Android Application on MVVM Architecture to Change Database

Nagaraj M[1], Chethana G[2]

[1,2]*Department of Electronics and Communication, RV College of Engineering, Bangalore.*

Abstract- The main objective of this app is to provide premium account and app upgradation facility and to get the details of the users of that app enable and monthly reports using MVVM architecture. Personalised performance reports for all the users of the batch and Chat facility, anytime, anywhere. Disable chat when you're busy, easily create their Control what info your service provider and admin can view for your batches online courses and sell on their mobile app. some other facilities like data is secured using AES 256bit encryption of courses providing maximum security, manage your application completely digitally. simplify the above work managing app is made which upgrades the app to new version and activation of premium account is done through this app.

***Key Words**:* MVVM, KOIN, API

1. INTRODUCTION

Android is a powerful operating system and it supports a large number of applications in Smartphones. These applications are more comfortable and advanced for users. The hardware that supports android software is based on the ARM architecture platform. The android is an open-source operating system that means that it's free and anyone can use it. The android has got millions of apps available that can help you manage your life one or another way and it is available to low cost in the market for that reason android is very popular.

API or Application Programming Interface is essentially how our app communicates with the Android operating system. API is what allows us to build our apps so they can run on android devices. Now as with everything over time new features are developed and old features refined and sometimes products are completely revamped.

Maintaining the larger database is difficult so api's are used to change the database and update the new things and allow other changes to the database that can done easily using MVVM architecture and some dependency injection so that UI can be changed frequently without changing the base code

2. PROBLEM STATEMENT

develop the android Class App using MVVM Architecture and KOIN dependency to avoid lagging and replace the Dagger 2 with the same so that We can develop a user-friendly App

Tools App helps the non-technical employees to get information about students, organization, and tutorsOBJECTIVES

The objective of this project is to create a user-friendly app that provides useful information about users such as student, tutor and organizations.

A literature survey was done on works on apps created using mvvm architecture. Some of them are listed below.

In [1], The study results aim to bring more clarity in the variety of MVVM design patterns and help practitioners to make better grounded decisions when selecting patterns.

In [2], This research provided some recommended solutions to satisfy the relations between MVVM objects in the project. Activity designed in [1], with MVVM design pattern for Android development is introduced and MVVM Framework, principle and advantage are also studied in detail.

In [3], Shifting from dagger2 to KOIN dependency injection, pragmatic lightweight dependency injection framework for Kotlin developers. Written in pure Kotlin using functional resolution only: no proxy, no code generation, no reflection.

In [4], Android AlertDialog can be used to display the dialog message with OK and Cancel buttons. It can be used to interrupt and ask the user about his/her choice to continue or discontinue.

In [5], Google otp 2-Step Verification provides stronger security for our Google Account by requiring a second verification step when you sign in. In addition to our password, we'll also need a code generated by the Google Authentication app on our phone. Platform and they can provide ratings based on the quality.

3. MVVM ARCHITECTURE SETUP

MVVM was designed to make use of data binding functions in WPF (Windows Presentation Foundation) to better facilitate the separation of view layer development from the rest of the pattern, by removing virtually all GUI code ("code-behind") from the view layer. Instead of requiring user experience (UX) developers to write GUI code, they can use the framework markup language (e.g.,

XML) and create data bindings to the view model, which is written and maintained by application developers. The separation of roles allows interactive designers to focus on UX needs rather than programming of business logic. The layers of an application can thus be developed in multiple work streams for higher productivity. Even when a single developer works on the entire code base, a proper separation of the view from the model is more productive, as user interface typically changes frequently and late in the development cycle based on end-user feedback. The MVVM pattern attempts to gain both advantages of separation of functional development provided by MVC, while leveraging the advantages of data bindings and the framework by binding data as close to the pure application model as possible. It uses the binder, view model, and any business layers' data-checking features to validate incoming data. The result is that the model and framework drive as much of the operations as possible, eliminating or minimizing application logic which directly manipulates the view.

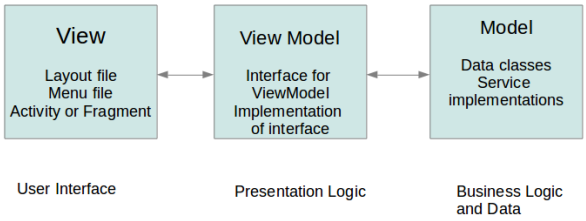

Fig. 3.1 Crop Database 1

Model : Model refers either to a domain model, which represents real state content (an object oriented approach), or to the data access layer, which represents content (a data-centric approach).View, As in the model-view-controller (MVC) and model-view-presenter (MVP) patterns, the view is the structure, layout, and appearance of what a user sees on the screen. It displays a representation of the model and receives the user's interaction with the view (clicks, keyboard, gestures, etc.), and it forwards the handling of these to the view model via the data binding (properties, event callbacks, etc.) that is defined to link

View Model : The view model is an abstraction of the view exposing public properties and commands. Instead of the controller of the MVC pattern, or the presenter of the MVP pattern, MVVM has a binder, which automates communication between the view and its bound properties in the view model. The view model has been described as a state of the data in the model. The main difference between the view model and the Presenter in the MVP pattern, is that the presenter has a reference to a view whereas the view model does not. Instead, a view directly binds to properties on the view model to send and receive updates. To function efficiently, this requires a binding technology or generating boilerplate code to do the binding the view and view model.

Data : Declarative data and command-binding are implicit in the MVVM pattern. In the Microsoft solution stack, the binder is a markup language called XML. The binder frees the developer from being obliged to write boiler-plate logic to synchronize the view model and view. When implemented outside of the Microsoft stack, the presence of a declarative data binding technology is what makes this pattern possible, and without a binder, one would typically use MVP or MVC instead and have to write more boilerplate (or generate it with some other tool).

4. API SETUP AND DATA FETCHING

Web APIs are the defined interfaces through which interactions happen between an enterprise and applications that use its assets, which also is a Service Level Agreement (SLA) to specify the functional provider and expose the service path or URL for its API users. An API approach is an architectural approach that revolves around providing a program interface to a set of services to different applications serving different types of consumers. When used in the context of web development, an API is typically defined as a set of specifications, such as Hypertext Transfer Protocol (HTTP) request messages, along with a definition of the structure of response messages, usually in an Extensible Markup Language (XML) or JavaScript Object Notation (JSON) format. An example might be a shipping company.

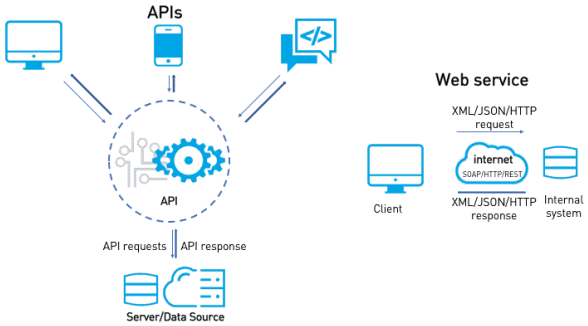

Fig.4.1: Api architecture to fetch the data

API that can be added to an eCommerce-focused website to facilitate ordering shipping services and automatically include current shipping rates, without the site developer having to enter the shipper's rate table into a web database. While "web API" historically has been virtually synonymous with web service, the recent trend (so-called Web 2.0) has been moving away from Simple Object Access Protocol (SOAP) based web services and service-oriented architecture (SOA) towards more direct representational state transfer (REST) style web resources and resource-oriented architecture (ROA). Part of this trend is related to the Semantic Web movement toward Resource Description Framework (RDF), a concept to promote web-based ontology engineering technologies. Web API's allow the combination *of multiple API's into new applications known as mashups. In the social media space, web API's have allowed web communities to facilitate sharing content and data between communities and applications. In this way, content that is created*

in one place dynamically can be posted and updated to multiple locations on the web. For example, Twitter's REST API allows developers to access core Twitter data and the Search API provides methods for developers to interact with Twitter Search and trends data

A. ui design

App bar In its most basic form, the action bar displays the title for the activity on one side and an overflow menu on the other. Even in this simple form, the app bar provides useful information to the users, and helps to give Android apps a consistent look and feel. As Shown in fig 4.2 Beginning with Android 3.0 (API level 11), all activities that use the default theme have an ActionBar as an app bar. However, app bar features have gradually been added to the native ActionBar over various Android releases. As a result, the native ActionBar behaves differently depending on what version of the Android system a device may be using. By contrast, the most recent features are added to the support library's version of Toolbar, and they are available on any device that can use the support library. For this reason, you should use the support library's Toolbar class to implement your activities' app bars. Using the support library's toolbar helps ensure that your app will In the admin form, once login credentials are entered, there will be a page in which he can enter a new crop and its parameter values or edit an already entered crop in the database. This data from the webpage is to be stored in the database. So, this will be processed through PHP and JavaScript.

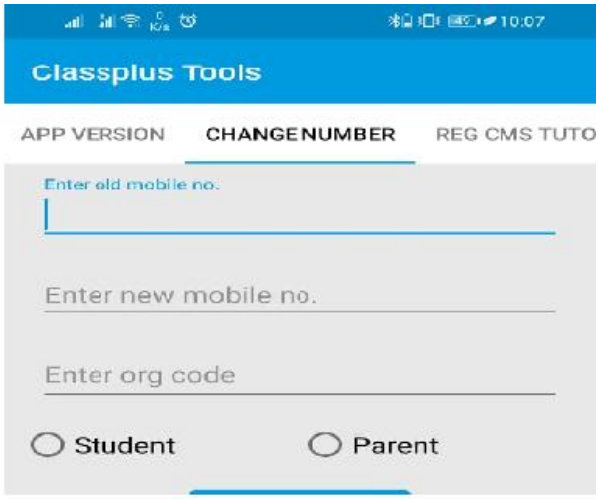

Fig. 4.2: Toolbar setup

have consistent behaviour across the widest range of devices. For example, the Toolbar widget provides a material design experience on devices running Android 2.1 (API level 7) or later, but the native action bar doesn't support material design unless the device is running Android 5.0 (API level 21) or later. Once you set the toolbar as an activity's app bar, you have access to the various utility methods provided by the v7 appcompat support library's ActionBar class. This approach lets you do a number of useful things, like hide and show the app bar. To use the ActionBar utility methods, call the activity's getSupportActionBar() method. This method returns a reference to an appcompat ActionBar object. Once you have that reference, you can call any of the ActionBar methods to adjust the app bar. For example, to hide the app bar, call ActionBar.hide().

Dialog box

A dialog is a small window that prompts the user to make a decision or enter additional information. A dialog does not fill the screen and is normally used for modal events that require users to take an action before they can proceed. The Dialog class is the base class for dialogs, but you should avoid instantiating Dialog directly. Instead, use one of the following sub classes: AlertDialog: A dialog that can show a title, up to three buttons, a list of selectable items, or a custom layout

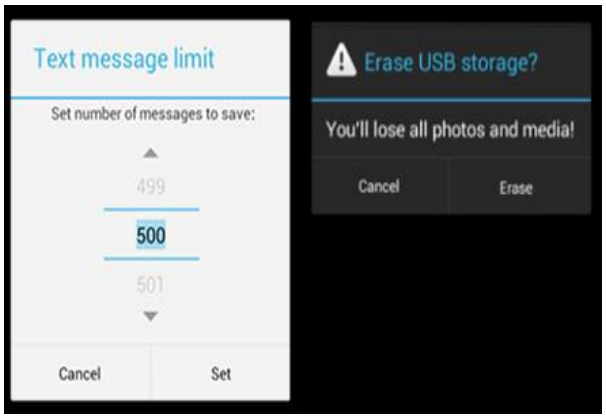

Fig.4.3: dialog box for confirmation

DatePickerDialog or TimePickerDialog: A dialog with a pre-defined UI that allows the user to select a date or time. Fig 4.3 define the style and structure for your dialog, but you should use a Dialog Fragment as a container for your dialog. The Dialog Fragment class provides all the controls you need to create your dialog and manage its appearance, instead of calling methods on the Dialog object. Using Dialog Fragment to manage the dialog ensures that it correctly handles lifecycle events such as when the user presses the Back button or rotates the screen. The Dialog Fragment class also allows you to reuse the dialog's UI as an embedded component in a larger UI, just like a traditional Fragment (such as when you want the dialog UI to appear differently on large and small screens).

5. RESULTS AND DISCUSSION

The figure 5.1 shows the Home page having options for a login. The user will have to enter the gmail to access the app features, thus anybody who has an access to the app can obtain the information of users they are looking for. The Admin will have to login through Gmail because he will be having access to the main database to which he can add a new user or update an existing one. Thus, security is provided to the database as well as made user-friendly to the customers.

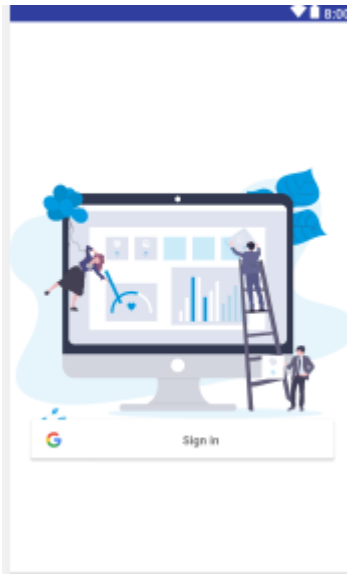

Fig. 7.1: Home page

The Figure 7.2 shows the Mobile number Entry after which he will be directed to user information in viewpager shown in figure 7.2 itself. User can get details of the person as shown in figure temperature would be displayed.

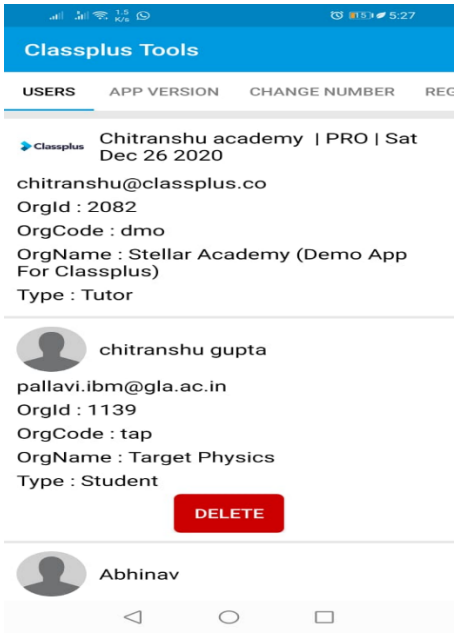

Fig. 7.2: getting user details

Here, organization code is entered as Arecanut in figure 7.3 which will fetch all the parameters related to that organisation. The page will appear as shown in figure 7.3

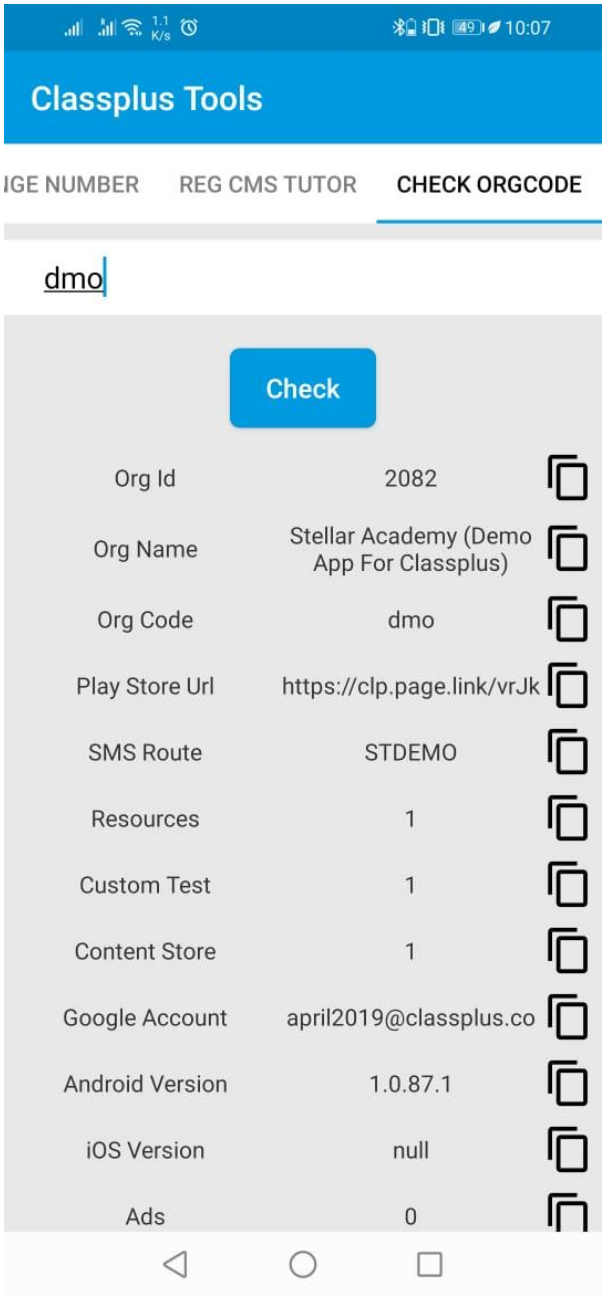

Fig. 7.3: org info output

To add new number old number and org details has to be added and future updates will come to new number., as shown in [8] will help the admin to manage number changes.

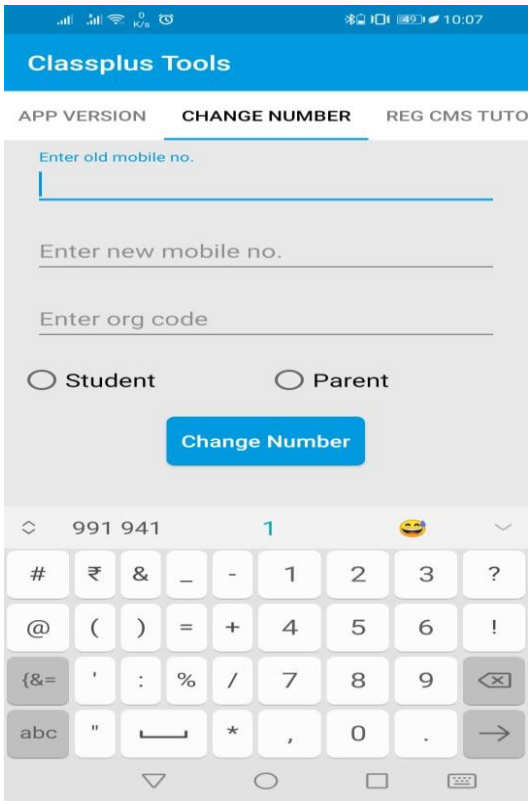

Fig. 7.5: change mobile number

6. CONCLUSION

MVVM pattern is a guidance architecture pattern for designing and developing complex applications to achieve better reusability, maintainability and extensibility with low degree of coupling. We should avoid using MVVM pattern for simple applications.

KOIN dependency injection helps us to increase our productivity with Kotlin. Rx-java helps us by handling the cache without creating caching classes, combining the reception of requests and results processing and getting rid of standard AsyncTask, decreasing memory leak by 90%,optimizing the code to increase an application response, making methods easier to combine premium account and app upgradation provides additional features which improves product efficiency. other details related tabs provide easy access to user solutions.

One of the ways to optimize the design is by doing with MVVM architecture by dividing the complete design into separate parts. Multiple times UI can be changed without disturbing the main code. Migration can be done from mvvm to others when there are new updates available in the market. New features can be added without changing the present version of the application.

7. ACKNOWLEDGEMENT

The authors thank Ms. Chethana G for providing excellent guidance in carrying out the work. We also thank Ms. Netravati, professor of RV College of Engineering for providing their valuable feedback.

8. REFERENCES

[1] X. Li, D. Chang, H. Pen, X. Zhang, Y. Liu, and Y. Yao, "Application of mvvm design pattern in mes," in 2015 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER), 2015, pp. 1374–1378.

[2] A. Syromiatnikov and D. Weyns, "A journey through the land of model-view-design patterns," in 2014 IEEE/IFIP Conference on Software Architecture, 2014, pp. 21– 30.

[3] K. Jeˇzek, L. Holy´, and P. Brada, "Dependency injection refined by extra-functional properties," in 2012 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), 2012, pp. 255–256.

[4] T. Limpiti, S. Khamyan, S. Chumsaeng, and N. Puttarak, "A smart dialogue box for the hearing impaired," in 2019 5th International Conference on Engineering, Applied Sciences and Technology (ICEAST), 2019, pp. 1–4.

[5] M. H. Eldefrawy, K. Alghathbar, and M. K. Khan, "Otp-based two-factor authentication using mobile phones," in 2011 Eighth International Conference on Information Technology: New Generations, 2011, pp. 327–331.

[6] B. S. Pochampally and J. Liu, "A secure tracking mobile app development," in 2017 12th IEEE Conference on Industrial Electronics and Applications (ICIEA), IEEE, Jun. 2017. doi: 10.1109/iciea.2017.8282926

[7] Prathibha S R, Anupama Hongal, Jyothi M P "*IOT Based Monitoring System in Smart Agriculture*" 2017 International Conference on Recent Advances in Electronics and Communication Technology (ICRAECT). 16-17 March 2017, 81-84. 10.1109/ICRAECT.2017.52..

[8] M. F. A. Samsudin, R. Mohamad, S. I. Suliman, N. M. Anas and H. Mohamad, "Implementation of wireless temperature and humidity monitoring on an embedded device," *2018 IEEE Symposium on Computer Applications & Industrial Electronics (ISCAIE)*, Penang, 2018, pp. 90-95, doi: 10.1109/ISCAIE.2018.8405450.

BIOGRAPHY

Nagaraj M,
BE final year,
Dept. of Electronics and Communication,
RV College of Engineering.

Ms. Chethana G
Assistant Professor,
Dept. of Electronics and Communication,
RV College of Engineering.