Bengal College of Engineering and Technology
Durgapur, 713212

Paper: CS504D (Object Oriented Programming)

Implementation of MVVM design using Android Architecture Components in a Task Manager app

Mini-project for the partial fulfillment of course work of *CS504D (Object Oriented Programming)* in 5th semester of B. Tech in Computer Science

Submitted by,

Debaleen Das Spandan,
University Roll Number: 12500117078
Aditya Kumar,
University Roll Number: 12500117110
Acquib Javed,
University Roll Number: 12500117112

Submitted to,

Biswajit Gope
Asst. Professor
Computer Science and Engineering
Bengal College of Engineering and Technology

Bona fide

This is to certify that this project report entitled “Implementation of MVVM design using Android Architecture Components in a Task Manager app” submitted to Bengal College of Engineering & Technology, is a *bona fide* record of work done by, Debaleen Das Spandan, University Roll Number: 12500117078; Aditya Kumar, University Roll Number: 12500117110 and Acquib Javed, University Roll Number: 12500117112 under my supervision.

Biswajit Gope

Asst. Professor

Computer Science and Engineering

Bengal College of Engineering and Technology

Place:

Date

Acknowledgement

I would like to place on record my deep sense of gratitude to **Mr. Biswajit Gope**, *Asst. Professor - Dept. of Computer Science & Engineering, Bengal College of Engineering & Technology*, for his generous guidance, help and useful suggestions. I am also grateful to community of Android Developers and persons behind the official guides to Android development hosted by Google. I would also like to thank my friends and family for motivating me to develop this project.

Signature(s) of Students

Debaleen Das Spandan,

University Roll Number: 12500117078

Aditya Kumar,

University Roll Number: 12500117110

Acquib Javed,

University Roll Number: 12500117112

Abstract

Designing software is an integral part of computer science. Various tools are used in making a software. Various design patterns along with several programming languages, development environments, databases etc. are used. One should be well equipped with the skills and concepts of a few among them.

In this project, we have explored one such app architecture – MVVM architecture and used platform specific tools and an Object-Oriented Programming language to implement the same.

Model-View-ViewModel (MVVM) is a software architectural pattern. MVVM facilitates a separation of development of the graphical user interface – be it via a markup language or GUI code - from development of the business logic or back-end logic (the data model).

This project explores the implementation of MVVM architecture in an Android app using Android Architecture Components.

Index

Introduction

Task Management is an integral part of our daily life. We need to perform various tasks day-in-day-out. But with 24 hours at our disposal each day, managing various tasks gets head scratching. Let’s face it, we cannot perform efficiently if we don’t manage our time and tasks well. Task management is as integral to the project management discipline as a ball to the game of football.

We, as students feel the necessity of managing our daily chores efficiently. The key to success is good discipline and discipline comes from following a routine. Now how can we make a good routine if we don’t manage our chores well? So, we need a tool to help us manage our daily tasks efficiently and properly. And what is a better tool than an app which can manage our tasks efficiently in our smartphone that we so dearly use daily? So, we have developed a Task Manager app for android platform.

We have used Model–view–ViewModel (MVVM) design to develop the app. This project explores the architecture of the app in brief.

Project URL: https://github.com/the-it-weirdo/TaskManager.git

Why choose Android?

Android is a mobile operating system based on a modified version of the Linux kernel and other open source software, designed primarily for touchscreen mobile devices such as smartphones and tablets. Android SDKs are free to use and there is a large community support.

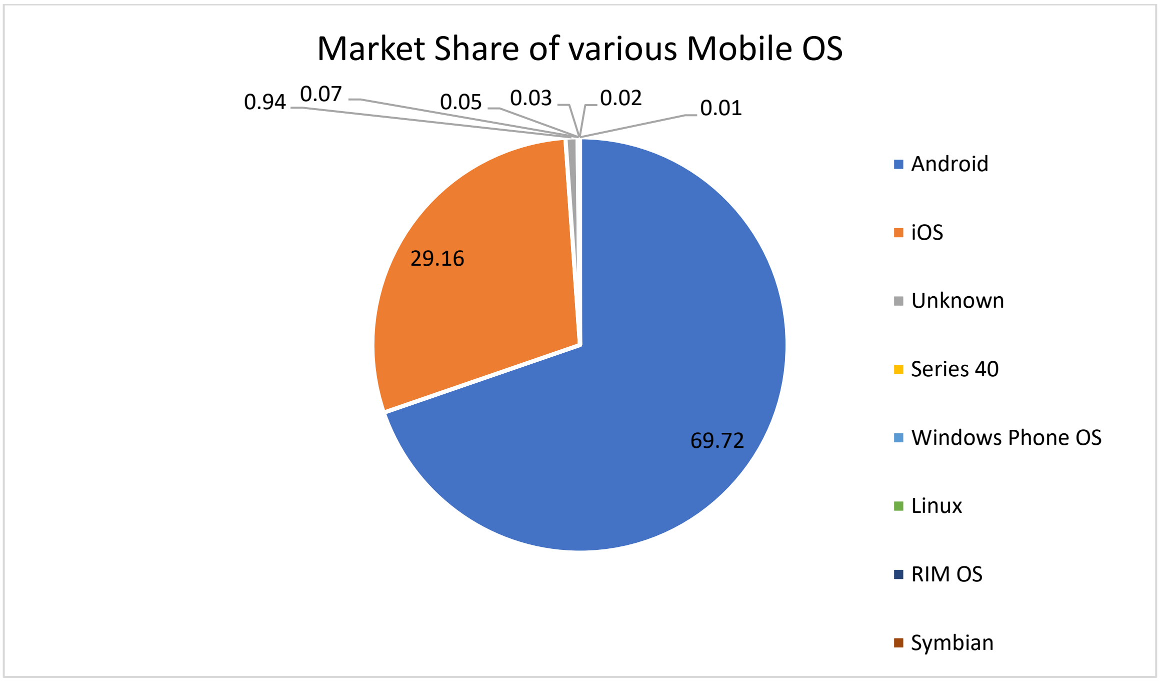


Figure 1

From the above chart it is clearly evident that Android has the highest market share followed by Apple's iOS. So, keeping all the features in mind, we choose Android to be the target platform.

Why a Task Manager App?

Task Management is an integral part of our daily life. Now we know there are various Task Manager app already available in the market. So why another? Well, we can’t say we have any special feature in our app. That is something we have planned for the future. We plan to introduce some useful and unique features in the app to make it market worthy. As of now, we chose this very app as it provides a fair conceptual challenge. Through this app, we can elaborately demonstrate our understanding about concepts of Object-Oriented-Programming in Java, Database queries and Android Architecture Components.

We have used Android Architecture Components like Room, ViewModel and LiveData to implement the MVVM architecture. The database used in this project is SQLite Database. The backend codes are written in Java. User Interfaces are designed using XML with Java support.

Resources required to develop the app

1. Android Studio IDE (We have used version 3.5 of the IDE).
2. An emulated device/a physical device running on android platform for testing.
3. SQLite Database.
4. Android Architecture components and Room Database.
5. Lots of study about Android app development.
6. A little bit brainstorming and
7. Lots and lots of coding (Mostly in Java and some XML).

Resources required to run the app

1. An android device running on Android version 5.0 (Lollipop) or above.
2. 15 MB storage for installing the app.
3. Common Sense.

Android Architecture Components

A long time ago, when Android Developers developed apps with some of the core components like Activity, BroadcastReceiver, Services and Content Provider, there was lots of hassle in making things work out. All those hassles can be avoided by using Android Architecture Components. The purpose of Architecture Components is to provide guidance on app architecture, with libraries for common tasks like lifecycle management and data persistence.

Architecture components helps developers structure their app in a way that is robust, testable, and maintainable with less boilerplate code.

Architecture Components provide a simple, flexible, and practical approach that frees developers from some common problems so that they can focus on building great experiences.

Following is a graphical representation of the recommended usage of Android Architecture Components:

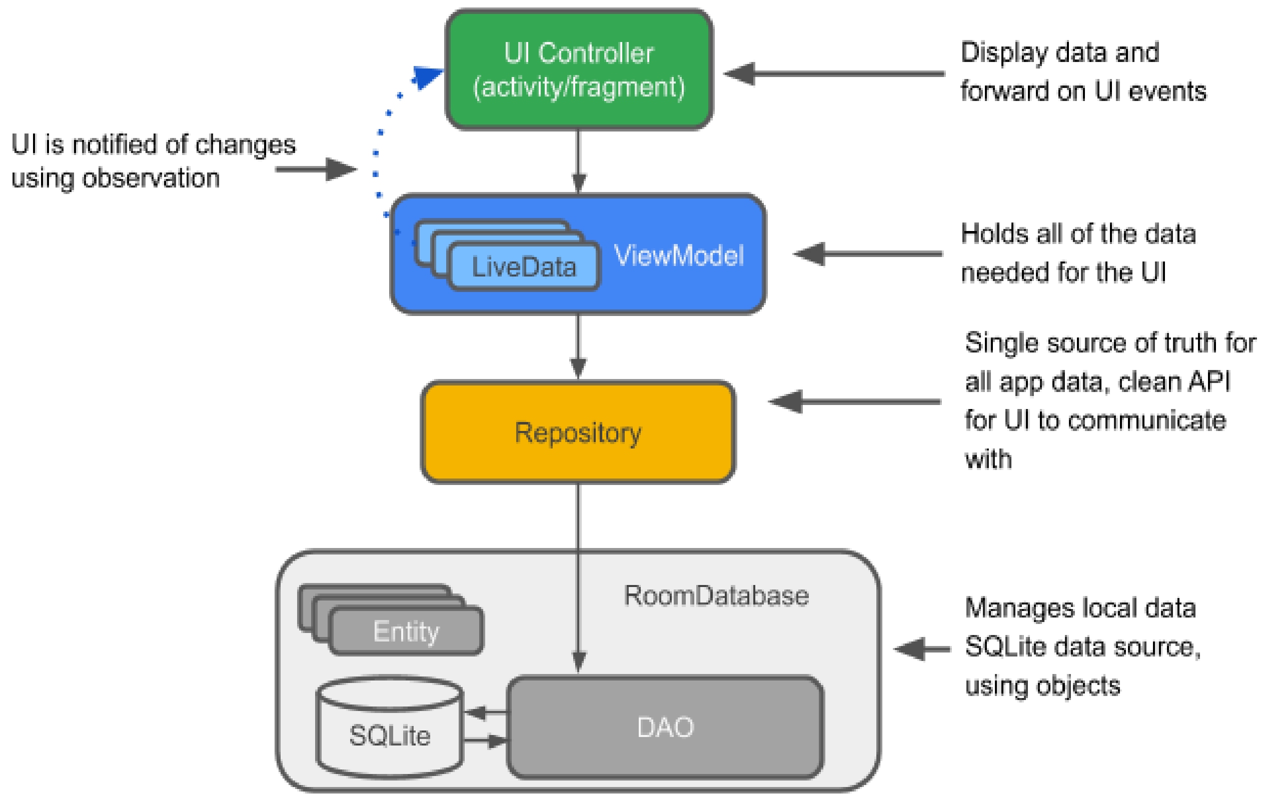

UI Controller (activity/fragment)
Display data and forward on UI events
UI is notified of changes using observation
LiveData
ViewModel
Holds all of the data needed for the UI
Repository
Single source of truth for all app data, clean API for UI to communicate with
RoomDatabase
Entity
SQLite
DAO
Manages local data SQLite data source, using objects

Figure 2: MVVM architecture.

Model-View-ViewModel (MVVM) is a software architectural pattern. MVVM facilitates a separation of development of the graphical user interface – be it via a markup language or GUI code - from development of the business logic or back-end logic (the data model).

Entity: When working with Architecture Components, this is an annotated class that describes a database table. In our app, we have two separate entities:

1. User and
2. Task

SQLite database: On the device, data is stored in an SQLite database. The Room persistence library creates and maintains this database.

DAO: Dao stands for Data access object. It is a mapping of SQL queries to functions. Without using Architecture Components, we would have to use the SQLiteOpenHelper class which raises further complications leading to a longer development time. When we use a DAO, we call the methods, and Room takes care of the rest. In our app, we have separate DAOs for the User entity and the Task entity.

Room database: Database layer on top of SQLite database that takes care of mundane tasks that otherwise would be handled with an SQLiteOpenHelper. It is a database holder that serves as an access point to the underlying SQLite database. The Room database uses the DAO to issue queries to the SQLite database. We have a single database for maintaining both the entities. This way, our database is composed of multiple tables.

Repository: It is a class for managing multiple data sources. A Repository class is actually a bridge between the database and the app. A repository class receives the data from various sources like local database, cloud, web etc. In our app, we have separate repositories for User and Task. That way, we can easily separate the different data received about the entities.

ViewModel: ViewModel is an element or rather a class that provides data to the UI. It acts as a communication center between the Repository and the UI. It hides where the data originates from the UI. ViewModel instances survive configuration changes. ViewModel is a key element in MVVM architecture as the lifecycle of ViewModel is independent of the lifecycle of activity/fragment in an Android app. We have used separate ViewModel classes for our User entity and Task entity. And also, we have strictly followed the google recommended rule of 1 ViewModel per Activity/Fragment. We will be discussing about the importance of ViewModel and its lifecycle with respect to the lifecycle of an activity in a following section.

LiveData: It is a data holder class. It can be observed by an activity/fragment. It always holds/caches latest version of data. It notifies its observers when the data has changed. LiveData is lifecycle aware. UI components just observe relevant data and don't stop or resume observation. Unlike a regular observable, LiveData is lifecycle-aware, meaning it respects the lifecycle of other app components, such as activities, fragments, or services. This awareness ensures LiveData only updates app component observers that are in an active lifecycle state.

ViewModel and its lifecycle

The ViewModel class is designed to store and manage UI-related data in a lifecycle conscious way. The ViewModel class allows data to survive configuration changes such as screen rotations. The Android framework manages the lifecycles of UI controllers, such as activities and fragments. The framework may decide to destroy or re-create a UI controller in response to certain user actions or device events that are completely out of a developer’s control.

If the system destroys or re-creates a UI controller, any transient UI-related data stored in them is lost. For example, in our app an activity includes a list of tasks. If we didn’t use ViewModel, when the activity is re-created for a configuration change, the new activity would have to re-fetch the list of tasks. For simple data, the activity can use the *onSaveInstanceState()* method and restore its data from the bundle in *onCreate()*, but this approach is only suitable for small amounts of data that can be serialized then deserialized, not for potentially large amounts of data like a list of tasks or bitmaps.

Another problem is that UI controllers frequently need to make asynchronous calls (like database queries) that may take some time to return. The UI controller needs to manage these calls and ensure the system cleans them up after it's destroyed to avoid potential memory leaks. This management requires a lot of maintenance, and in the case where the object is re-created for a configuration change (for example rotation of device by user), it's a waste of resources since the object may have to reissue calls it has already made.

UI controllers such as activities and fragments are primarily intended to display UI data, react to user actions, or handle operating system communication, such as permission requests. Requiring UI controllers to also be responsible for loading data from a database or network adds bloat to the class. Assigning excessive responsibility to UI controllers can result in a single class that tries to handle all of an app's work by itself, instead of delegating work to other classes. When a single class tries to handle all of an app's task all by itself, more resources are required and the app consumes more primary memory which is an expensive thing to do in a mobile device. Assigning excessive responsibility to the UI controllers in this way also makes testing a lot harder. It's easier and more efficient to separate out view data ownership from UI controller logic.

ViewModel objects are automatically retained during configuration changes so that data they hold is immediately available to the next activity or fragment instance. If the activity is re-created, it receives the same ViewModel instance that was created by the first activity. When the owner activity is finished, the framework calls the ViewModel object's *onCleared()* method so that it can clean up resources. The ViewModel remains in memory until the Lifecycle it's scoped to goes away permanently: in the case of an activity, when it finishes, while in the case of a fragment, when it's detached.

The following figure illustrates the various lifecycle states of an activity as it undergoes a rotation and then is finished. The illustration also shows the lifetime of the ViewModel next to the associated activity lifecycle. This particular diagram illustrates the states of an activity. The same basic states apply to the lifecycle of a fragment.

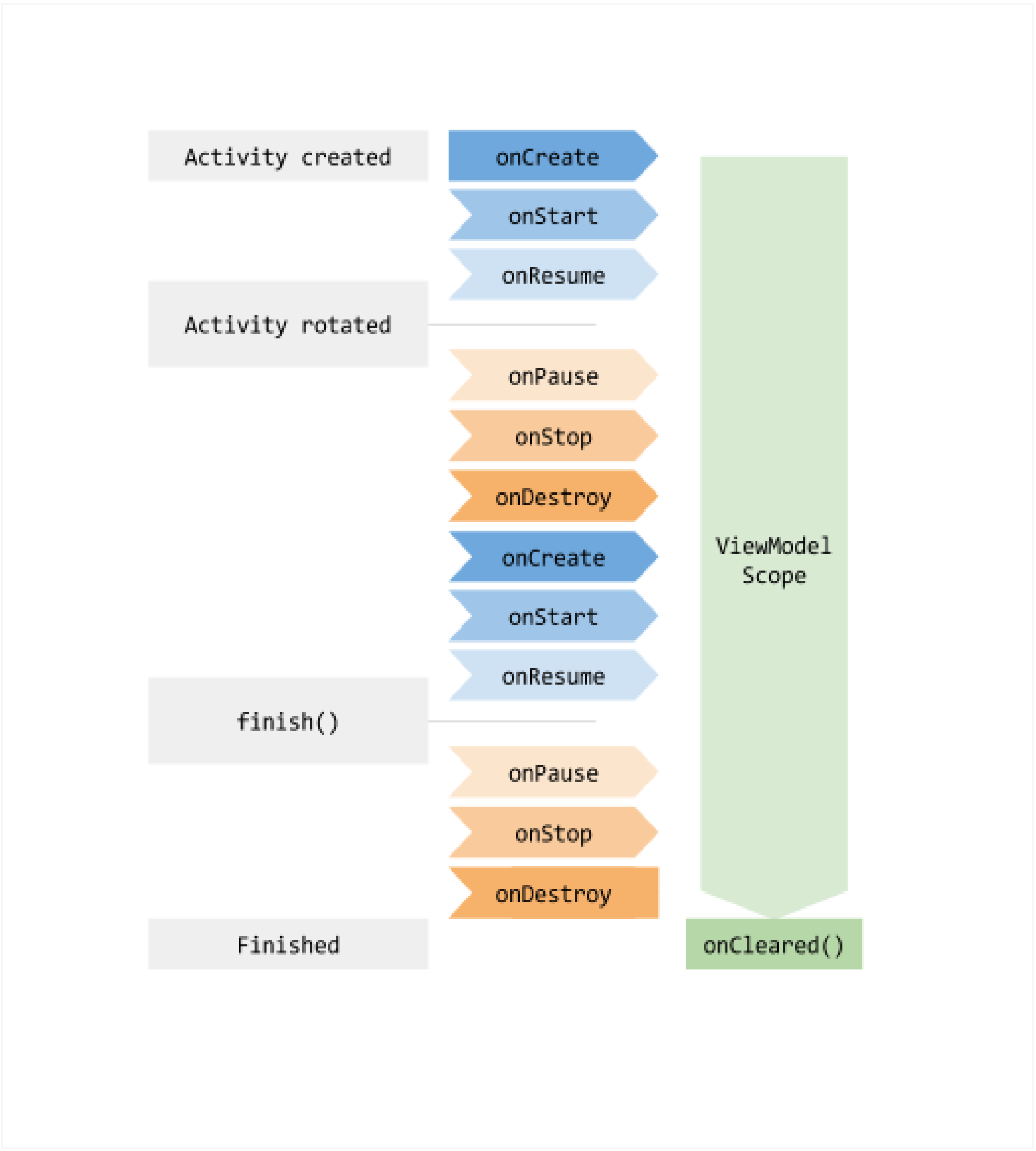


Figure 3: Lifecycle/Scope of a ViewModel with respect to lifecycle of an activity.

Features provided by our Task Manger app

- User based. Multiple users can create account and use the app.
- Tasks can be classified into distinct categories. We have 2 categories for now viz. ‘Home’ and ‘Work’.
- The app shows live time and date in its main page.
- The app greets the user in accordance with the live time of user’s locale.
- The app stores logged in state in its *SharedPreferences* memory. So, the user doesn’t need to login in each time the app restarts.
- We have provided input validation for various fields in the app (for e.g. email, password etc.)

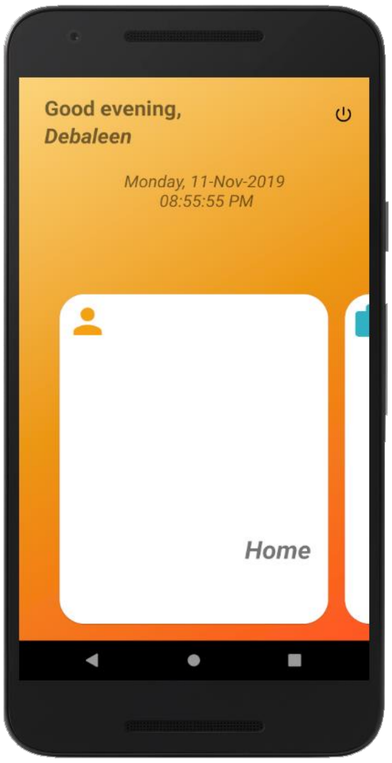
Future updates

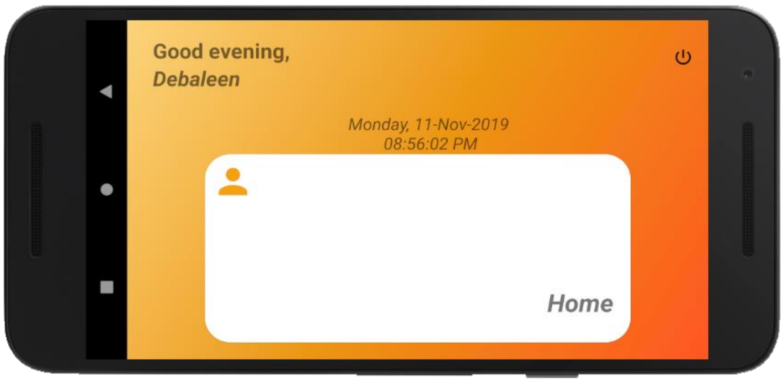
Although our app provides basic functionality and some good features, it is not yet market ready. To make the app market ready, we will need to introduce some unique and useful features. The future updates to this app may include the following features:

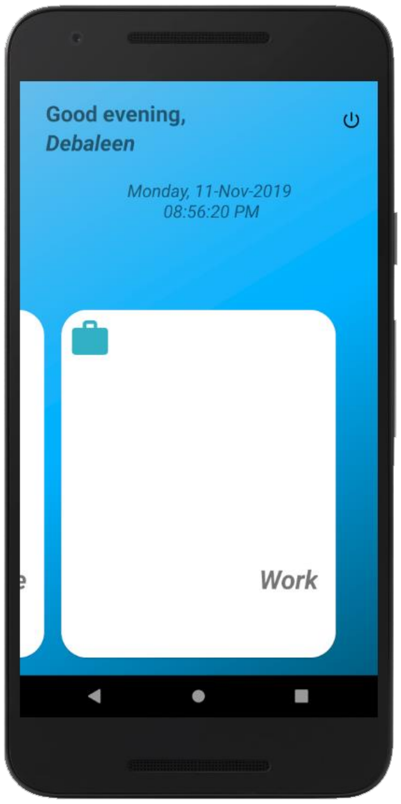
- Custom categories. Users will be able to create their own custom categories.
- Introduction of time-based task management. Users will be able to set a time by which he/she needs to complete the task.
- Incorporation with calendar apps so that users can plan their tasks beforehand.
- Notifications for upcoming tasks. The app will notify the user of any upcoming task that needs to be completed.
- Suggest user an efficient time management plan based on his/her daily activity.

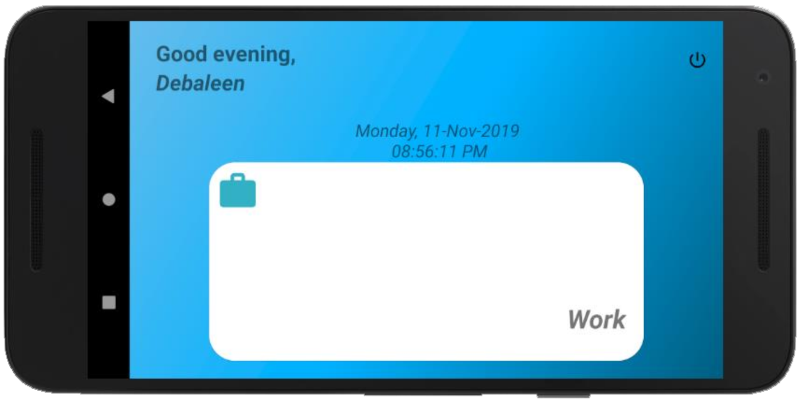
In addition to the features mentioned above, the future updates may also introduce some other unique feature if the developer deems the feature fit and in accordance to the app.

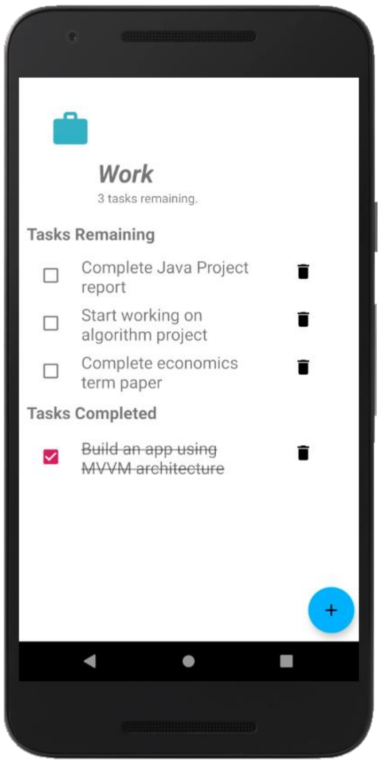
App Screenshots










Work

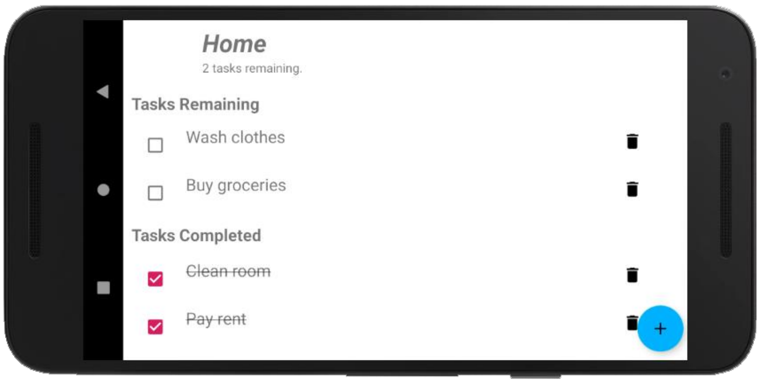
3 tasks remaining.

Tasks Remaining

- [ ] Complete Java Project report
- [ ] Start working on algorithm project
- [ ] Complete economics term paper

Tasks Completed

- [x] ~~Build an app using MVVM architecture~~


Home

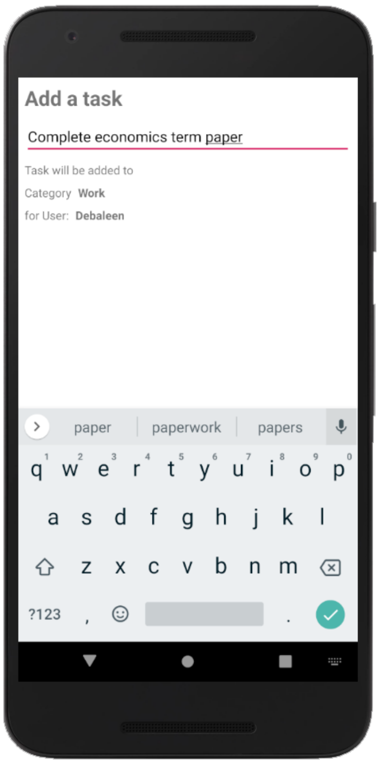
2 tasks remaining.

Tasks Remaining

- [ ] Wash clothes
- [ ] Buy groceries

Tasks Completed

- [x] ~~Clean room~~
- [x] ~~Pay rent~~

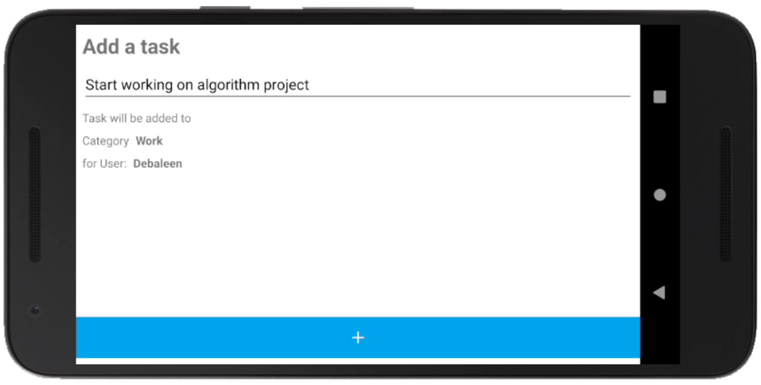

Add a task

Complete economics term paper

Task will be added to

Category **Work**

for User: **Debaleen**

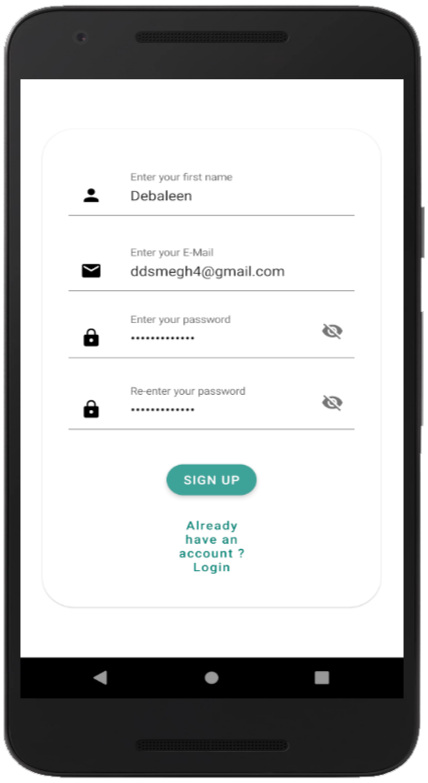

Add a task

Start working on algorithm project

Task will be added to

Category **Work**

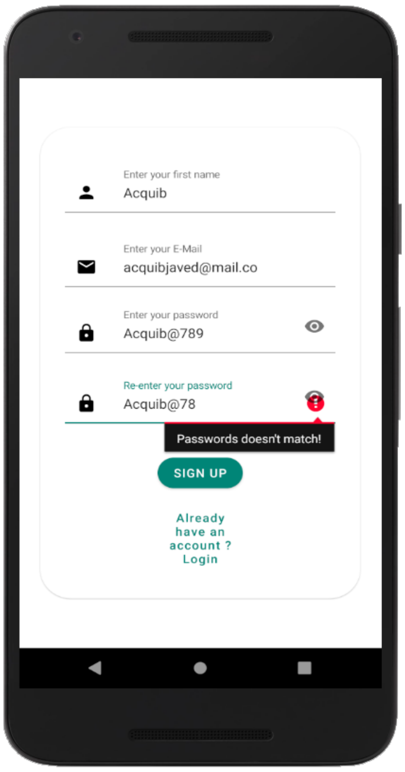
for User: **Debaleen**


Enter your first name
Debaleen

Enter your E-Mail
ddsmegh4@gmail.com

Enter your password
•••••••••••••

Re-enter your password
•••••••••••••

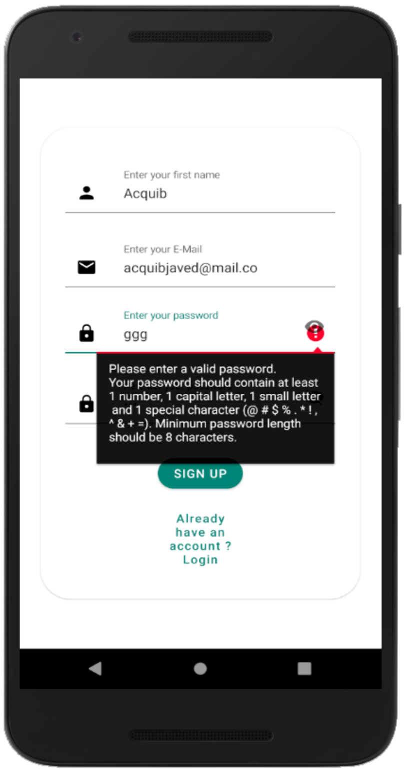

Enter your first name
Acquib

Enter your E-Mail
acquibjaved@mail.co

Enter your password
Acquib@789

Re-enter your password
Acquib@78

Passwords doesn't match!


Enter your first name
Acquib

Enter your E-Mail
acquibjaved@mail.co

Enter your password
ggg

Please enter a valid password.
Your password should contain at least 1 number, 1 capital letter, 1 small letter and 1 special character (@ # $ % . * ! , ^ & + =). Minimum password length should be 8 characters.

References

1. Market share data obtained from https://netmarketshare.com with the following filters:
    a. Operating-System-Market-Share
    b. Device Type: Mobile
    c. Date Interval: Monthly
    d. Date Start: 2018-04
    e. Date End: 2019-03

    The readymade data can be found at https://bit.ly/2rvn80T
2. Android App Development guide: https://developer.android.com/guide
3. Guide to Android Architecture Components: https://developer.android.com/topic/libraries/architecture
4. Using android architecture components example: https://codelabs.developers.google.com/codelabs/android-room-with-a-view/#0