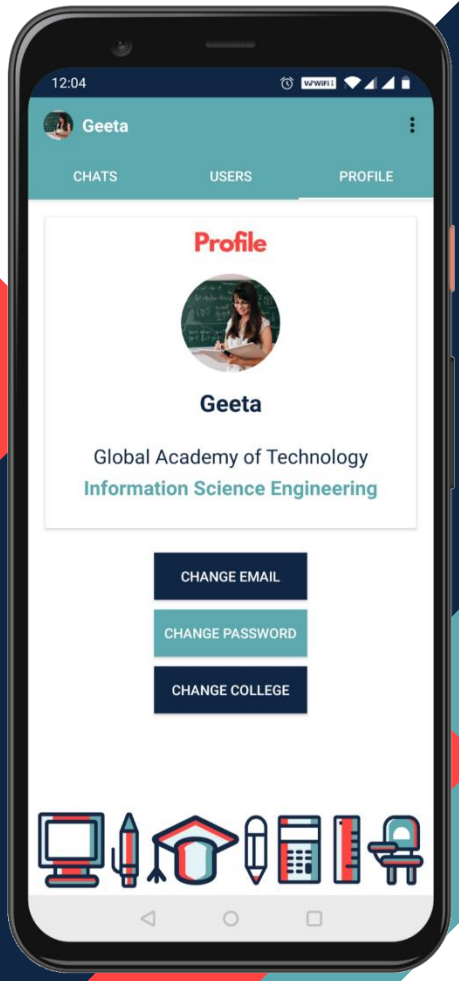
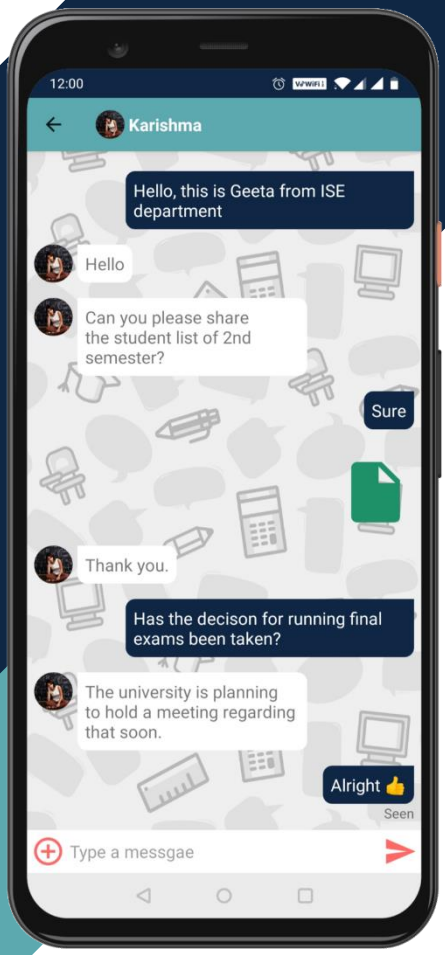
MyStaffRoom

MOBILE APPLICATION





PROJECT REPORT

By

KARISHMA HEGDE

ABSTRACT

Mobile application development is the process of creating software applications that run on a mobile device, and a typical mobile application utilizes a network connection to work with remote computing resources. Hence, the mobile development process involves creating installable software bundles (code, binaries, assets, etc.), implementing backend services such as data access with an API, and testing the application on target devices.

Messaging applications which are also known as chat applications are platforms that enable instant messaging. Many such apps have developed into broad platforms enabling additional functionalities such as status update, group messaging, and conversational commerce (e-commerce via chat).

The project has been designed to facilitate institutional faculty to communicate with colleagues. The android application makes easy for professors or teachers of a particular educational organization to contact other faculties. The application is dedicated to help them share text messages, image files and other documents through the chat interface.

TABLE OF CONTENTS

LIST OF FIGURES

INTRODUCTION

1.1 INTRODUCTION TO MOBILE APPLICATION DEVELOPMENT

Mobile application development is the process of creating software applications that run on a mobile device, and a typical mobile application utilizes a network connection to work with remote computing resources. Hence, the mobile development process involves creating installable software bundles (code, binaries, assets, etc.), implementing backend services such as data access with an API, and testing the application on target devices.

There are two dominant platforms in the modern smartphone market. One is the iOS platform from Apple Inc. The iOS platform is the operating system that powers Apple's popular line of iPhone smartphones. The second is Android from Google. The Android operating system is used not only by Google devices but also by many other OEMs to build their own smartphones and other smart devices.

There are four major development approaches when building mobile applications:

- **Native Mobile Applications** – Installed directly onto the smartphone and can work, in most cases, with no internet connectivity depending on the nature of the application.
- **Cross-Platform Native Mobile Applications –** Cross-platform application developers create a unified API running on top of a native SDK, make use of native IDEs, and build iOS and Android applications that share the same codebase.
- **Hybrid Mobile Applications –** Hybrid apps are part native applications, part web applications. Often, organizations build hybrid apps as wrappers for an existing web page; in that way, they hope to get a presence in the application store, without spending significant effort for developing a different application.
- **Progressive Web Applications –** A progressive web application is a type of application software delivered through the web, built using common web technologies including HTML, CSS and JavaScript. It is intended to work on any platform that uses a standards-compliant browser.

The Mobile Application Development Lifecycle –

There are two interlinked core components of a mobile application:

1) the mobile application “Front-End” that resides on the mobile device, and
2) the services “Back-End” that supports the mobile front-end.

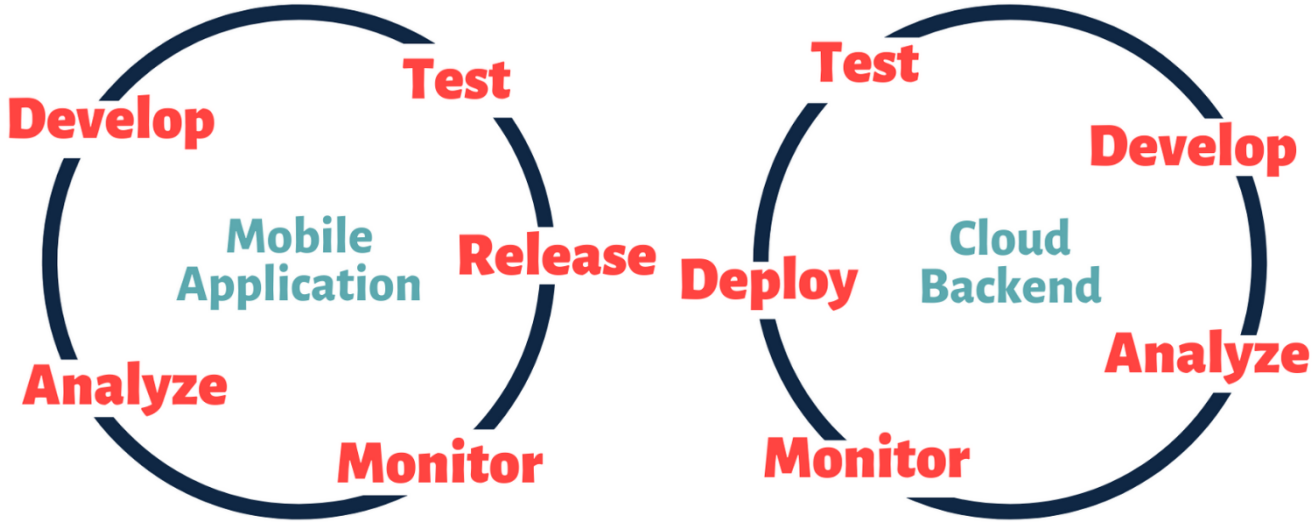


Figure 1: Mobile Application Development Lifecycle

The mobile front-end applications increasingly rely on and integrated with back-end services which provide data to be consumed through the mobile front-end. Such data can include, for example, product information for e-commerce apps or flight info for travel and reservation apps. For a mobile game, the data may include new levels or challenges and scores or avatars from other players. The mobile front-end obtains the data from the back-end via a variety of service calls such as APIs. In some cases, these APIs may be owned and operated by the same entity developing the mobile application. In other cases, the API may be controlled by a third party and access is granted to the mobile application via a commercial arrangement.

1.2 INTRODUCTION TO ANDROID STUDIO

Android Studio is the official integrated development environment (IDE) for Google's Android operating system, built on JetBrains' IntelliJ IDEA software and designed specifically for Android development. Android Studio was announced on May 16, 2013 at the Google I/O conference. In 2019, Kotlin replaced Java as Google's preferred language for Android application development. Java is still supported, as is C++.

The following features are provided in the current stable version:

- Gradle-based build support
- Android-specific refactoring and quick fixes
- Lint tools to catch performance, usability, version compatibility and other problems
- ProGuard integration and app-signing capabilities
- Template-based wizards to create common Android designs and components

- A rich layout editor that allows users to drag-and-drop UI components, option to preview layouts on multiple screen configurations.
- Support for building Android Wear apps
- Built-in support for Google Cloud Platform, enabling integration with Firebase Cloud Messaging (Earlier 'Google Cloud Messaging') and Google App Engine.
- Android Virtual Device (Emulator) to run and debug apps in the Android studio.

To support application development within the Android operating system, Android Studio uses a Gradle-based build system, emulator, code templates, and GitHub integration. Every project in Android Studio has one or more modalities with source code and resource files. These modalities include Android application modules, Library modules, and Google App Engine modules. Android Studio uses an Instant Push feature to push code and resource changes to a running application. A code editor assists the developer with writing code and offering code completion, refraction, and analysis. Applications built in Android Studio are then compiled into the APK format for submission to the Google Play Store.

Figure 2: Android Studio Logo

Android Studio Gradle:

The Android build system compiles application resources and source code, and packages them into APKs that we can test, deploy, sign, and distribute. Android Studio uses Gradle, an advanced build toolkit, to automate and manage the build process, while allowing the developer to define flexible custom build configurations. Each build configuration can define its own set of code and resources, while reusing the parts common to all versions of an application. The Android plugin for Gradle works with the build toolkit to provide processes and configurable settings that are specific to building and testing Android applications.

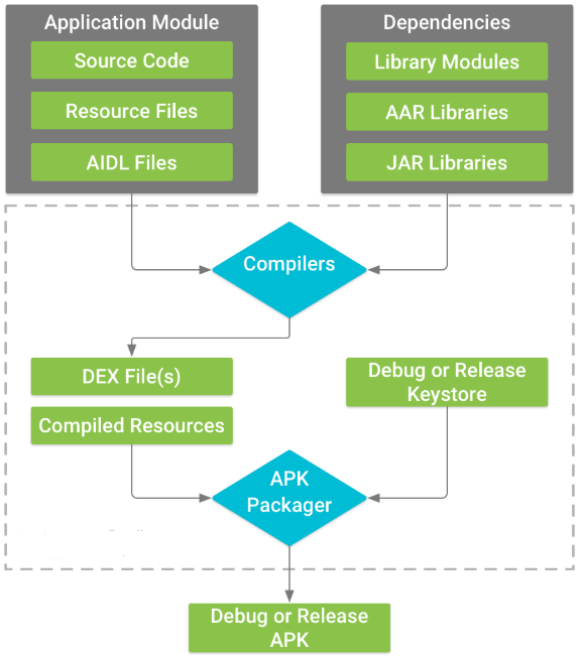


Figure 3: Build Process of Android Application Module

Gradle and the Android plugin run independent of Android Studio. This means that we can build your Android

apps from within Android Studio, the command line on your machine, or on machines where Android Studio is not installed (such as continuous integration servers). The build process involves many tools and processes that convert the project into an Android Application Package (APK).

Android Application Development Workflow:

In order to efficiently build a well-designed application for Android, we require some specialized tools. The following list provides an overview of the process to build an Android app and includes links to some Android Studio tools you should use during each phase of development. he following list provides an overview of the process:

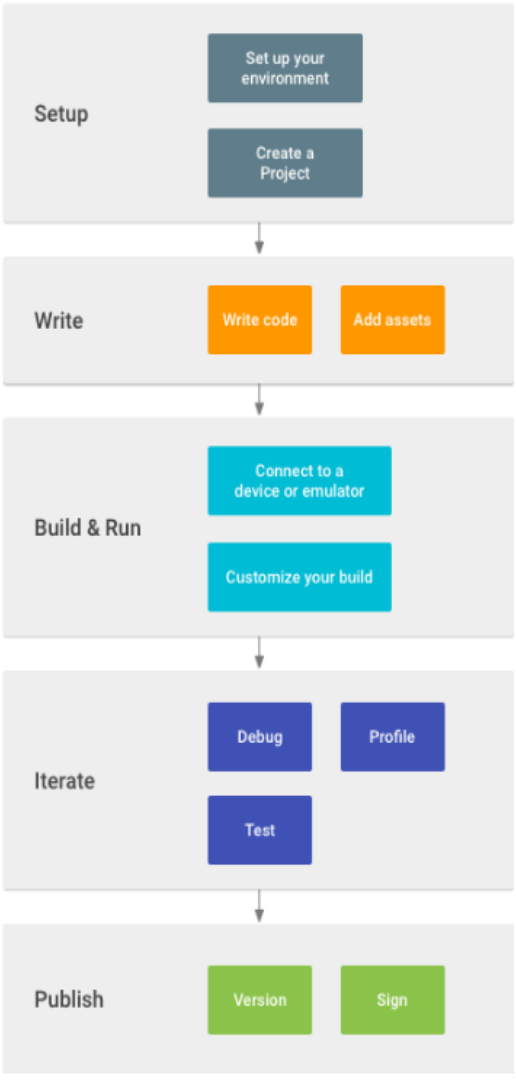


Figure 4: Workflow for development

1. **Set up the workspace**

Install Android Studio and create a project.

2. **Code the application**

Android Studio includes a variety of tools and intelligence to help you work faster, write quality code, design a UI, and create resources for different device types.

3. **Build and run**

During this phase, the project is built into a debuggable APK package that can be installed and run on the emulator or an Android-powered device. The build can be customizable such as creating build variants that produce different types of APKs from the same project.

4. **Debug, profile, and test**

This is the iterative phase in which we continue writing code for the application but with a focus on eliminating bugs and optimizing app performance. Of course, creating tests will help in those endeavours.

5. **Publish**

Versioning of the ready application and signing it with a key are one of the important steps in the final stage.

1.3 INTRODUCTION TO GOOGLE FIREBASE

Firebase is a Backend-as-a-Service (Baas). It provides developers with a variety of tools and services to help them develop quality apps, grow their user base, and earn profit. It is built on Google's infrastructure. As of March 2020, the Firebase platform has 19 products, which are used by more than 1.5 million apps. Firebase frees developers to focus crafting fantastic user experiences. Mobile application developers do not need to manage servers or write APIs. Firebase acts as the server, API and datastore, all written so generically that it can be modified to suit most needs.

Figure 5: Google Firebase

Realtime Database:

Firebase provides a real-time database and back-end as a service. The service provides application developers an API that allows application data to be synchronized across clients and stored on Firebase's cloud. The database is also accessible through a REST API and bindings for several JavaScript frameworks such as AngularJS, React, Ember.js and Backbone.js. The REST API uses the Server-Sent Events protocol, which is an API for creating HTTP connections for receiving push notifications from a server. Developers using the real-time database can secure their data by using the company's server-side-enforced security rules. The mobile application is connected to Firebase through a WebSocket. WebSockets are faster than HTTP. A single WebSocket call is required to connect to the entire database. All synchronizes automatically through that single WebSocket as fast as the client's network can carry it. Firebase sends new data as soon as it's updated. When the client saves a change to the data, all connected clients receive the updated data almost instantly.

Firebase Storage:

Firebase Storage provides secure file uploads and downloads for Firebase apps, regardless of network quality, to be used for storing images, audio, video, or other user-generated content. It is backed by Google Cloud Storage. Firebase Storage has its own system of security rules to protect your GCloud bucket from the masses, while granting detailed write privileges to your authenticated clients. Firebase SDKs for Cloud Storage perform uploads and downloads regardless of network quality. Uploads and downloads are robust, meaning they restart where they stopped, saving the user's time and bandwidth. Firebase SDKs for Cloud Storage integrate with Firebase Authentication to provide simple and intuitive authentication for developers. Developers can use declarative security model to allow access based on

filename, size, content type, and other metadata. Cloud Storage for Firebase is built for exabyte scale when applications have a large user base.

Firebase Authentication:

Firebase Authentication is a service that can authenticate users using only client-side code. It supports social login providers like Facebook, Google and email. Firebase auth has a built-in email/password authentication system. It also supports OAuth2 for Google, Facebook, Twitter and GitHub. Authenticate users with their email addresses and passwords. The Firebase Authentication SDK provides methods to create and manage users that use their email addresses and passwords to sign in. Firebase Authentication also handles sending password reset emails.

Firebase Cloud Messaging:

Firebase Cloud Messaging (FCM) is a free, cross-platform messaging solution that lets you send push notifications to your audience, without having to worry about the server code. By using FCM alongside Firebase's Notifications Composer we can create notifications that target very specific sections of the user base, often without having to write any special code. Using FCM, we can notify a client application if any data is available to sync. Users can be sent notification messages to drive user re-engagement and retention. For use cases such as instant messaging, a message can transfer a payload of up to 4KB to a client application.

REQUIREMENT SPECIFICATION

2.1 SOFTWARE REQUIREMENTS

- Operating System : GNOME or KDE desktop Tested on gLinux based on Debian.
- IDE : Android Studio (Version 3.6 or above)
- Backend tool : Google Firebase account

2.2 HARDWARE REQUIREMENTS

- Processor : Intel® Core™ i3 or higher
- RAM : 8 GB or higher
- Hard Disk : 20GB for Linux (Ubuntu) + 2 GB of available disk space minimum, 4 GB Recommended (500 MB for IDE + 1.5 GB for Android SDK and emulator system image)
- Resolution : 1280 x 800 minimum screen resolution

2.3 MISCELLANEOUS REQUIREMENTS

- Stable internet connection during for gradle build and Firebase.
- Mobile device for live testing of application.

2.4 FUNCTIONAL REQUIREMENTS

In software engineering and systems engineering, a functional requirement defines a function of a system or its component, where a function is described as a specification of behaviour between outputs and inputs. The functional requirement of the messaging application in the project are:

- **User Registration and Login**

  Registration and user identification are done with the help of a valid email id and password credential.
- **Access to contacts**

  Once the user has logged in successfully, they should be able to access the appropriate set of contacts in the user list.

- **Message Status**

  User must be informed whether the message has been sent successfully. It should also tell the user if it has been read by the recipient or not.
- **Media Sharing**

  Users must be able to send media files such as images and documents of different formats. They must be able to download the media messages.
- **Profile Update**

  Users of the application must be able to change the credentials of their profile such as display picture, email or password.

2.5 NON-FUNCTIONAL REQUIREMENTS

A non-functional requirement is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviours. Non-functional requirements are often called "quality attributes" of a system. It includes execution qualities such as safety, security and usability, which are observable during operation (at run time). Evolution qualities include testability, maintainability, extensibility and scalability, which are embodied in the static structure of the system. Some of the non-functional requirements of a messaging application are:

- **Scalability**

  The application should be able to provide good quality instant messaging services to a large user base at all times.
- **Privacy**

  Messages shared between users should be encrypted to maintain privacy.
- **Robustness**

  In case user's device crashes, a backup of their chat history must be stored on remote database servers to enable recoverability.
- **Performance**

  Application must be lightweight and must send messages instantly.

SYSTEM DEFINITION

3.1 PROJECT DESCRIPTION

The project focuses on developing an Android application that allows faculties of educational organizations like schools and colleges to communicate with each other via the messaging application. Since there are no dedicated messaging application solutions for faculties to communicate, this application assists professors and teachers by connecting them with all other faculty belonging to the institution they are associated with.

Front-end User Interface (Mobile Application):

- The user must first register in the registration form that is provided in the application. Basic credentials such as name, institution name and department to which they work under must be provided.
- For authentication purpose, user must provide a valid email id and set a password of sufficient length.
- Once the user registers successfully, they can login to access the application's services.
- The user can set up their profile by adding a display photo in the *Profile* fragment.
- The user can access the *User* fragment to view a list of all other faculty belonging to the same institution along with which department they work at.
- To message any user, click on the name in the list.
- The user can send simple text messages. They can also upload photos and send it. The user is also provided with the feature for sending document files of different kinds such as PDF, ppt, docx, etc.
- Once the receiver opens the message, the application reports to the user about the same. It is indicated by the text *'seen'* under the sent message.
- The user also receives notifications in the notification bar of the phone, when the application is not open.

Back-end Server (Firebase):

- In order to make use of Firebase services in developing the mobile application, the Firebase project corresponding to the application must be created on the developer's Firebase account.

- Next, this project is connected to the project on Android studio with the help of the configuration file that is generated when the Firebase project is created. This file is added to the Android Studio project.
- The Authentication section displays the list of registered users. Since the Auth method in the project uses email as the credential, the email is displayed in the list.
- The firebase database stores the following information:
    1. *Chatlist* – indicates which users have a chat history
    2. *Chats* – It stores the history of messages sent by users. It contains the following fields:

        isseen: Value changes to "seen" when user opens the message

        message: Content of sent message

        receiver: Indicates the receiver's user ID

        sender: Indicates the sender's user ID

        type: Type of message sent (text/image/file)
    3. *Colleges* – List of colleges that can be selected by the user according to the institute they work at.
    4. *Department* – List of different departments
    5. *Tokens* – Used for sending notifications
    6. *Users* – User information is stored. With the firebase assigned user ID as the root, the fields are stored:

        college: College name of the user

        department: Department of the user

        id: Unique id assigned by Firebase to the user

        imageURL: Database URL of profile image set by the user. If no image is uploaded, this field is set to none.

        Search: User's name is converted to lower-case to help search function.

        Status: Indicates whether the user is online or offline.

        Username: Name of the user.

3.2 BUILT-IN FUNCTIONS USED

Any function that is provided as part of a high-level language and can be executed by a simple reference with specification of arguments. Some of the built-in functions provided by Android Java and Firebase that are used in the development of the project are:

• onCreate ()	:	Called when the activity is first created. This is where do all of the normal static is set up such as creating views, binding data to lists, etc.
• getInstance ()	:	Gets the default Firebase Database instance.
• getCurrentUser ()	:	The recommended way to get the current user is by calling the getCurrentUser method. If no user is signed in, it returns null
• getReference (String path)	:	Gets a Database Reference for the provided path.
• onDataChange ()	:	Read and listen for changes to the entire contents of a path.
• child (String path)	:	Get a DataSnapshot for the location at the specified relative path.
• getValue ()	:	getValue () returns the data contained in this snapshot as native types.
• setText (CharSequence text)	:	Sets the text to be displayed in a TextView.
• getSupportFragmentManager ()	:	Return the Fragment Manager for interacting with fragments associated with this activity.
• getChildren ()	:	Gives access to all of the immediate children of this snapshot.
• getUid ()	:	Returns a user identifier as specified by the authentication provider.
• startActivity (Intent intent Bundle bundle)	:	Launches a new activity.
• onResume ()	:	When the activity enters the resumed state, it comes to the foreground, and then the system invokes the onResume () callback. This is the state in which the app interacts with the user.
• onPause ()	:	The system calls this method as the first indication that the user is leaving the current activity. It indicates that the activity is no longer in the foreground.

3.3 USER-DEFINED FUNCTIONS

- register () : The function is called when all fields in the registration form have been filled correctly and proceeds to create a user account for the new user using the built-in firebase function. It also updates *Users* in the Firebase Database.
- sendMessage (String sender, final String receiver, String message) : The function is called when the send button is clicked. Creates a HashMap with all the details of the message and makes an entry under *Chats* in the database.
- sendNotification (String receiver, final String username, final String message) : Used to send notifications to the receiver when a message is sent.
- readMessage (final String myid, final String userid, final String imageurl) : Creates a list of all the chats the current user has.
- seenMessage (final String userid) : The function is called when the user opens the chat window to read the message. The parameter *'isseen'* in the database is changed to *true*.
- currentUser (String userid) : The function is used to create the Shared Preference "*currentUser*".
- status (String status) : The function is used to update the status of the user i.e. whether they are online or offline.
- uploadImage () : Used to store media messages (images) in Firebase Storage under the directory '*Image_messages*'. It also updates *Chats* in the backend database.
- uploadFile () : Used to store messages of type files or document in Firebase Storage under the directory *'Files'*. It also updates Chats in the backend database.
- reauthenticate () : Used to re-authenticate a user when they choose to change any of their existing credential like email, password, college or department.

3.4 CLASS DIAGRAM

Figure 6: Class Diagram of the Project

3.5 WIREFRAME (SCREEN FLOW)

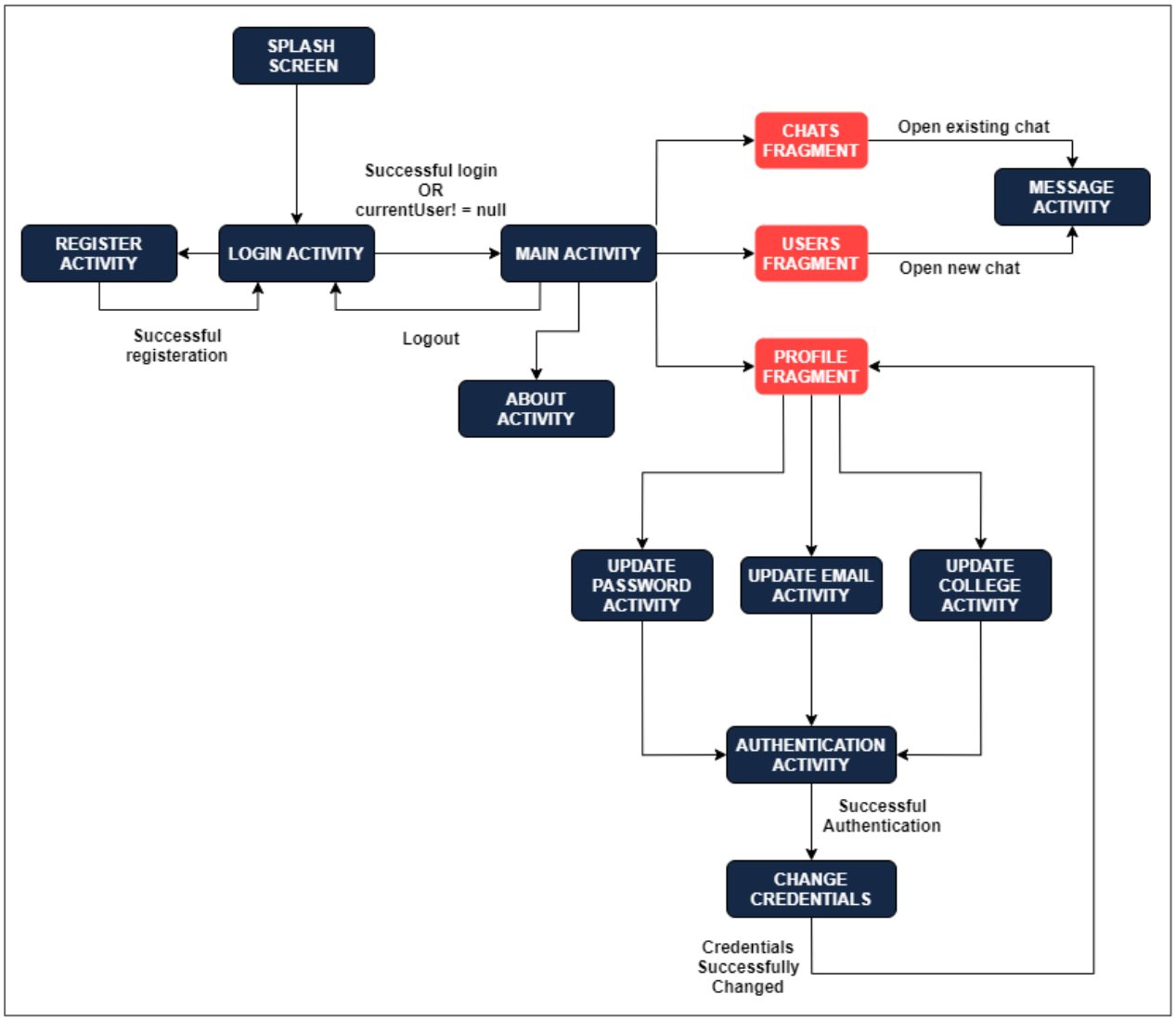


Figure 7: Wireframe

IMPLEMENTATION

4.1 SOURCE CODE

MainActivity.java

```
package com.mystaffroom.views;

import android.annotation.NonNull;
import android.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.text.TextUtils;
import android.util.Patterns;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ProgressBar;
import android.widget.Spinner;
import android.widget.TextView;
import android.widget.Toast;

import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.Task;
import com.google.firebase.auth.AuthResult;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.auth.FirebaseUser;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;
import com.mystaffroom.R;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

public class RegisterActivity extends AppCompatActivity {

    FirebaseAuth auth;
    DatabaseReference reference, database_college, database_departments;

    Spinner colleges_spinner, department_spinner;

    ProgressBar progressBar;
```

```
EditText username, email, password;
TextView login;
Button signup;

String selected_college, selected_department;

List<String> collegeList = new ArrayList<>();
List<String> departmentList = new ArrayList<>();

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_register);

    username = findViewById(R.id.edit_text_username);
    email = findViewById(R.id.edit_text_email);
    password = findViewById(R.id.edit_text_password);
    signup = findViewById(R.id.button_signup);
    login = findViewById(R.id.text_view_login);
    colleges_spinner = findViewById(R.id.college_spinner);
    department_spinner = findViewById(R.id.department_spinner);

    progressBar = findViewById(R.id.progressbar);

    collegeList.add("Select College");
    departmentList.add("Select your Department");

    auth = FirebaseAuth.getInstance();

    database_college =
    FirebaseDatabase.getInstance().getReference("Colleges");
    database_departments =
    FirebaseDatabase.getInstance().getReference("Department");

    database_college.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(DataSnapshot dataSnapshot) {
            for(DataSnapshot dataSnapshot1: dataSnapshot.getChildren()){
                String collegeName = dataSnapshot1.getValue(String.class);
                collegeList.add(collegeName);
            }
              ArrayAdapter<String> arrayAdapter = new ArrayAdapter<String>
        (RegisterActivity.this, android.R.layout.simple_spinner_item,
        collegeList);

        arrayAdapter.setDropDownViewResource(android.R.layout.simple_spinner_d
        ropdown_item);
            colleges_spinner.setAdapter(arrayAdapter);
        }
        @Override
```

```
        public void onCancelled(DatabaseError databaseError) {

              Toast.makeText(RegisterActivity.this,databaseError.getMessage(),
              Toast.LENGTH_LONG).show();
        }
    });

        database_departments.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(DataSnapshot dataSnapshot) {
            for(DataSnapshot dataSnapshot1: dataSnapshot.getChildren()){
                String deptName = dataSnapshot1.getValue(String.class);
                departmentList.add(deptName);
            }
              ArrayAdapter<String> arrayAdapter = new ArrayAdapter<String>
              (RegisterActivity.this, android.R.layout.simple_spinner_item,
              departmentList);

              arrayAdapter.setDropDownViewResource(android.R.layout.simple_spi
              nner_dropdown_item);
            department_spinner.setAdapter(arrayAdapter);
        }
        @Override
        public void onCancelled(DatabaseError databaseError) {

              Toast.makeText(RegisterActivity.this,databaseError.getMessage(),
              Toast.LENGTH_LONG).show();
        }
    });

        colleges_spinner.setOnItemSelectedListener(new
        AdapterView.OnItemSelectedListener() {
        @Override
        public void onItemSelected(AdapterView<?> parent, View view, int
        position, long id) {
            selected_college = (String) parent.getItemAtPosition(position);
        }

        @Override
        public void onNothingSelected(AdapterView<?> parent) {
        }

    });

    department_spinner.setOnItemSelectedListener(new AdapterView.
    OnItemSelectedListener() {
        @Override
        public void onItemSelected(AdapterView<?> parent, View view, int
        position, long id) {
            selected_department = (String) parent.getItemAtPosition(position);
        }
```

```
    @Override
    public void onNothingSelected(AdapterView<?> parent) {
    }

});

login.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
          Intent intent = new Intent(RegisterActivity.this,
          LoginActivity.class);
          intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TASK |
          Intent.FLAG_ACTIVITY_NEW_TASK);
          startActivity(intent);
          finish();
    }
});

signup.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        String username_text = username.getText().toString().trim();
        String email_text = email.getText().toString().trim();
        String password_text = password.getText().toString().trim();

        if (TextUtils.isEmpty(username_text)){
            username.setError("Username required");
            username.requestFocus();
            return;
        }
        if (TextUtils.isEmpty(email_text)){
            email.setError("Email required");
            email.requestFocus();
            return;
        }
        if (password_text.length()<8 || password_text.length()>12){
            password.setError("Password should be 8-12 characters");
            password.requestFocus();
            return;
        }

        if (!Patterns.EMAIL_ADDRESS.matcher(email_text).matches()) {
            email.setError("Invalid Email");
            email.requestFocus();
            return;
        }
        if (username.length()<2){
            username.setError("Invalid Name");
            username.requestFocus();
```

```
                return;
            }
            register(username_text,email_text,password_text);

        }
    });

}

private void register(final String username, String email, String password)
{
    if (selected_college.equals("Select College") ){
        Toast.makeText(RegisterActivity.this,"Please select college",
        Toast.LENGTH_LONG).show();
        return;
    }
    if (selected_department.equals("Select your Department")){
        Toast.makeText(RegisterActivity.this, "Please select department",
        Toast.LENGTH_LONG).show();
        return;
    }
    progressBar.setVisibility(View.VISIBLE);
    auth.createUserWithEmailAndPassword(email,password)
            .addOnCompleteListener(new OnCompleteListener<AuthResult>() {
                @Override
                public void onComplete(@NonNull Task<AuthResult> task) {
                    if (task.isSuccessful())
                    {
                        FirebaseUser firebaseUser = auth.getCurrentUser();
                        assert firebaseUser != null;
                        String userid = firebaseUser.getUid();

        reference =
        FirebaseDatabase.getInstance().getReference("Users").child(userid);

                        HashMap<String, String> hashMap = new HashMap<>();

                        hashMap.put("id",userid);
                        hashMap.put("username", username);
                        hashMap.put("imageURL", "default");
                        hashMap.put("status", "Offline");
                        hashMap.put("search", username.toLowerCase());
                        hashMap.put("college", selected_college);
                        hashMap.put("department", selected_department);

        reference.setValue(hashMap).addOnCompleteListener(new
        OnCompleteListener<Void>() {
        @Override
        public void onComplete(@NonNull Task<Void> task) {
             if (task.isSuccessful()){
```

```
                Intent intent = new Intent(RegisterActivity.this,
                LoginActivity.class);
                Toast.makeText(RegisterActivity.this,"Account created! Please
                login",Toast.LENGTH_LONG).show();
                FirebaseAuth.getInstance().signOut();

                intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TASK |
                Intent.FLAG_ACTIVITY_NEW_TASK);
                startActivity(intent);
                progressBar.setVisibility(View.GONE);
                finish();
                                    }
                                }
                            });
                        } else
                        {

                            Toast.makeText(RegisterActivity.this,"Registeration not
                            successful. Try again",Toast.LENGTH_LONG).show();
                            progressBar.setVisibility(View.GONE);
                        }
                    }
                });
            }
        }
```

TESTING AND RESULTS

Testing is defined as an activity to check whether the actual results match the expected results and to ensure that the software system is Defect free. It involves execution of a software component or system component to evaluate one or more properties of interest. Testing also helps to identify errors, gaps or missing requirements in contrary to the actual requirements. It can be either done manually or using automated tools. Some prefer saying Software testing as a White Box and Black Box Testing.

5.1 DIFFERENT TYPES OF TESTING

- **Unit Testing**

  Individual components are tested to ensure that they operate correctly. Each component is tested independently, without other system components.
- **Module Testing**

  A module is a collection of dependent components such as an object class, an abstract data type or some looser collection of procedures and functions. They are module related components, so can be tested without other system modules.
- **System Testing**

  This is concerned with finding errors that result from unanticipated interaction between sub-system interface problems.
- **Acceptance Testing**

  The system is tested with data supplied by the system customer rather than simulated test data.

5.2 TEST CASES

No.	Test Case	Expected Result	Actual Result	Remarks
1.	Registration Form Validation	All fields must be validated syntactically for correct entries. All mandatory fields must be filled.	All fields are validated.	PASS
2.	Login with invalid email	Display error message	Toast with error message displayed.	PASS
3.	Enter incorrect password	Display error message	Toast with error message displayed.	PASS

4.	Send blank message	Display warning that no text has been typed.	Toast message requesting to type a message is displayed	PASS
5.	First Message	Move user to *Chats* fragment when first message is sent to the user	Moves to *Chat* fragment after receiver replies back.	FAIL
6.	File Type	User should be allowed only to send specific type of files.	When '*Image'* is selected, only image files are displayed. When '*File'* is selected, only document files are displayed.	PASS
7.	User status	User status must change when they go online or offline.	User status is updated appropriately.	PASS
8.	Message status.	User must be shown when message is delivered and when it is opened by the receiver.	Message status changes from '*Delivered'* to '*Seen'* when opened by the receiver.	PASS
9.	Send notification when application is not in foreground.	User must receive message notification in notification bar when a message is sent to them.	Notification is received.	PASS
10.	Do not send notification when application is in foreground.	Application must not receive if user is currently using the application.	Notification is received when user in all but chat activity.	FAIL
11.	Update Profile Photo	Tap on the profile photo icon in *Profile* section to select new photo.	User can change profile photo by selecting from gallery.	PASS
12.	Change college credential	User must be able to change to a different college.	User is provided with a dropdown menu to select new college. If the same college is selected, user is given an error toast.	PASS
13.	Messages are sent and received without delay.	With good internet connection on sender and receiver side, messages sent must be updated on the application with less or no delay.	Messages are instantly updated on the chat screen.	PASS
14.	Send message offline	Message sent without internet connection. Message received when connection is established.	Receiver receives message but not notified.	FAIL

OUTPUTS AND SNAPSHOTS

The section describes the activity screens of the MyStaffRoom mobile application. The snapshots are shown below for each module.

Figure 8: Splash Screen

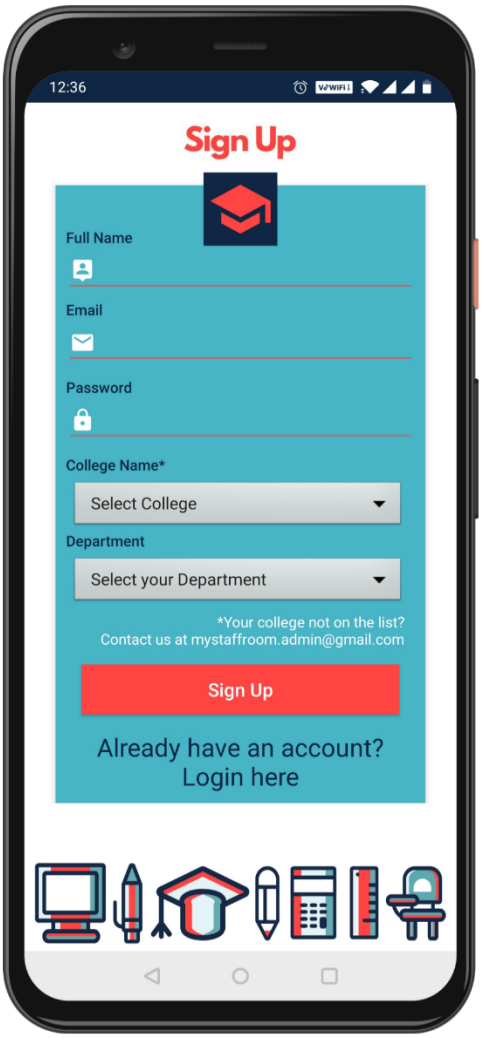


Figure 9: Registration Screen

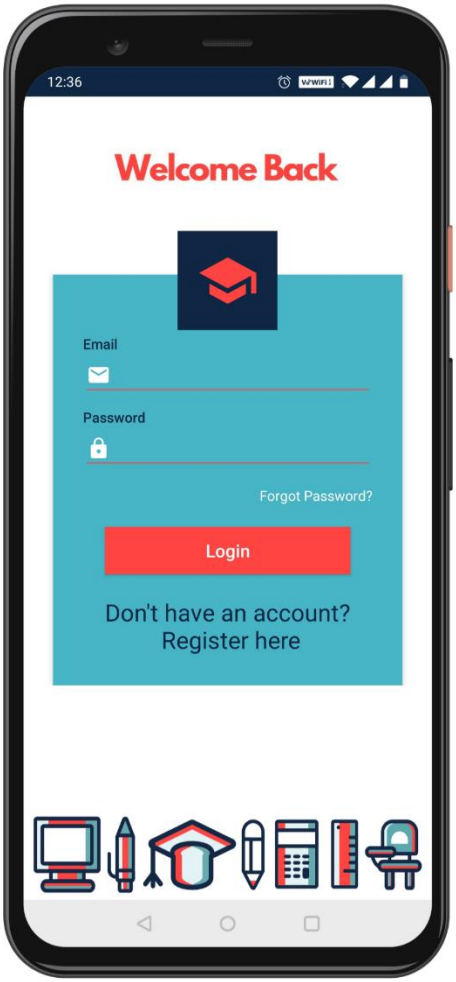


Figure 10: Login Screen

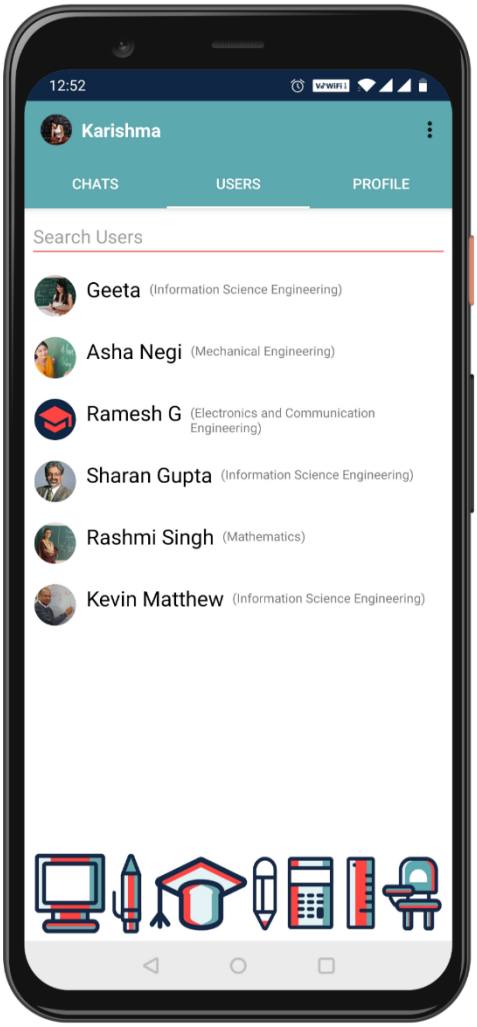


Figure 11: Users' Screen

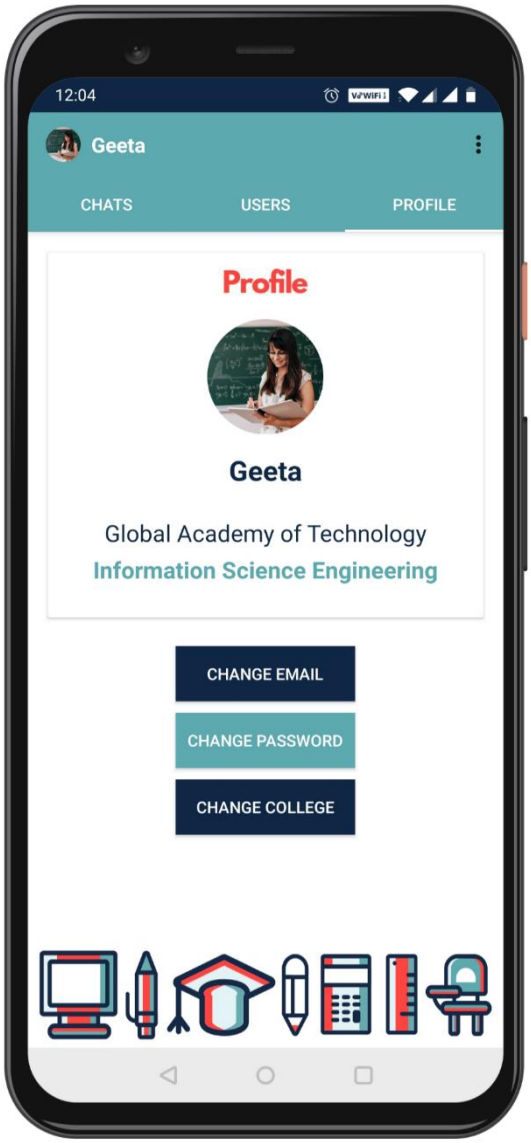


Figure 12: Profile Activity

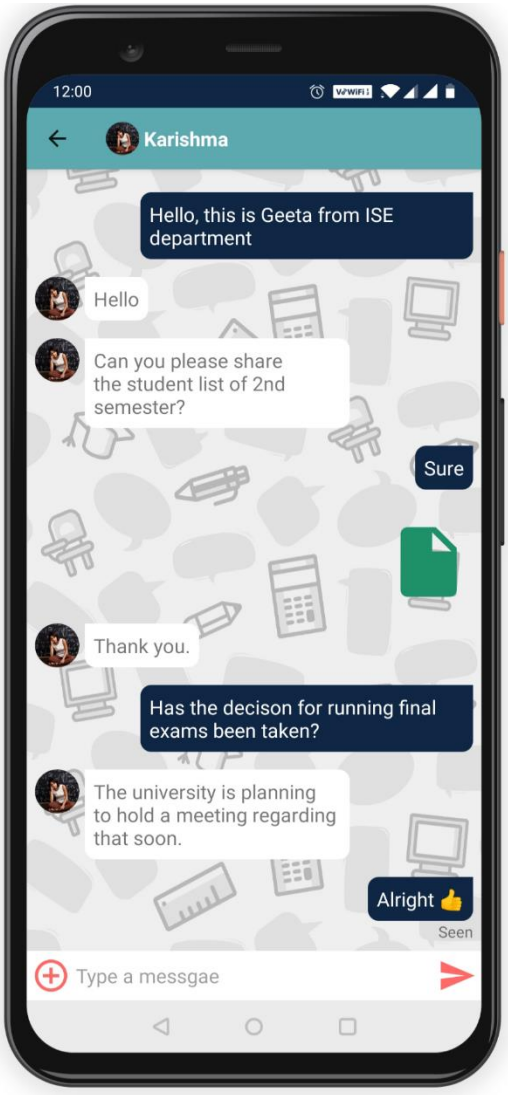


Figure 13: Chat Screen

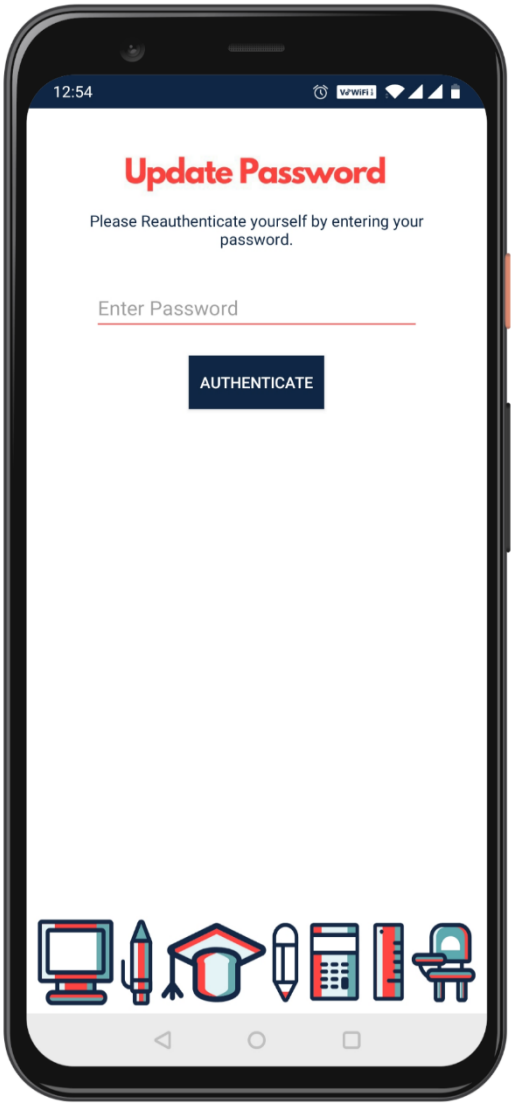


Figure 14: Authenticate Screen

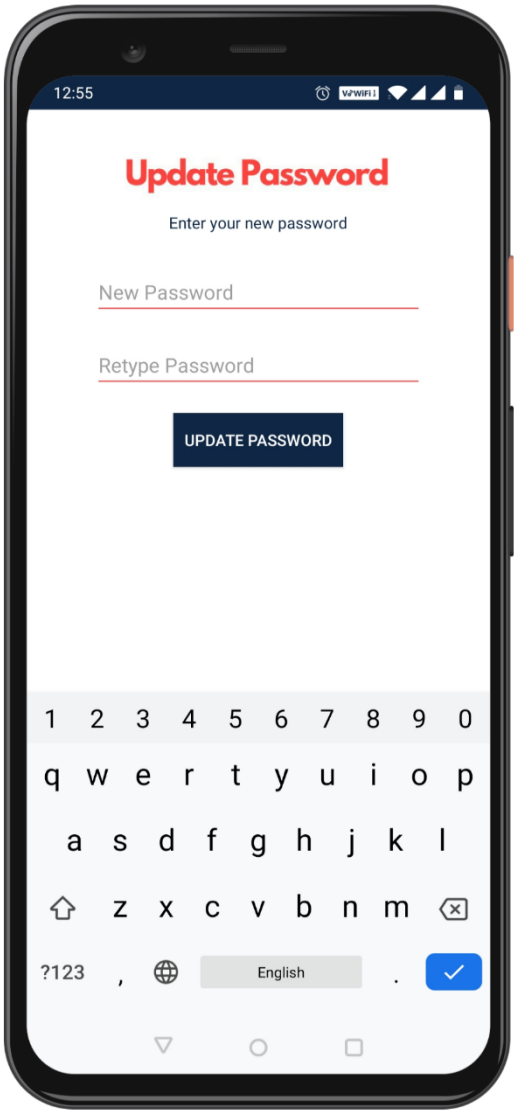


Figure 15: Update Password Screen

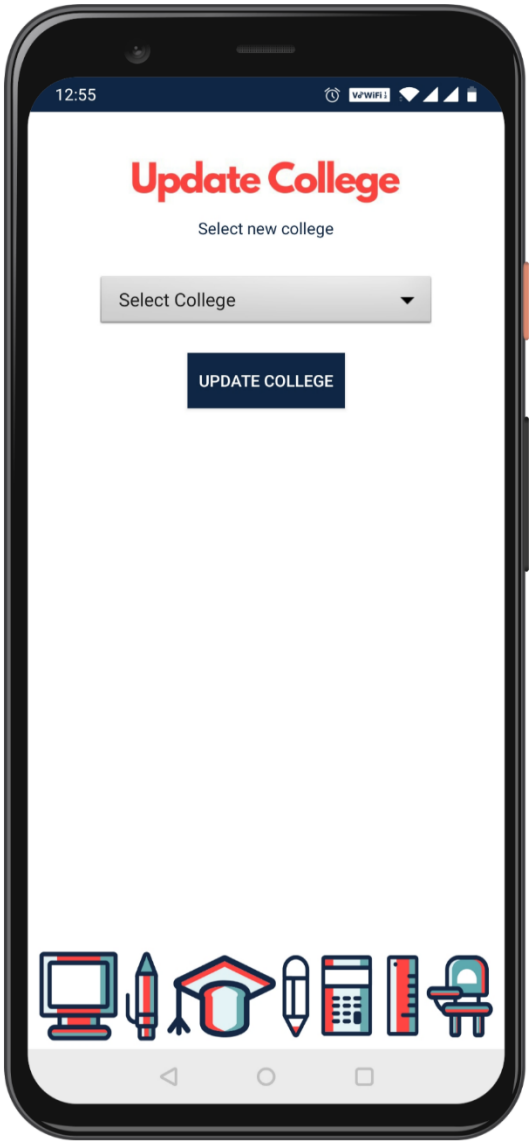


Figure 16: Update College Screen

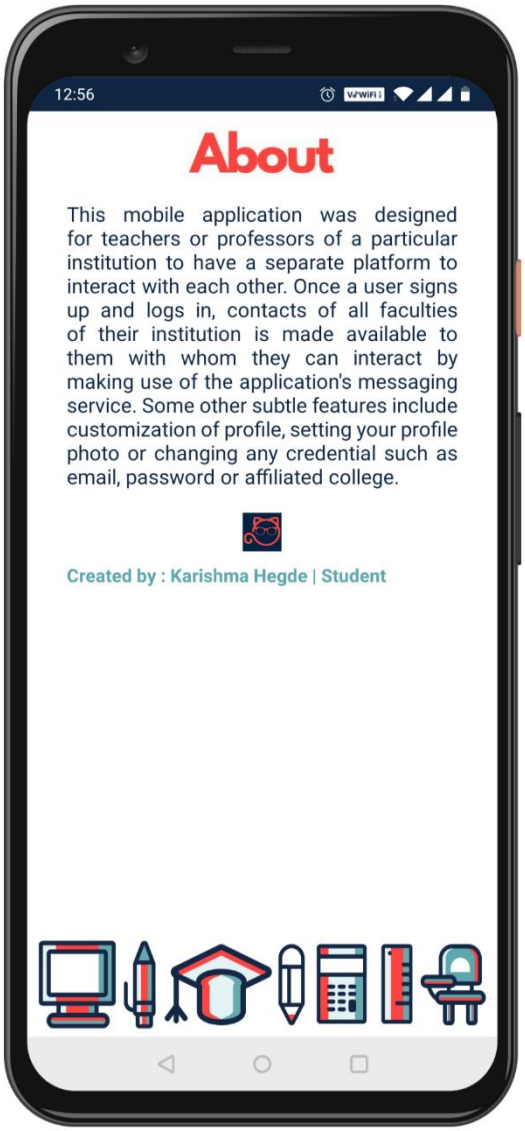


Figure 17: About Screen

CONCLUSION

In recent years, with the rapid growth in technology, mobile phones have become a vital part in human life. Although the core functionality of mobile phones is telecommunication, mobile applications have made many things possible which was previously unimaginable. Smartphones have given a new face to service sector with the invention of mobile applications. Mobile applications are designed to perform a specific task. Time consumption has been reduced with the popularity of mobile applications and now goods and services are at the user's fingertips.

The project involved the development of an Android application for messaging. The application is designed for professors or teachers of any institute to interact with their colleagues via the mobile application. As there is no dedicated messaging service for the teaching community, this application serves the purpose. The application involves a front end developed in Android Studio and a backend server which is facilitated by Google Firebase. Firebase provides a variety of features such as storage, database and notification services that could be implemented conveniently for free. Firebase is a suitable solution for mobile application development as programmers can concentrate on the mobile application side, while the back-end server solutions are taken care. The project allows a user to access and communicate to any faculty belonging to the same institution.

REFERENCES

[1] Chat App with Firebase – Android Studio Tutorials https://www.youtube.com/playlist?list=PLzLFqCABnRQftQQETzoVMuteXzNiXmnj8

[2] Messaging Apps https://en.wikipedia.org/wiki/Messaging_apps

[3] Android Software Development https://en.wikipedia.org/wiki/Android_software_development

[4] What is Mobile Application Development https://aws.amazon.com/mobile/mobile-application-development/

[5] Cross-platform vs Native Mobile App Development https://rb.gy/hmzu4m

[6] Android Studio https://searchmobilecomputing.techtarget.com/definition/Android-Studio

[7] Configure your build https://developer.android.com/studio/build

[8] What is Firebase? https://www.educative.io/edpresso/what-is-firebase

[9] WhatsApp Requirement Specification https://frost.ics.uci.edu/inf43/SampleSRS5.pdf

[10] Firebase https://en.wikipedia.org/wiki/Firebase

[11] Firebase Authentication https://firebase.google.com/docs/auth

[12] Manage Users in Firebase https://firebase.google.com/docs/auth/android/manage-users

[13] Data Snapshot https://firebase.google.com/docs/reference/android/com/google/firebase/database/DataSnapshot

[14] Chat App with Firebase - Send Image in message https://www.youtube.com/watch?v=0GKFKJ3lT38&t=368s