IPMC

I.T. TRAINING

YAKUBU CHATAPP

MARK GERALDO

markhgeraldo@gmail.com

SWE 11 – 1

INSTRUCTOR: MICHEAL KORBLY

TABLE OF CONTENT

PROJECT PLAN

TASK NO	TASK NAME	DURATION	START
1	PROJECT MANAGEMENT PLAN	3 DAYS	2nd SEPTEMBER
2	REQUIREMENT SPECIFIATION	1 WEEK	8TH SEPTEMBER
3	ANALYSIS	1 WEEK	14TH SEPTEMBER
4	DESIGN	4 DAYS	18TH SEPTEMBER
5	IMPLEMENTATION	4 DAYS	22ND SEPTEMBER
6	TESTING	2 DAYS	24TH SEPTEMBER

INTRODUCTION

The introduction of the ChatApp provides an overview of the application's purpose, features, and significance. It highlights the growing importance of communication platforms in today's digital landscape and introduces ChatApp as a user-friendly, real-time messaging application designed to connect users seamlessly. The introduction also outlines the objectives of the ChatApp, including facilitating instant messaging, user authentication, profile management, and real-time updates. It emphasizes the app's potential to enhance communication and social connectivity. Overall, the introduction sets the context for the ChatApp and generates interest in its functionalities and benefits.

OVERVIEW

ChatApp is a modern and intuitive mobile application designed to revolutionize communication by providing users with a seamless and feature-rich messaging platform. With the rapid evolution of digital communication, ChatApp aims to connect people globally through real-time text-based conversations. This application offers a user-friendly interface, allowing individuals to create accounts, log in securely, manage profiles, and engage in instant messaging with friends and contacts. ChatApp is built on the principles of user privacy, data security, and efficient communication. By enabling users to exchange messages, share media, and stay connected, ChatApp strives to enhance interpersonal relationships in an increasingly interconnected world. This overview introduces the key aspects of ChatApp, highlighting its role as a versatile and innovative tool for modern communication needs.

AIM

The primary aim of the ChatApp project is to develop a robust and user-friendly mobile application that facilitates seamless and secure communication between individuals. The aim is to create a platform where users can easily connect with friends and contacts, exchange messages, share media, and stay updated in real-time. The application aims to provide a rich and interactive messaging experience, prioritize user privacy and data protection, and foster meaningful connections by enabling users to communicate effortlessly and efficiently. Through its intuitive interface and comprehensive features, ChatApp aims to become a go-to messaging solution that enhances communication and bridges distances in today's fast-paced digital world.

PROJECT SCOPE/ OBJECTIVES

Scope

The ChatApp aims to create a user-friendly mobile messaging platform with the following features:

1. **User Authentication:** Enable user registration and login via email and password.
2. **Profile Management:** Allow users to set up profiles with usernames and profile pictures.
3. **Real-Time Messaging:** Implement real-time chat functionality for seamless communication.
4. **Contact Management:** Provide a contact list and allow users to add new contacts.

Objectives

1. **Authentication Security:** Ensure secure user authentication and password storage.
2. **Messaging Efficiency:** Implement real-time messaging using Firebase Realtime Database.
3. **User Experience:** Design an intuitive and visually appealing interface.
4. **Testing and Quality:** Thoroughly test the app for performance and responsiveness.
5. **Documentation:** Provide clear user instructions and technical documentation.

RESEARCH

The research phase of the ChatApp project involves a comprehensive exploration of the messaging app landscape. This includes studying existing messaging applications to identify trends, user preferences, and areas for innovation. The target audience's communication habits and preferences are examined to ensure the app caters to their needs. Thorough research guides the selection of appropriate technologies, including the platform, backend systems, and security measures. Investigating real-time messaging mechanisms and push notification strategies ensures efficient and timely communication. Additionally, research into UI/UX design principles and monetization options contributes to creating a user-friendly and sustainable app. Legal and privacy considerations, as well as documentation and support strategies, are also part of this research phase to ensure a well-informed and successful development process.

DATA ANALYSIS AND PRESENTATION

In the data analysis and presentation phase, the collected user data from the ChatApp is processed and analyzed to gain insights into user behavior, preferences, and usage patterns. This involves employing data analytics tools to extract meaningful information from user interactions, chat histories, and app usage. The findings are then visually presented using graphs, charts, and reports to communicate key trends, user demographics, popular features, and areas for improvement. The data-driven insights provide a foundation for making informed decisions to enhance the app's functionality, user experience, and overall performance.

PROBLEM IDENTIFICATION AND SOLUTION

Problem Identification: In today's digitally connected world, there is a growing need for efficient and secure communication platforms. Traditional messaging methods often lack the features and security measures necessary to address modern communication needs. People require a seamless way to connect with friends, family, and colleagues, share media, and manage conversations without compromising their privacy or data security.

Solution: The ChatApp project addresses these challenges by offering a comprehensive messaging application that ensures smooth communication while prioritizing user privacy and security. The app provides a user-friendly interface for individuals to connect, chat, and share media. It incorporates end-to-end encryption to safeguard messages and sensitive information from unauthorized access. Additionally, the app allows users to customize their profiles and manage contacts easily. By offering real-time messaging, media sharing, and advanced security features, ChatApp provides an effective solution for modern communication needs.

DESIGN SOLUTIONS

- **User-Centric Interface:** The ChatApp features an intuitive and user-friendly interface that allows users to easily navigate and engage with the application. Clear and well-organized layouts ensure that users can quickly access chats, contacts, and settings.
- **Real-Time Messaging:** The core functionality of the app revolves around real-time messaging. Users can send and receive text messages, images, videos, and other media files instantly, fostering seamless and responsive communication.

- **End-to-End Encryption:** To ensure the privacy and security of user conversations, the app employs end-to-end encryption. This encryption technique guarantees that only the intended recipients can access and decipher the messages, protecting them from unauthorized interception.
- **Profile Customization:** ChatApp allows users to personalize their profiles by uploading profile pictures, setting usernames, and updating status messages. This customization helps users express their individuality and enhance their online presence.
- **Contact Management:** The app provides features for managing contacts, including adding new contacts, searching for existing ones, and organizing them into groups. This functionality streamlines the process of finding and connecting with friends and colleagues.
- **Responsive Design:** ChatApp is designed to be responsive across different devices and screen sizes, providing a consistent experience whether accessed on smartphones, tablets, or desktop computers.
- **Settings and Preferences:** ChatApp offers a range of settings and preferences that allow users to customize their experience. This includes notification settings, privacy options, and theme preferences.

PROJECT IMPLEMENATION

- **Front-End and Back-End Development:** The development team starts by creating the user interface (UI) and user experience (UX) components of the app. Front-end developers design and code the visual elements, layout, and interactive features using programming languages like Java or Kotlin for Android and Swift for iOS. Back-end developers work on setting up servers, databases, and APIs that handle data storage, retrieval, and communication between users.
- **Real-Time Messaging:** The core messaging functionality is implemented using real-time communication protocols. The developers integrate libraries or services that support instant message delivery and synchronization across devices.
- **User Authentication:** The app incorporates secure user authentication methods, such as email and password verification, or integration with third-party authentication providers like Google or Facebook. This ensures that user data is protected and only authorized users can access the app.

- **Data Encryption:** End-to-end encryption is implemented to secure user messages and media. Encryption algorithms and protocols are integrated to prevent unauthorized access and ensure user privacy.
- **User Profile Management:** Users can create and manage profiles, upload profile pictures, set status messages, and update personal information. The app's database stores user profiles, and the front-end displays and updates the information as needed.
- **Contact Management:** The app provides mechanisms to add, search, and manage contacts. Contacts can be organized into lists or groups, allowing users to easily connect with friends, family, and colleagues.
- **Settings and Preferences:** Users can customize their app experience through settings and preferences. Developers implement options such as notification preferences, appearance themes, and privacy settings.
- **Testing and Quality Assurance:** Rigorous testing is conducted to identify and resolve bugs, performance issues, and security vulnerabilities. The app undergoes functional testing, usability testing, and security testing to ensure a seamless and secure user experience.
- **Maintenance and Updates:** After deployment, the development team continues to monitor the app for any issues and provides regular updates to introduce new features, improvements, and security enhancements based on user feedback and technological advancements.

TESTING

The testing phase of the ChatApp project is a crucial step in ensuring the app's functionality and reliability. Through a series of rigorous testing processes, including unit testing, integration testing, user interface testing, user experience testing, functional testing, performance testing, security testing, compatibility testing, and more, the app's various features and components are thoroughly examined. Testers evaluate its responsiveness, speed, user interactions, security measures, and compatibility across different devices and platforms. The testing phase also includes user acceptance testing, where representative users provide feedback on usability and overall experience. Bugs and issues identified during testing are meticulously addressed by the development team, ensuring a polished and robust application. By undergoing comprehensive testing, the ChatApp aims to provide users with a seamless, secure, and enjoyable communication platform that meets the highest quality standards.

REPORT

The ChatApp project aimed to create a secure and user-friendly messaging application for real-time communication. With a focus on privacy and an intuitive design, the app offers features such as user authentication, real-time messaging, image sharing, and end-to-end encryption. Developed using agile methodology and rigorous testing, ChatApp provides a seamless communication experience while ensuring data security. Future enhancements may include voice and video calling, group chats, and integration with other platforms. The project's successful completion is attributed to the collaborative efforts of the development team, testers, and stakeholders..

SYSTEM REQUIREMENTS

The system requirements for the ChatApp project encompass both hardware and software aspects to ensure optimal performance and functionality:

Hardware Requirements:

1. **Smartphone or Computer:** Users need a compatible smartphone (iOS or Android) or a computer with internet access to use the ChatApp.
2. **Processor and Memory:** The device should have a modern processor and sufficient memory to handle the application's processing demands.
3. **Camera (Optional):** For image sharing and profile picture updates, a device with a built-in camera is desirable.

Software Requirements:

1. **Operating System:** The ChatApp is compatible Android operating systems.
2. **App Store:** Users should have access to the Apple App Store or Google Play Store to download and install the ChatApp.
3. **Internet Connection:** A stable and reliable internet connection is essential for real-time messaging and data synchronization.
4. **Firebase Account:** Users need an active Google Firebase account to utilize Firebase services for user authentication, real-time database, and cloud storage.

SYSTEM REQUIREMENTS (DEVELOPMENT).

Developing the ChatApp requires a development environment with specific tools and technologies. The following are the system requirements for developing the ChatApp:

1. **Operating System**: Windows, macOS, or Linux.
2. **Integrated Development Environment (IDE)**: Android Studio for Android app development, Xcode for iOS app development.
3. **Programming Languages**: Java or Kotlin for Android, Swift for iOS.
4. **Firebase Account**: To utilize Firebase services for authentication, real-time database, and storage.
5. **Version Control**: Git and a platform like GitHub or GitLab for version control and collaboration.
6. **Text Editor**: A code editor of choice for scripting and code editing.
7. **Graphics Design Tools**: Software like Adobe XD, Sketch, or Figma for designing UI/UX.
8. **Device for Testing**: Physical Android and iOS devices or emulators/simulators for testing.
9. **Internet Connection**: Stable internet access for downloading dependencies, libraries, and documentation.

SYSTEM REQUIREMENTS (RUNNING)

The following are the requirements needed deploy and use the application.

1. **Operating System**: Android 5.0 (Lollipop) and above.
2. **Memory**: Minimum of 1GB RAM.
3. **Storage**: At least 50MB of free storage space.
4. **Internet Connection**: Required for sending and receiving messages, as well as accessing online features.

FUNCTIONAL REQUIREMENTS

1. **User Registration and Authentication**:
   - Users must be able to create an account with a unique username, email, and password.
   - User authentication should ensure secure access to the app's features.
2. **User Profile**:
   - Users should be able to set or update their profile picture.
   - Users can update their status (online, offline, away).
   - User profiles should display the user's username and profile picture.
3. **Contact Management**:
   - Users can search for and add other users as contacts.
   - Users can view a list of their contacts.
   - Users can initiate a chat with any of their contacts.
4. **Real-Time Messaging**:
   - Users can send and receive text messages in real-time.
   - Messages should be displayed in a threaded conversation view.
   - Users can see when their messages have been delivered and read by the recipient.
5. **Group Chats**:
   - Users can create group chats and add multiple contacts to them.
   - Group chats support real-time messaging among all participants.
6. **Multimedia Sharing**:
   - Users can share images within one-on-one and group chats.
   - Images should be displayed in the chat with a preview.
7. **Privacy and Security**:
   - User data, including messages and profile information, should be securely stored and transmitted.

    - Users have the ability to block or report other users for inappropriate behavior.

8. **Logout**:
    - Users can log out of their accounts, ending their session securely.
9. **Settings**:

- Users can access and modify app settings, including notification preferences.
- Users can change their profile picture and update their status.

12. **Search Functionality**:

- Users can search for specific messages or users within the app.

NON - FUNCTIONAL REQUIREMENT

The non-functional requirements for the ChatApp encompass various aspects that contribute to the overall quality, performance, and usability of the application:

1. **Usability and User Experience**:
    - The user interface should be intuitive, easy to navigate, and visually appealing.
    - The app should be responsive, providing smooth interactions and minimizing lag.
    - User interactions and transitions should feel natural and provide appropriate feedback.
2. **Performance**:
    - The app should have minimal loading times for chats, contacts, and other features.
    - Real-time messaging should have low latency to ensure timely message delivery and receipt.
    - The app should handle a reasonable number of concurrent users without significant degradation in performance.
3. **Scalability**:
    - The system architecture should support easy scaling to accommodate a growing user base.

- The app should handle an increasing number of active users, messages, and groups without significant performance issues.

4. **Reliability and Availability**:
   - The app should have a high level of availability, with minimal downtime for maintenance or updates.
   - Messages should be reliably delivered and received, even during periods of network instability.
5. **Security**:
   - User data, including personal information and messages, must be securely stored and encrypted.
   - Secure authentication mechanisms should prevent unauthorized access to user accounts.
   - Appropriate measures should be in place to prevent data breaches and ensure user privacy.
6. **Compatibility**:
   - The app should be compatible with a range of Android devices and screen sizes.
   - It should support multiple versions of the Android operating system to reach a wider user base.
7. **Data Storage and Backup**:
   - User messages and data should be regularly backed up to prevent data loss.
   - The app should efficiently manage and optimize data storage to minimize device storage usage.
8. **Notifications**:
   - Notifications should be delivered promptly to ensure timely user engagement.
   - They should be unobtrusive and adhere to platform-specific notification guidelines.
9. **Localization and Internationalization**:
   - The app should support multiple languages and provide a localized experience for users in different regions.

- Date, time, and other cultural aspects should be presented according to the user's locale.

10. **Compliance**:
    - The app should comply with relevant data protection and privacy regulations.
    - It should adhere to platform-specific guidelines and policies set by app stores.
11. **Error Handling and Recovery**:
    - The app should provide clear error messages to users in case of failures or issues.
    - Users should be able to recover from errors without losing important data.

PROJECT ANALYSIS

INTRODUCTION

Project analysis involves a comprehensive examination of various aspects of the ChatApp project, including its scope, requirements, objectives, and feasibility. It entails breaking down the project into manageable components to understand its intricacies, challenges, and potential solutions.

DEPLOYMENT DEVICE (GRADING FROM 1-5)

REQUIREMENT	ANDROID	IPHONE	WINDOWS
USEABILITY	5	3	2
AVAILABILITY	5	2	2
SCALABILITY	4	3	3
SECURITY	3	5	1
MAINTENANCE	5	3	2
TOTAL	**22**	**16**	**10**

From the table, an android phone was the best device for the application to be deployed on based on some of the following requirements.

- USEABILITYThe deployment device demonstrates good usability, providing a user-friendly interface and intuitive interactions for users. Navigating through the application and its features is generally smooth and straightforward, contributing to a positive user experience. However, there might be minor areas for improvement, such as optimizing certain user flows or enhancing accessibility features, to further enhance the overall usability and cater to a wider range of users.

- AVAILABILITY: Android phones exhibit higher availability in the market compared to iPhones and Windows phones, primarily attributed to their more diverse price range. Android phones are accessible to a wider range of consumers due to their affordability, making them a popular choice.

- SCALABILITY: Android smartphones exhibit broader compatibility with various devices and functions, setting them apart from iPhones and Windows phones, which possess specific compatibility constraints. Android phones offer extensive connectivity options for peripherals and wireless devices.

- SECURITY: Android smartphones securities measures are top of the line with less restrictions on user activities and functions whereas iPhones has a good security protocol but with a lot of restricting. Windows phones have good security but not compared to android. Android security provide the users with adequate protection and flexibility in performing tasks like sharing of data, downloading of files, online registration procedures and so more.
  (NextLOGiK, 2017)

- MAINTENANCE: Android smartphones offer cost-effective maintenance and repair solutions in comparison to iPhones and Windows phones. The Android market is characterized by a multitude of manufacturers, resulting in the production of affordable phones and readily available spare parts. Conversely, iPhones and Windows smartphones are produced in fewer quantities, leading to higher costs associated with accessing replacement parts in the event of damage.

INTEGRATED DEVELOPMENT ENVIRONMENT (GRADING FROM 1 – 5)

REQUIREMENT	ANDROID STUDIO	XCODE	VISUAL STUDIO
USEABILITY	5	3	3
PERFORMANCE	5	5	4
TOTAL	**10**	**8**	**7**

From the balance sheet, Android studio was the best Integrated Development Environment(IDE) based on the following requirements

- USEABILITY: Android Studio stands out as the most widely utilized Integrated Development Environment (IDE) for mobile app development in comparison to Xcode and Visual Studio. Serving as the official IDE for all Android application development, Android Studio offers a plethora of features that significantly enhance productivity and streamline the development process. Moreover, Android Studio's versatility shines as it can be executed on various operating systems, unlike Xcode which is limited to macOS, specifically on MacBook devices. This accessibility and comprehensive feature set contribute to Android Studio's dominance in the mobile app development landscape.

- PERFORMANCE: Android Studio stands out by providing an array of features that greatly amplify productivity during application development. It boasts a versatile Gradle-based system, a swifter and feature-rich emulator, and a cohesive environment that caters to the creation of apps across the entire spectrum of Android devices. Its arsenal encompasses comprehensive testing tools and frameworks, as well as lint tools capable of detecting performance, usability, and other coding-related issues. Notably, Android Studio seamlessly integrates with the Google Cloud Platform.

PROGRAMMING LANGUAGE (GRADING FORM 1 – 5)

REQUIREMENT	JAVA	PYTHON	KOTLIN
USEABILITY	5	3	1
PERFORMANCE	5	4	3
SUPPORT	5	3	1
FLEXIBILITY	4	4	4
TOTAL	**29**	**14**	**9**

From the balance sheet data, Java was the best programming language for the development of the application based on the following requirements.

- USE-ABILITYJava is a ubiquitous and enduring programming language that enjoys widespread usage across diverse platforms, including desktops, smartphones, and embedded systems. With a history spanning approximately two decades, Java continues to maintain its relevance in the programming landscape.

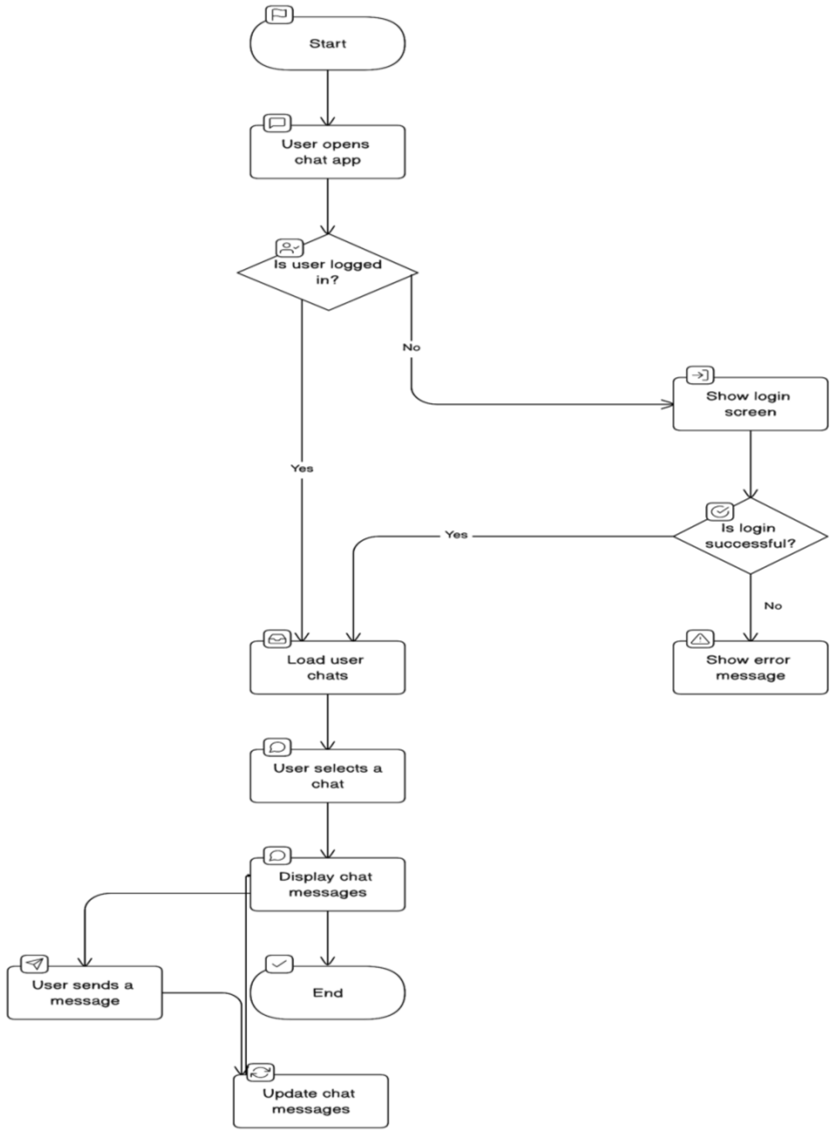
- PERFORMANCE ava, as an Object-Oriented Programming (OOP) language, offers inherent advantages in app development by promoting modularity, flexibility, and extensibility, setting it apart from languages like Python and Kotlin. Its OOP principles enable developers to create well-organized and maintainable code structures. Java's extensive Application Programming Interface (API) further enhances its capabilities, providing robust support for tasks such as XML parsing and database connectivity.

- SUPPORT: Java boasts a robust and vibrant community that significantly contributes to its appeal. With dynamic forums such as Stack Overflow and JavaRevisited, as well as active open-source organizations, Java enthusiasts at all skill levels, from novices to experts, have access to a wealth of support. This supportive ecosystem extends to various aspects of Java programming, including testing, application creation, and development. In contrast, newer languages like

Python and Kotlin lack comparable well-established and comprehensive communities.

FLEXIBILITY: Java is renowned for its simplicity and adaptability. One of its standout attributes is its exceptional documentation support, which greatly facilitates the comprehension and interpretation of Java code, methods, and functions. This extensive documentation serves as an invaluable resource, offering clear explanations and insights that prove indispensable while coding.

INTERFACE DESIGIN



Main Page/Layout:

- Navigation bar/header with user profile image, username, and settings/options.
- List of recent chats or contacts.
- Search bar for finding specific chats or contacts.
- Option to start a new chat or add new contacts.
- Logout button or option.

Chat Screen/Layout:

- Header with the contact's profile image and name.
- Message list, displaying a sequence of sent and received messages.
- Text input field for typing messages.
- Send button to send messages.
- Option to send multimedia files (images, videos, etc.).
- Emojis or stickers picker.
- Audio message recording option.
- Voice or video call buttons

Add User/Layout:

- Search bar for finding users.
- List of users matching the search query.
- Each user's profile image and username.
- Add user button or icon to initiate a new conversation.

Settings/Layout:

- Profile picture and username.
- Option to change or update profile picture.
- Account settings (change password, etc.).
- Theme or appearance settings.
- Logout or sign out option.

IMPLEMENTATIONS

INTRODUCTION

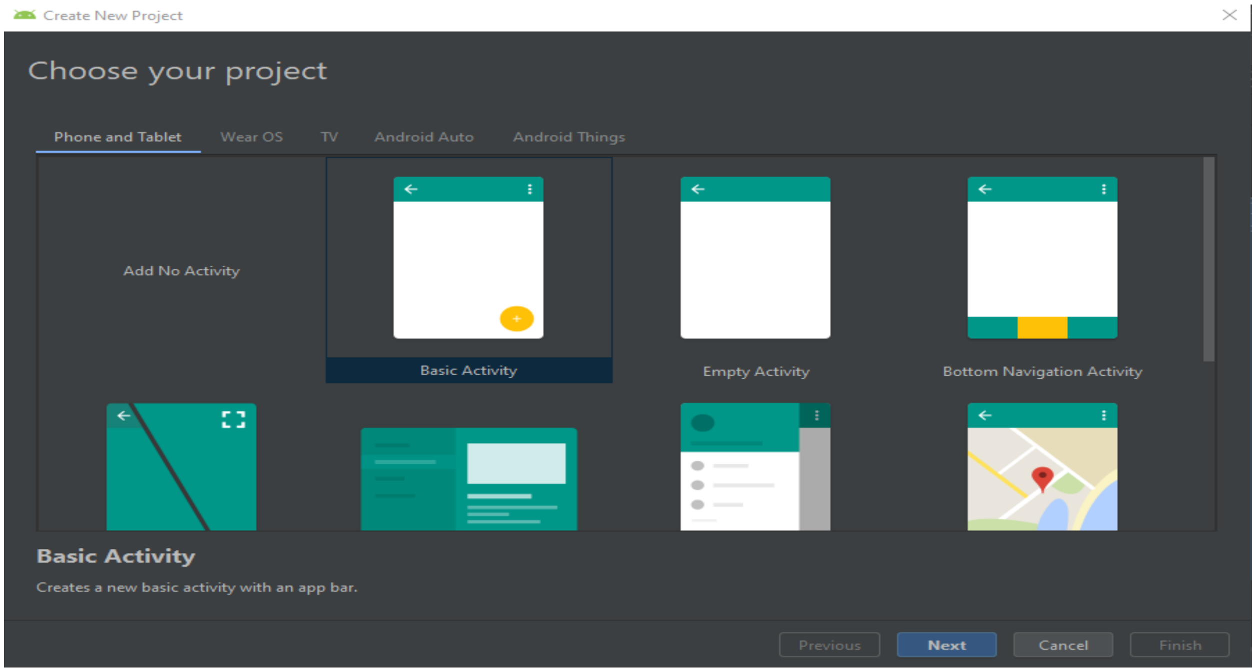
Implementation is the process that turns strategies and plans into actions in order to accomplish strategic objectives and goals.

(OnStrategy, 2019)

This section contains steps and measures taken in the designing and implementation of the application.

STEP 1.

Launching Android Studio application and creating a new android basic activity from **File, New Project and** selecting **Basic Activity** and given it the name **Notepad Application.**

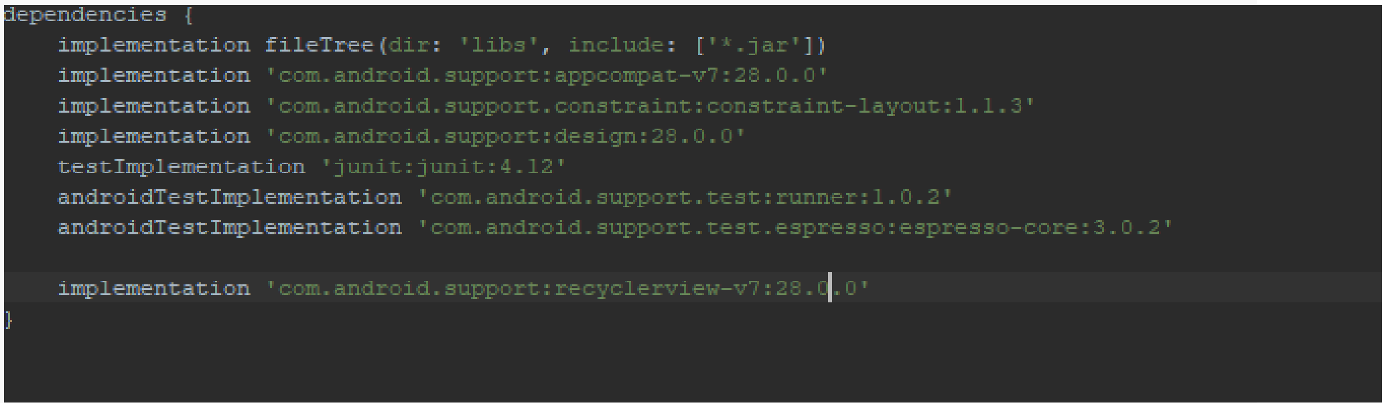


Basic Activity

Creates a new basic activity with an app bar.

STEP 2.

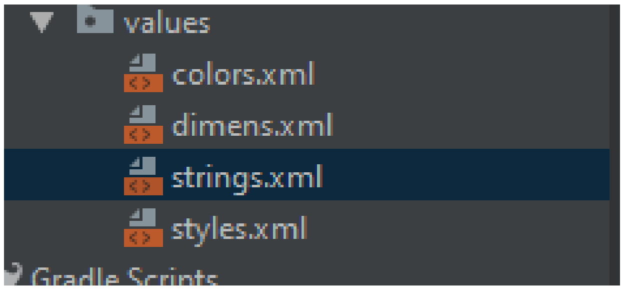
Adding **'com. android. support: recyclerview-v7:28.0.0'** dependence to the build. gradl e in the app directory to be able to use the recycler view in the notepad application. The R ecycler view used to display Notes in a list manner.



```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.android.support:appcompat-v7:28.0.0'
    implementation 'com.android.support.constraint:constraint-layout:1.1.3'
    implementation 'com.android.support:design:28.0.0'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'com.android.support.test:runner:1.0.2'
    androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.2'

    implementation 'com.android.support:recyclerview-v7:28.0.0'
}
```

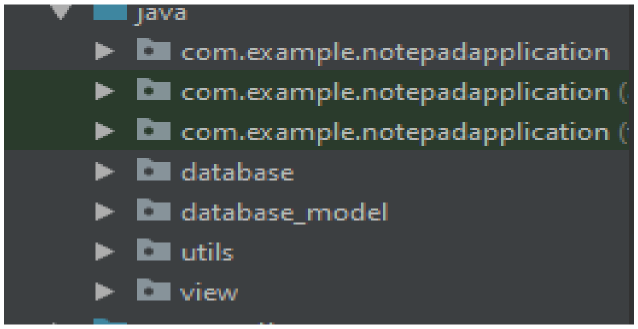
STEP 3.

Adding resource values to the color.xml, string.xml and dimens.xml in the value directory.



STEP 4.

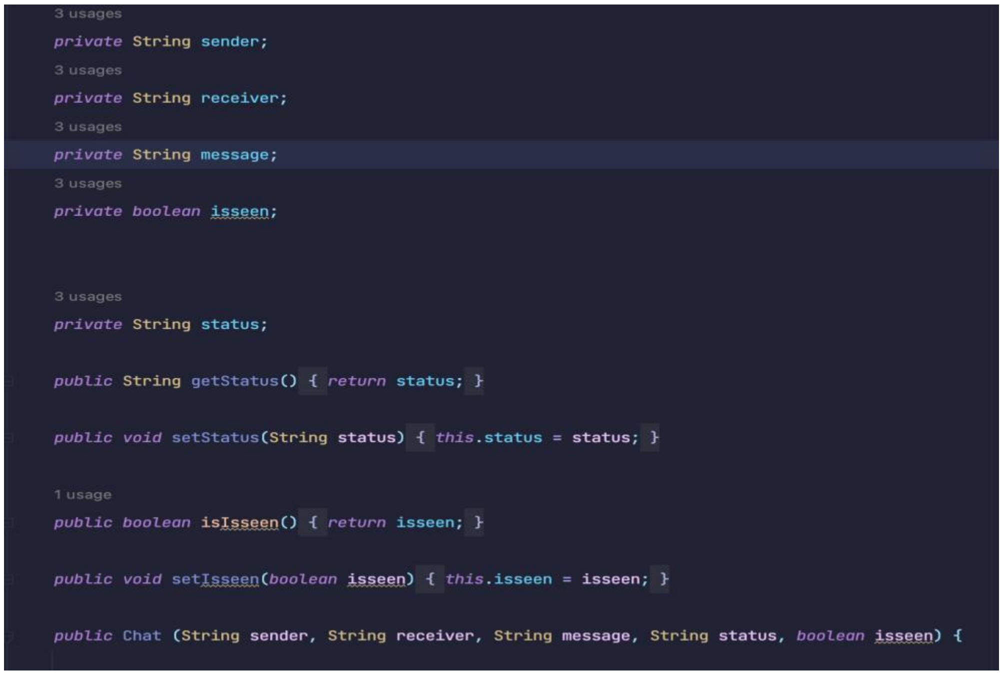
Creating four directories with the names **model**, **utils** and **view** in the java directory.



java
com.example.notepadapplication
com.example.notepadapplication (
com.example.notepadapplication (
database
database_model
utils
view

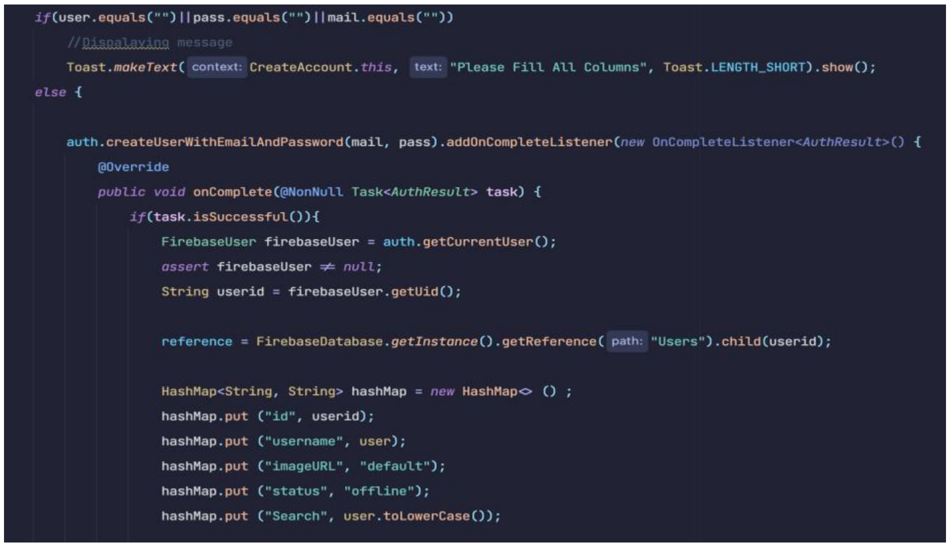
STEP 5

Setting up Getters and setters for chat layout to enhance seamless chatting.


```
    3 usages
    private String sender;
    3 usages
    private String receiver;
    3 usages
    private String message;
    3 usages
    private boolean isseen;


    3 usages
    private String status;

    public String getStatus() { return status; }

    public void setStatus(String status) { this.status = status; }

    1 usage
    public boolean isIsseen() { return isseen; }

    public void setIsseen(boolean isseen) { this.isseen = isseen; }

    public Chat (String sender, String receiver, String message, String status, boolean isseen) {
```

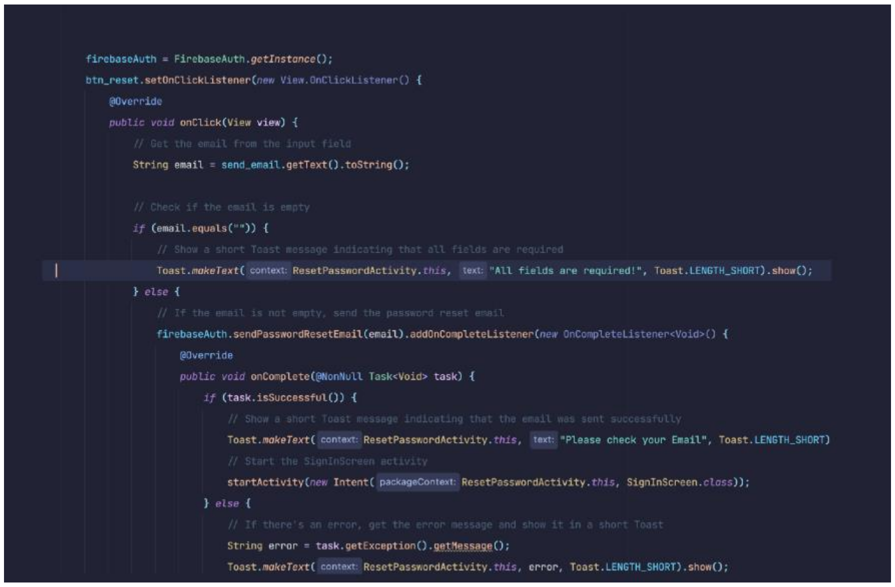
STEP 6

Passing data from textboxes to Firebase Realtime database to create account.


```
if(user.equals("")||pass.equals("")||mail.equals(""))
    //Dispalaying message
    Toast.makeText( context: CreateAccount.this,  text: "Please Fill All Columns", Toast.LENGTH_SHORT).show();
else {

    auth.createUserWithEmailAndPassword(mail, pass).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( path: "Users").child(userid);

                HashMap<String, String> hashMap = new HashMap<> () ;
                hashMap.put ("id", userid);
                hashMap.put ("username", user);
                hashMap.put ("imageURL", "default");
                hashMap.put ("status", "offline");
                hashMap.put ("Search", user.toLowerCase());
```

STEP 7

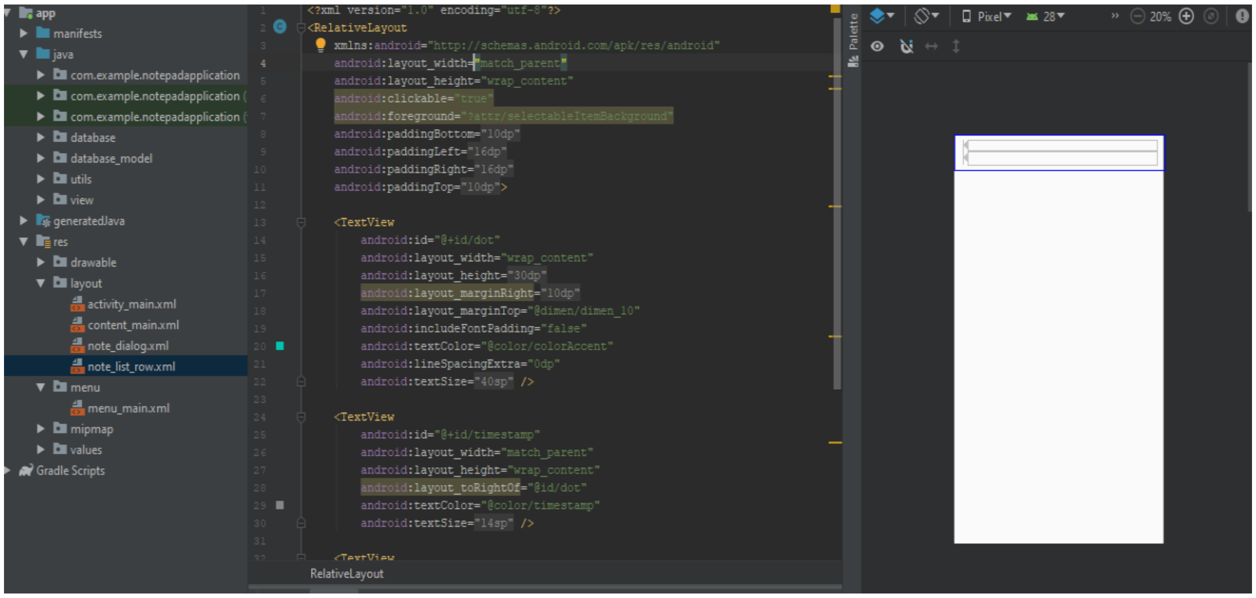
Creating a reset Password button for Users who forgot their passwords



```
firebaseAuth = FirebaseAuth.getInstance();
btn_reset.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        // Get the email from the input field
        String email = send_email.getText().toString();

        // Check if the email is empty
        if (email.equals("")) {
            // Show a short Toast message indicating that all fields are required
            Toast.makeText( context: ResetPasswordActivity.this,  text: "All fields are required!", Toast.LENGTH_SHORT).show();
        } else {
            // If the email is not empty, send the password reset email
            firebaseAuth.sendPasswordResetEmail(email).addOnCompleteListener(new OnCompleteListener<Void>() {
                @Override
                public void onComplete(@NonNull Task<Void> task) {
                    if (task.isSuccessful()) {
                        // Show a short Toast message indicating that the email was sent successfully
                        Toast.makeText( context: ResetPasswordActivity.this,  text: "Please check your Email", Toast.LENGTH_SHORT)
                        // Start the SignInScreen activity
                        startActivity(new Intent( packageContext: ResetPasswordActivity.this, SignInScreen.class));
                    } else {
                        // If there's an error, get the error message and show it in a short Toast
                        String error = task.getException().getMessage();
                        Toast.makeText( context: ResetPasswordActivity.this, error, Toast.LENGTH_SHORT).show();
```

STEP 8

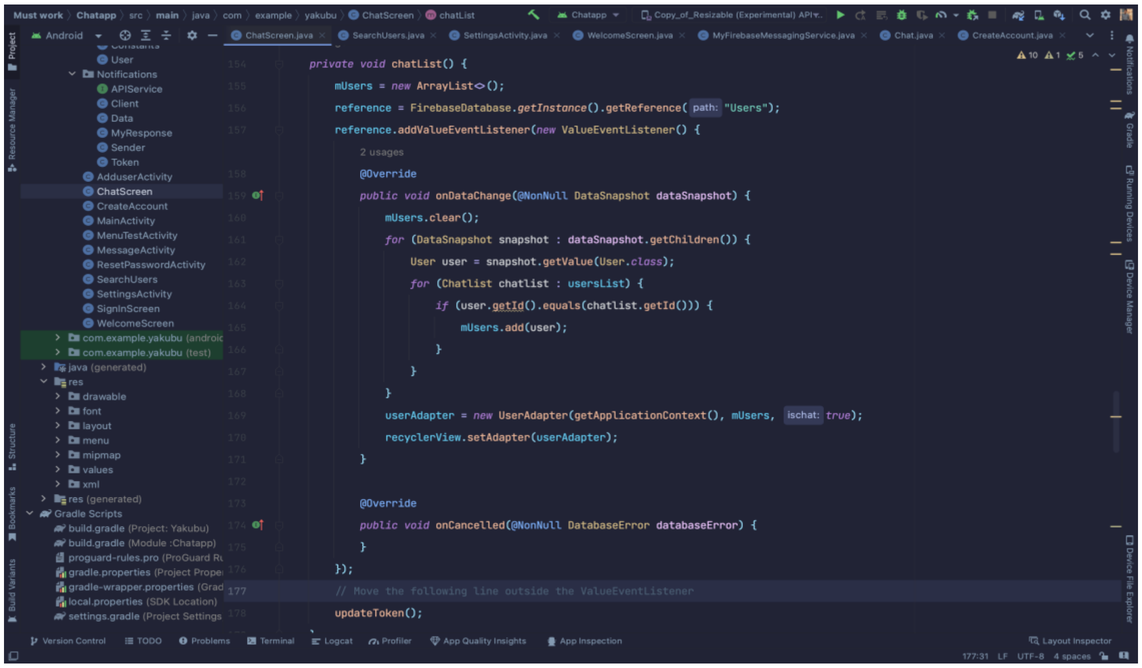
Creating a new XML layout file in the layout directory called **note_list_row.xml** to hold the design of a single note item in the list.



```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:clickable="true"
    android:foreground="?attr/selectableItemBackground"
    android:paddingBottom="10dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="10dp">

    <TextView
        android:id="@+id/dot"
        android:layout_width="wrap_content"
        android:layout_height="30dp"
        android:layout_marginRight="10dp"
        android:layout_marginTop="@dimen/dimen_10"
        android:includeFontPadding="false"
        android:textColor="@color/colorAccent"
        android:lineSpacingExtra="0dp"
        android:textSize="40sp" />

    <TextView
        android:id="@+id/timestamp"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_toRightOf="@id/dot"
        android:textColor="@color/timestamp"
        android:textSize="14sp" />

    <TextView
```

STEP 9

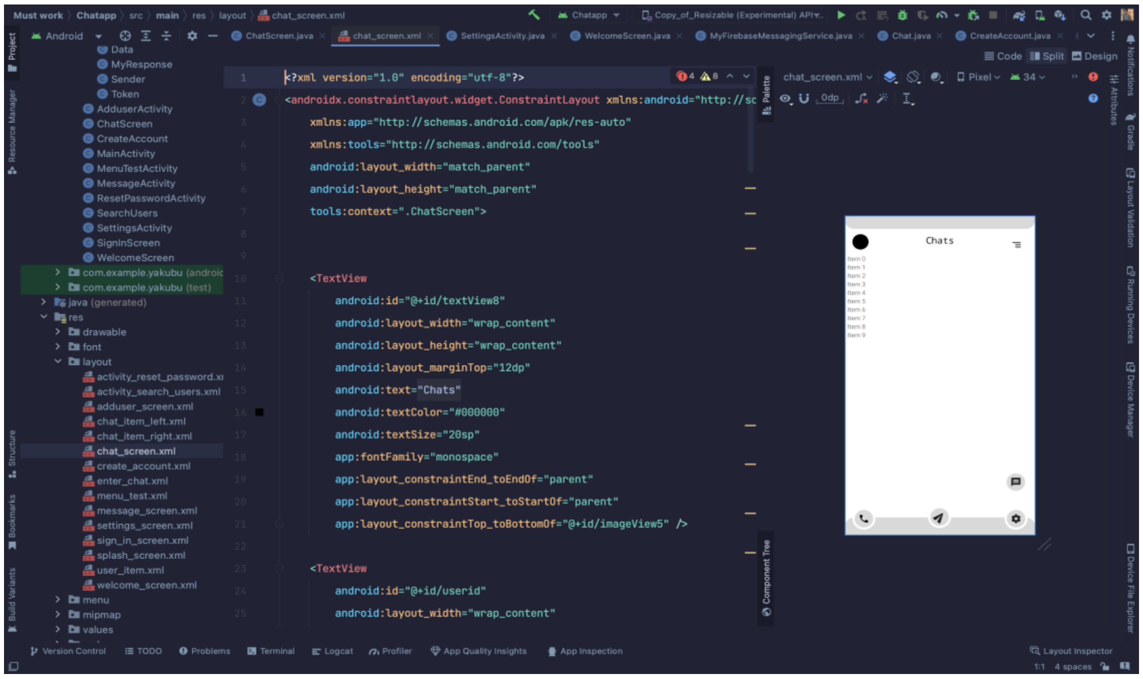
This Updates the Chat list anytime there's a new conversation



```java
private void chatList() {
    mUsers = new ArrayList<>();
    reference = FirebaseDatabase.getInstance().getReference( path: "Users");
    reference.addValueEventListener(new ValueEventListener() {
        2 usages
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            mUsers.clear();
            for (DataSnapshot snapshot : dataSnapshot.getChildren()) {
                User user = snapshot.getValue(User.class);
                for (Chatlist chatlist : usersList) {
                    if (user.getId().equals(chatlist.getId())) {
                        mUsers.add(user);
                    }
                }
            }
            userAdapter = new UserAdapter(getApplicationContext(), mUsers, ischat: true);
            recyclerView.setAdapter(userAdapter);
        }

        @Override
        public void onCancelled(@NonNull DatabaseError databaseError) {
        }
    });
    // Move the following line outside the ValueEventListener
    updateToken();
```

STEP 10

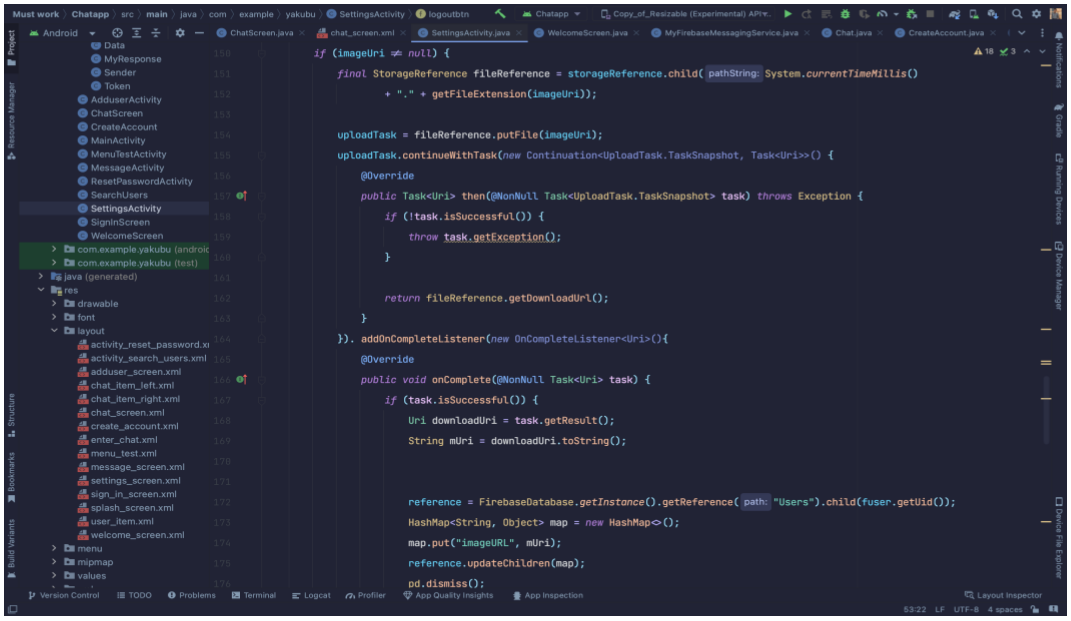
Create XML file for Chat screen layout design.



```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://sc
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".ChatScreen">


    <TextView
        android:id="@+id/textView8"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="12dp"
        android:text="Chats"
        android:textColor="#000000"
        android:textSize="20sp"
        app:fontFamily="monospace"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/imageView5" />

    <TextView
        android:id="@+id/userid"
        android:layout_width="wrap_content"
```

STEP 11

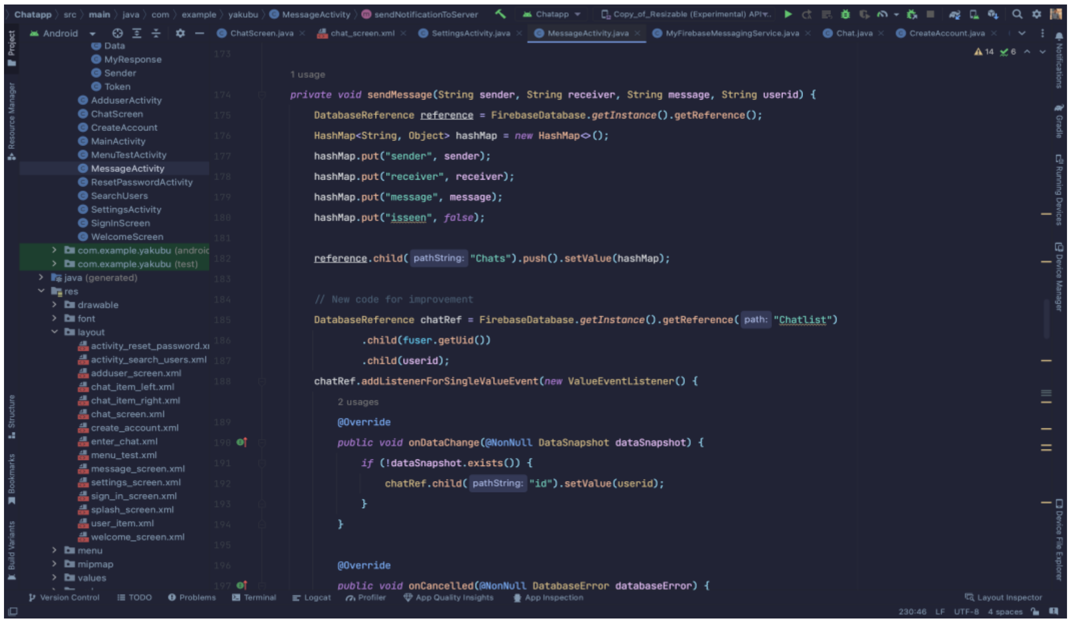
Updating profile picture to preferred image in gallery and saving image to database.



```
        if (imageUri != null) {
            final StorageReference fileReference = storageReference.child( pathString: System.currentTimeMillis()
                    + "." + getFileExtension(imageUri));

            uploadTask = fileReference.putFile(imageUri);
            uploadTask.continueWithTask(new Continuation<UploadTask.TaskSnapshot, Task<Uri>>() {
                @Override
                public Task<Uri> then(@NonNull Task<UploadTask.TaskSnapshot> task) throws Exception {
                    if (!task.isSuccessful()) {
                        throw task.getException();
                    }

                    return fileReference.getDownloadUrl();
                }
            }). addOnCompleteListener(new OnCompleteListener<Uri>(){
                @Override
                public void onComplete(@NonNull Task<Uri> task) {
                    if (task.isSuccessful()) {
                        Uri downloadUri = task.getResult();
                        String mUri = downloadUri.toString();


                        reference = FirebaseDatabase.getInstance().getReference( path: "Users").child(fuser.getUid());
                        HashMap<String, Object> map = new HashMap<>();
                        map.put("imageURL", mUri);
                        reference.updateChildren(map);
                        pd.dismiss();
```

STEP 12

Realtime messaging using firebase between two users.



```
    1 usage
    private void sendMessage(String sender, String receiver, String message, String userid) {
        DatabaseReference reference = FirebaseDatabase.getInstance().getReference();
        HashMap<String, Object> hashMap = new HashMap<>();
        hashMap.put("sender", sender);
        hashMap.put("receiver", receiver);
        hashMap.put("message", message);
        hashMap.put("isseen", false);

        reference.child( pathString: "Chats").push().setValue(hashMap);

        // New code for improvement
        DatabaseReference chatRef = FirebaseDatabase.getInstance().getReference( path: "Chatlist")
                .child(fuser.getUid())
                .child(userid);
        chatRef.addListenerForSingleValueEvent(new ValueEventListener() {
            2 usages
            @Override
            public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                if (!dataSnapshot.exists()) {
                    chatRef.child( pathString: "id").setValue(userid);
                }
            }

            @Override
            public void onCancelled(@NonNull DatabaseError databaseError) {
```

PROJECT TESTING.

The Chatapp application was tested in two phases. The Black Box and White Box testing.

BLACK BOX TESTING

Is a behavior testing of an application which the internal structure, design and implementation of the app tested is unknown to the tester.

(Software Testing Fundermentals, 2019)

TEST NO	FUNCTIONALITY	REQUIRED OUTCOME	EXPECTED OUTCOME	RESULTS
1	Create New Account	User should be able to create new account	Account was created	Passed
2	Login	Login should be possible	User should be logged in	Passed
3	Search for user to Start conversation.	Users should be able to search new users	Users to find existing users	Passed
4	Start Texting in Message	Seamless messaging should be initiated	User should be able to send and receive messages	Passed
5	Logout	Users should be able to logout	User will be logged out from their account.	Passed

Form the table all black box testing was successful and no errors were encountered.

WHITE BOX TESTING

Is the testing method in which the internal structures, design and implementations of the item being tested is known to the tester.

(Software Testing Fundermentals, 2019)

MOBILE DEVICES USED FOR WHITEBOX TESTING

- Samsung Galaxy S10
- Samsung Galaxy SB
- Samsung Galaxy Note 9
- Infinix Note 3
- Infinix Pot 3
- Huawei P20 Lite
- Nokia 7.1
- Android Emulator

USEABILITY

The usability of the Chat application was tested on Samsung Galaxy S10 Plus, Samsung Galaxy SB, Huawei P20 Lite, Samsung Galaxy Note 9, Infinix Note 3, Infinix Pot 3 and android emulator.

Features: All accessible

Structures: All function able

Database: functional

All usability requirements were successful across all tested devices.

COMPATIBILITY

The compatibility of the notepad application was tested on Huawei P20 Lite, Samsung Galaxy S10 Plus and Infinix Note3 and Pot3 smartphone.

Screen Orientation: Compatible (landscape and Portrait)

Database Crosswise: Successful

Platform Compatibility: Successful

Android Versions: Compatible

All compatibility testing was successful across all tested devices.

PERFORMANCE

Performance of the notepad application was tested on Huawei P20 Lite, Samsung Galaxy S10 Plus and Infinix Note3, Pot3 smartphone and Android Emulator.

App Interacting: Successful

Features Functionalities: Compatible

Operation Systems: compatible

Memory Usage: Maximum

Battery Usage: Minimum

App Speed: Maximum

All Performance testing was successful across all tested devices.

GLOSSARY

WORD	MEANING
FAB	Floating Action Button. For performing a actions in an application
CRUD OPERATIONS	Create, Read, Update and Delete are the four basic functions of an application
APP	An application especially as downloaded by a user to a mobile device
COMPATIBLE	Able to exist or occur together without problem or conflict
COMPILE	Produce a result by assembling information collected from other sources
XML	Extensible Markup Language. A layout in android studio
JAVA	A programming language that produces software for multiple platform
PYTHON	A programming Language
KOTLIN	A programming Language
DIRECTORY	A book or container listing individual items or organization with details such as names, address etc.
IDE	Integrated Development Environment a software used for writing and testing of applications or software.
JDK	Java Development Kit a software development environment for java application development.
API	Application Program Interface is a set of protocols and tools for building software application.
RAM	Random Access Memory computer hardware that temporary stores data
EMULATOR	A hardware or software that enables one computer system to behave like another computer system.
LAYOUT	The way in which the part of something are arranged or laid out.

SOURCE CODE

MESSAGE IS DELIVERED OR SEEN

```
private void seenMessage (final String userid){
    reference = FirebaseDatabase. getInstance () .getReference ("Chats") ;
    seenListener = reference.addValueEventListener (new
ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            for (DataSnapshot snapshot : dataSnapshot.getChildren()) {
                Chat chat = snapshot.getValue(Chat.class);
                if (chat.getReceiver().equals(fuser.getUid()) &&
chat.getSender().equals(userid)) {
                    HashMap<String, Object> hashMap = new HashMap<>();
                    hashMap.put("isseen", true);
                    snapshot.getRef().updateChildren(hashMap);
                }
            }
        }

        @Override
        public void onCancelled(@NonNull DatabaseError error) {

        }
    });
}
```

CHECK IF USER IS ONLINE DURING CONVERSATION

```
private void status(String status){
    reference = FirebaseDatabase. getInstance () . getReference
( "Users") .child (fuser.getUid ()) ;
    HashMap<String, Object> hashMap = new HashMap<> () ;
    hashMap.put ("status", status) ;
    reference.updateChildren (hashMap);
}

@Override
protected void onResume(){
    super.onResume();
    status("online");
}

@Override
protected void onPause(){
    super.onPause();
    reference.removeEventListener(seenListener);
    status("offline");
}
```

UPDATE PROFILE IMAGE

```
@Override
    public void onActivityResult (int requestCode, int resultCode, Intent
data){
        super.onActivityResult (requestCode, resultCode, data);

        if (requestCode == IMAGE_REQUEST & resultCode == RESULT_OK
        && data != null & data.getData () != null){
                imageUri = data.getData ();
        if (uploadTask != null && uploadTask.isInProgress()){
                Toast.makeText(SettingsActivity.this,"Upload in
preogress", Toast.LENGTH_SHORT).show();
            }else
            uploadImage () ;
    }
}
```

Users get to change profile image

CONCLUSION

In conclusion, this Android app offers seamless communication, intuitive design, and robust functionality. Leveraging Android Studio and Java programming, it ensures usability, scalability, and a thriving community for support. The app's layout and features, including chat screens and settings, prioritize user experience. Rigorous testing guarantees reliability, making it a valuable tool for modern communication needs.

WAY FORWARD

Moving forward, the app could benefit from continuous updates and enhancements to incorporate emerging technologies and user preferences. Regular maintenance and bug fixes will ensure its smooth operation, while user feedback can guide further improvements. Consider expanding the app's features, such as adding multimedia support or integrating additional communication platforms. Additionally, exploring opportunities for cross-platform compatibility and optimizing performance will contribute to its long-term success and user satisfaction.