradle scripts. These scripts then can be used to easily build, cleanup and debug the project with a click of a button. Also the management of dependencies becomes much easier as the programmer can now quickly switch to newer or older version of a dependency like a third party library by simply changing a line of code in the Gradle build script. It is an essential tool that cuts down tremendously on the process of building a project. Gradle is integrated in Android Studio for easier use and comes preinstalled with it.

For more information refer to:

[**https://gradle.org**](https://gradle.org)

* **Kotlin**

Kotlin is an object oriented (OO) programming language developed by JetBrains that runs on the Java Virtual Machine (JVM). It is the preferred programming language for writing android applications according to Google since May 7 2019, replacing Java as the primary language. In essence, Kotlin is a very similar language to Java and was designed to be a better version of Java, bringing many syntactical upgrades that reduce boilerplate code and make the language more expressive as well many structural upgrades like default object immutability that push programmers to new and more efficient paradigms of writing code like functional programming. Kotlin is fully interoperable with Java code, meaning you can easily introduce new Kotlin code to an existing Java project, or easily migrate an entire Java codebase to Kotlin using automated scripts. Kotlin comes preinstalled with Android Studio.

For more information refer to:

[**https://developer.android.com/kotlin**](https://developer.android.com/kotlin)

[**https://en.wikipedia.org/wiki/Kotlin\_(programming\_language)**](https://en.wikipedia.org/wiki/Kotlin_(programming_language))

* **php**

php is a multipurpose scripting language mainly used for writing backend web scripts. All the backend code that implements the remote data storage and synchronization functionality is written in php and served through an nginx web server. php was selected over Java as it is much easier and faster to write and deploy a small size project with it, instead of creating a proper API using a Java technology like Spring MVC.

For more information refer to:

[**https://www.php.net/**](https://www.php.net/)

* **SQL**

SQL is a domain specific programming language used in managing and structuring data stored in a Relational Database Management System (RDBMS) like SQLite or MySQL. In the project SQL is used to communicate both with the integrated SQLite local database on the android device as well as the remote MySQL database on the server. SQL is used to create the database schema and to perform CRUD (Create, Read, Update, Delete) operations by communicating with the RDBMS.

For more information refer to:

[**https://en.wikipedia.org/wiki/SQL**](https://en.wikipedia.org/wiki/SQL)

* **Android SDK**

The Android Software Development Kit (SDK) is a collection of tools and libraries provided by Google that allow programmers to write android applications. It contains many classes like Activity or Service that provide the foundation of each android application, tools like the emulator which allows the programmer to easily deploy and test the application in a multitude of android environments, a management tool to keep track of the different versions of the SDK and documentation. The Android SDK provides support for Kotlin, Java and C++ and comes preinstalled with the Android Studio.

For more information refer to:

[**https://developer.android.com/docs**](https://developer.android.com/docs)

* **Kotlin Coroutines**

Coroutines is a Kotlin library that simplify the creation and use of asynchronous operations in android applications. In was created to provide an alternative, more robust and easier method of using asynchronous operations than the old Android AsyncTask class. Use of Kotlin Coroutines makes the code much more expressive and readable and tremendously reduces the boilerplate code associated with instantiating and using asynchronous objects with the old Android libraries.

For more information refer to:

[**https://developer.android.com/kotlin/coroutines**](https://developer.android.com/kotlin/coroutines)

* **Room**

Room is persistence library that provides an abstraction layer over SQLite. It allows easy mapping of SQL tables to Plain Old Java Objects (POJOs), as well as the automatic creation and implementation of the SQLite database schema and it’s migration to newer database versions. Using code reflection, Room provides concrete implementations of abstract interfaces the programmer writes using annotations, thus reducing a lot the complexity and the boilerplate code required to communicate with the embedded SQLite database.

For more information refer to:

[**https://developer.android.com/jetpack/androidx/releases/room**](https://developer.android.com/jetpack/androidx/releases/room)

* **Retrofit**

Retrofit is an opensource library designed to allow the easy consumption and provision of HTTP calls. Using custom interfaces defined by the programmer with special annotations, Retrofit automates and streamlines the production and consumption of HTTP calls by taking care of operations like mapping model objects to JSON objects using parsers, handling the creation of HTTP connections etc. It is an industry standard library for connecting an android application with it’s backend API.

For more information refer to:

[**https://square.github.io/retrofit/**](https://square.github.io/retrofit/)

* **JUnit**

Software testing refers to the process of testing a codebase by using automated testing methods to ensure that the codes behaves as intended and to discover and fix errors before they become apparent to the end users. JUnit is a Java library for automated testing that provides support for writing and executing Unit Tests. Unit Testing is a method of software testing where small segments of the code like functions and methods are tested against predefined inputs and outputs to ensure correct behavior. By using automated tests the codes becomes much more robust and maintainable, errors are reduced and it becomes much easier to verify that new additions of code or modification of old code file does not introduce new undetected errors.

For more information refer to:

[**https://en.wikipedia.org/wiki/Unit\_testing**](https://en.wikipedia.org/wiki/Unit_testing)

[**https://junit.org/junit5/**](https://junit.org/junit5/)

* **Espresso**

Espresso is an Android library provided by Google that allows developers to easily write and deploy User Interface (UI) tests. Testing the user interface of an android application can be quite complicated and cannot be done using standard unit testing libraries like JUnit because instantiation of the classes to be tested requires the instantiation of an android application with a running lifecycle. Espresso provides support for UI testing by providing all necessary tools required to write UI tests and deploy them on different devices using the android emulator. With Espresso you can easily create automated test suits that verify the correct appearance and functionality of all the application’s UI elements.

For more information refer to:

[**https://developer.android.com/training/testing/espresso**](https://developer.android.com/training/testing/espresso)

**Application Architecture**

**MVVM**

The projects uses the Model-View-ViewModel architectural pattern. This is the standard modern architectural pattern than encompasses the best practices for structuring and android application’s code. The MVVM architecture is derived from the Model-View-Controller (MVC) architecture first used in the creation of desktop applications. Unlike desktop applications who have a single point of entry, android applications have multiple points of entry, as the different android activities can be designed to start separately from the rest of the application. This feature of the Android OS allows an application to communicate with different parts of other applications that perform specific subtasks. For example, to share an image file from your application to facebook, you can broadcast the image to the facebook application installed in the device. This boots a specific Activity in the facebook app whose job is to post images on the user’s account and is different from the Activity used to boot the facebook application.

These differences in system design let do a differentiation of the standard MVC architecture and the creation of the MVVM pattern.

****The MVVM architecture is best described with the image bellow.

image source:[**https://developer.android.com/jetpack/guide**](https://developer.android.com/jetpack/guide)

**View**

The View is the component that includes all the activities and fragments of the application that are responsible for creating and updating the user interface. In older applications that do not use the MVVM pattern the Activity classes also contain code that manipulates the application’s data and thus they tend to be big and cluttered, which makes them hard to maintain and makes the code much more difficult to understand. By following the principle of separation of concerns using the MVVM pattern, Activities are decupled by code that is responsible for data manipulation and their only purpose is to implement the UI and handle operating system interactions like the user input events.

**ViewModel**

The ViewModel is the layer that stands between the activities and the data. It responsible from providing data to the View layer so they can be displayed through and Activity of Fragment. The ViewModel can also manipulate the data, perform all kinds of business logic, or forward user requests to modify existing data. The Android SDK provides a specific class called AndroidViewModel to represent this layer of abstraction. AndroidViewModel is decupled from the Activity class. This means that when an Activity changes configuration or is destroyed and recreated the AndroidViewModel is not affected as it knows nothing about the Activity as depicted by the dependency arrows in the above diagram. This way the Android OS can freely destroy and recreate an Activity when it decides to do so while the Activity’s data can be easily retrieved in their proper state from the recreated Activity by accessing the ViewModel.

The ViewModel is also the place where LiveData objects are instantiated and live. LiveData are observable objects hat are designed as data holders that notify all observers when their state has changed using callback methods. This way, different components of the system, like Activities can have access to these data without creating rigid and explicit dependency paths between them. LiveData objects also respect the lifecycle of activities and include cleanup methods that prevent memory leaks and excessive memory consumption. Prior to the introduction of the LiveData class third party libraries like RxJava were used to provide this functionality.

**Model**

The Model contains data classes that represent the different entities of the domain. These data classes are persistent, meaning they are stored in the local SQLite database on the android device. Driving the UI from persistent model classes has many advantages such as allowing it work independently of an internet connection and, also ensures that the users do not lose data if the Android OS decides to destroy the Activity.

**Repository**

The Repository is an abstraction layer that stands between the ViewModel and the data source. The Repository is responsible for handling all the data requests from the ViewModel and forward them to the appropriate data source. In most android applications there are two different data sources that are used, a local and a remote one. The local data source is used as a data cache to provide the different Activities with immediate and quick access to the underlying data and, also to allow the application to function to some extent without and internet connection. The remote data source is most applications the real source of new data. When new remote data are available and there is an established internet connection the application can query the remote data source and then in the background update the local data cache with the new entries.

The repository layer is the place where all the classes associated with Room and Retrofit live.

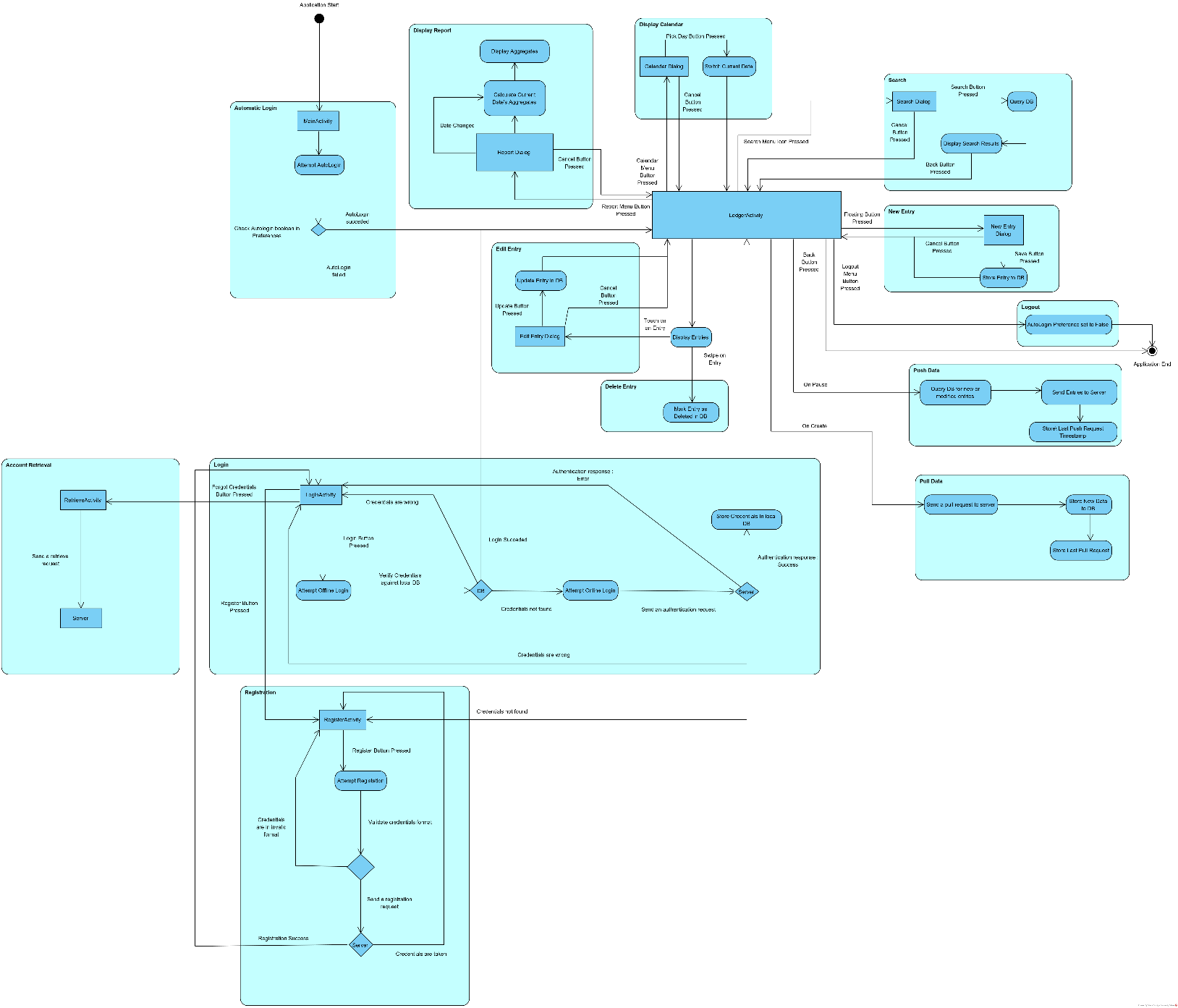
**Activity Flow Diagram**

Diagram created with Visual Paradigm 16: <https://www.visual-paradigm.com/>

The diagram can be found in a better quality in the project’s GitHub page: <https://github.com/exophrenik/Moneytrack2/blob/master/Documentation/Activity%20Flow%20Diagram.png>

**Note:** When using the word activity below, we are referring to UML activities and not Android activities. An Android activity will be referred to as an Activity, with a capital A to denote the difference.

**AutoLogin activity**

Moneytrack has a single place of entry, the MainActivity. When the application is booted the MainActivity class is instantiated. This Activity has no user interface and is used mainly to implement the auto login activity. Upon creation of the Activity the autoLogin property in the autoLogin shared preference file is queried and if enabled, the application skips the login part and automatically transfers control to the LedgerActivity using the stored user credentials. If the autoLogin property is not enabled control is transferred to the LoginActivity.

**Login activity**

In this activity the LoginActivity class instantiated and user is served a login interface where he can input his username and password. If the login button is pressed a login process with be initiated. First the system will use the local database to authenticate the user. If the user’s credentials are wrong, he will be served an error message. If the credentials are not found in the local database, it means the user has not logged in form this device in the past or has chosen not save his credentials and therefore an online authentication will be attempted. If the server responds that the credentials are wrong, and error message will be served, while if the credentials are not found in the remote database the user will be prompted through a message to create an account. In case of successful authentication, local of remote, the control is transferred to the LedgerActivity. The user can also click the “Ι forgot my username and password” text where the control is transferred to the RetrieveActivity or he can click the “I don’t have an account” text where the control is transferred to the RegisterActivity.

**Account Retrieval activity**

In this activity the RetrieveActivity class is instantiated, and a request for account retrieval containing the user’s username is send to the server. If the server finds the username in the remote database, it emails the user’s credential to the registered email address.

**Registration activity**

In this activity the RegisterActivity class is instantiated, and the user is served a register interface. After filling all the respective fiends and the provided values pass format check, a registration request is sent to the server containing the user provided values. If the values pass all the server checks a new account is created and the control is transferred back to the LoginActivity, otherwise an error response is sent, and the user is informed about the failed registration attempt and the reason though an error message.

**Entry Display activity**

Upon successful user authentication control is transferred to the this activity and the LedgerActivity class is instantiated. The user served with an interface that contains the current day, buttons to move to a new date, a menu with icon buttons, a new entry floating button, as well as a scrollable list view containing all the entries of the current date.

**New Entry Activity**

By clicking on the floating button control is transferred to this activity and a Dialog is instantiated where the user is served an interface to provide the information of a new Entry. If the Save button is clicked the new entry is saved in the local database, while if he clicks the cancel button, the values of the fields are discarded. In both cases control is returned to the Entry Display activity.

**Edit Entry Activity**

By clicking on an entry in the list view control is transferred to this activity and a Dialog is instantiated where the user is served an interface where he can replace the existing values of that Entry with new ones. If he clicks the update button the Entry in the local database is updated with the new values, while if he clicks the cancel button the new values are discarded. In both cases control is returned to the Entry Display activity.

**Delete Activity**

When the user swipes left or right in a list entry that entry is marked as deleted in the local database.

**Display Report activity**

By clicking on the scroll menu icon the user transfers control to this activity, a Dialog is instantiated and the user is served an interface where he can see Entry aggregates for different dates. The aggregates are first calculated by querying the local database and then they are displayed in the dialog. If the user choses to change the date using the respective spinner the aggregates are recalculated and redisplayed. By clicking the back button or touching outside the space Dialog control is returned to the Entry Display activity.

**Display Calendar activity**

By clicking the calendar menu icon the user transfers control to this activity, a Dialog is instantiated and the user is served an interface where he can see a calendar and pick the day he wishes. This dialog enables the user to quickly reach far away dates. After selecting a date if the user clicks the pick date button the date in the LedgerActivity changes and the list of entries is updates to reflect the new date. If he clicks the back button, or the space outside the Dialog the date is not changed. In both cases control returns to the Entry Display activity.

**Search activity**

By clicking on the search menu icon the user transfers control to this activity, a Dialog is instantiated and the user is served an interface where he can set a list of search criteria. By clicking the search button a query is dispatched to the local database and a list of entries matching the specified search criteria is returned and displayed in the LedgerActivity. The list remains visible until the user clicks the back button where the control is returned to the Entry Display activity.

**Logout activity**

By clicking the logout button in the extended menu the user transfers control to this activity. The autoLogin property in the autoLogin preference file is set to false and the application exits.

**Push Data**

Control to this activity is transferred whenever the onPause method of the LedgerActivity is called. When this happens the local database is queried and all the entries that have been created or modified at a time later to the time saved in the lastPushDatime property of the user’s preferences file, are gathered and sent to the remote server for storage. Upon successful execution of that request the lastPushRequestDatetime property is updated with the current timestamp.

**Pull Data**

Control to this activity is transferred whenever the onPause method of the LedgerActivity is called. The lastPullRequestDatetime property of the user’s preferences file is queried and sent to the server to begin a pull request. The server in turns responds by sending a list of entries that have been modified after the lastPullRequestDatetime. Those entries are then stored in local database and/or replace the previous entries if they are most recent versions of them. The lastPullRequestDatetime property is then update with the current timestamp.

**Project Structure**

**Directories**

Moneytrack2/ The root directory

.gradle/ Contains all the internal files of the Gradle builder.

.idea/ Contains various project preferences for the Android Studio.

build/ Contains autogenerated cache files for faster project building

documentation/ Contains all the documentation files associated with the project.

Like the diagrams and this document.

gradle/ Contains the gradle.jar file.

external libraries/ Contains the source files of all the external libraries used in the project.

app/ Contains the application’s main module.

app/build Gradle build files for the current module.

app/release This is where the artifacts produced by building the application are stored.

app/libs Contains libraries specific to this module.

app/src Contains the source files of the module.

app/src/androidTest/java Contains the source files of the Unit Test associated with the Android

platform, like the UI tests.

app/src/test/java Contains the source files if the regular Unit Tests.

app/src/main/java Contains the source files of the current module.

app/src/main/res Contains all the resources used by the application like layout files,

drawables, strings, values etc.

**Packages**

com.apboutos.moneytrack Root package

.boot Contains the entry point of the program the MainActivity.class

.view Contains the classes associated with the View layer.

.viewModel Contains the classes associated with the ViewModel layer.

.viewModel.receiver Contains the BroadcastReceiver classes.

.utilities Contains various utility classes.

.utilities.converter Contains classes that perform data conversions.

.utilities.error Contains error enumerations.

.model.database Contains the classes associated with the model layer.

.model.database.converter Contains classes that perform data conversions on entity objects.

.model.database.dao Contains the Data Access Object interfaces used by Room.

.model.database.database Contains the abstract database class.

.model.database.enity Contains the classes that represent the model entities.

.model.repository Contains the classes associated with the repository layer.

.model.repository.local Contains the classes that communicate with the local repository.

.model.repository.remote Contains the classes tat communicate with the remote repository.

**Class Diagram**

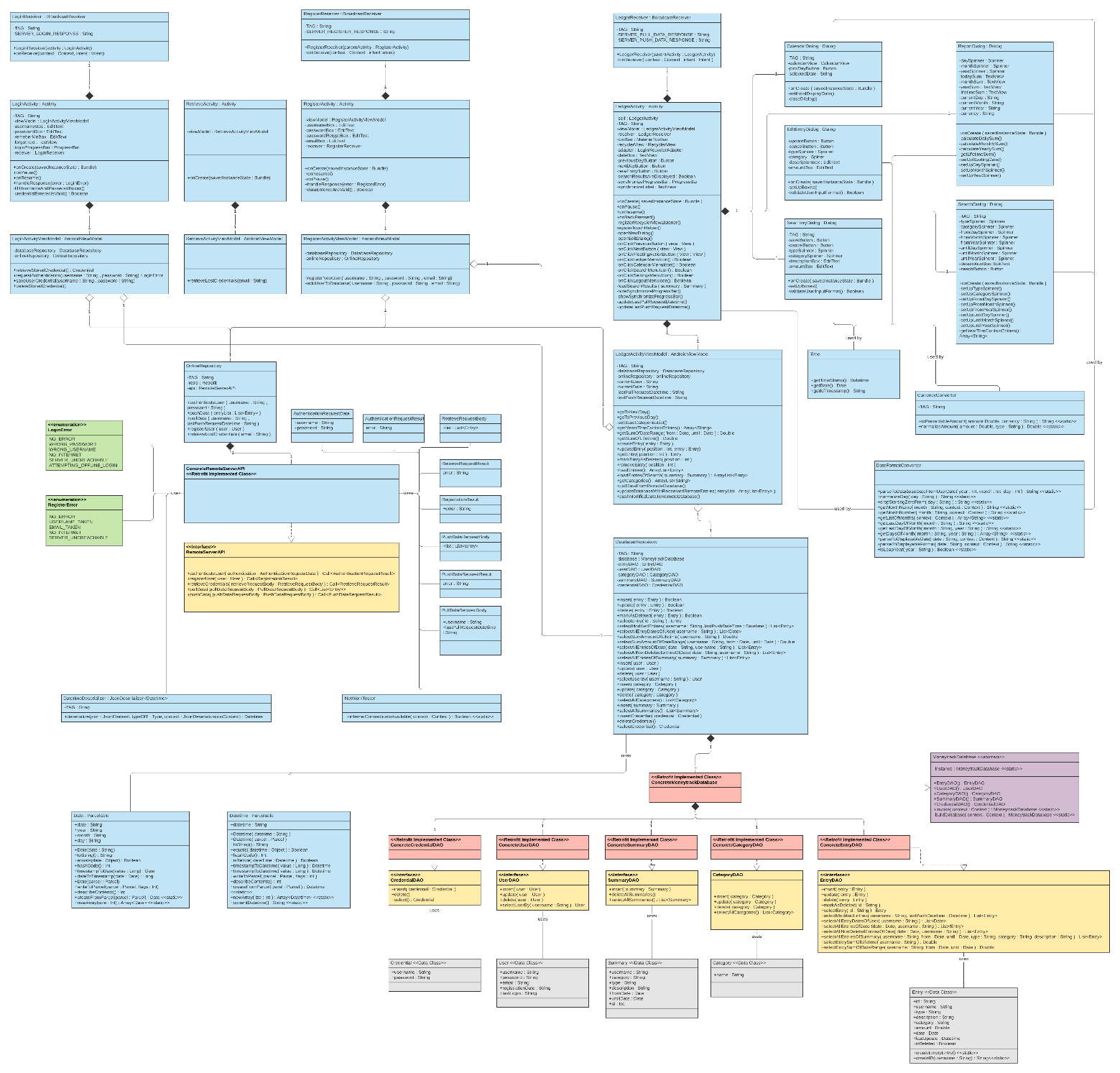


Diagram created with Lucidchart : https://www.lucidchart.com

The diagram can be found in a better quality in the project’s GitHub page: <https://github.com/exophrenik/Moneytrack2/blob/master/Documentation/UML%20class%20diagram.png>