

# ACKNOWLEDGMENT

We sincerely owe our gratitude to all people who helped and guided us in completing the project work.

We are thankful to [REDACTED], Principal, [REDACTED] for having supported us in our academic endeavours.

We are thankful to [REDACTED], Head of Department of Computer Science and Engineering, [REDACTED] for providing us timely suggestions, encouragement, and support to complete the mini-project.

We would like to sincerely thank our project guide, [REDACTED], Asst. Professor in Dept. of Computer & Science and Engineering for providing relevant information, valuable guidance, and encouragement to complete the mini-project.

We would also like to thank all our teaching and non-teaching staff members of the Department. We are grateful to the college for keeping labs open whenever required and providing us with Systems and Required software.

We are always thankful to our Parents for their valuable support and guidance in every step. Also, thank all our friends for their support and guidance throughout the project.

We express our deepest gratitude and indebted thanks to [REDACTED] which has provided us an opportunity in fulfilling our most cherished desire of reaching our goal.

Yours Sincerely,

[REDACTED]

[REDACTED]

# **ABSTRACT**

Communication through the internet is becoming vital these days. Instant messaging can be considered as a platform to maintain communication. This system will enable the users to communicate with other users through text messages with the help of the internet. This application is based on Android with the backend as Firebase provided by Google.

The application will have a register page through which the user can register and log in. The home page of the application contains the previous messages if any. The user can be able to search for the other users who have registered in the application. Users can send and receive text messages from other users instantly.

# TABLE OF CONTENT

## ACKNOWLEDGEMENT

## ABSTRACT

## TABLE OF CONTENT

### CHAPTER – I

#### **Introduction -----1**

##### **1.1 Introduction to Kotlin-----1**

##### **1.2 Features of Kotlin-----1**

### CHAPTER – II

#### **Requirement Specification-----2**

##### **2.1 Hardware Requirements-----2**

##### **2.2 Software Requirements-----2**

### CHAPTER – III

#### **About the Project-----3**

##### **3.1 Overview-----3**

##### **3.2 Benefits of the application-----3**

### CHAPTER – IV

#### **System Design-----4**

##### **4.1 List of Components-----4**

### CHAPTER – V

#### **Implementation-----7**

##### **5.1 Kotlin Code-----7**

### CHAPTER – VI

#### **Conclusion-----12**

##### **6.1 Future Enhancements-----12**

### CHAPTER – VII

#### **Snapshots-----13**

### CHAPTER – VIII

#### **Bibliography-----16**

##### **8.1 Web References-----16**

## CHAPTER – I

# INTRODUCTION

### 1.1 Introduction to Kotlin

JetBrains developed IntelliJ IDEA, which is the basis for Android Studio. In 2011, the company introduced the Kotlin language. Kotlin is a statically-typed, modern programming language that runs on a Java Virtual Machine (JVM) by compiling Kotlin code into Java byte-code. It can also be compiled into JavaScript source code and native executables. Kotlin is flexible. Kotlin is an object-oriented language, and a “better language” than Java, but still be fully interoperable with Java code. Though Kotlin was production-ready, the language wasn’t stable. When important changes in the language happened, developers had to change their codebase. Five years later, Kotlin released 1.0. At Google I/O 2017, Google announced that Android will support Kotlin as a first-class programming language from now on. For this to happen, the 3.0 release of Android Studio (AS) integrated Kotlin support out of the box! The following three minor releases of AS continued to improve the Kotlin support and the tools available. Kotlin support for JavaScript (i.e., Classic back-end) is considered stable in Kotlin 1.3 by its developers, while Kotlin/JS (IR-based) in version 1.4, is considered alpha. Kotlin/Native Runtime (e.g., Apple support) is considered beta.

### 1.2 Features of Kotlin

- Statically typed
- Data Classes
- Concise
- Safe
- Interoperable with Java
- Functional and Object-Oriented Capabilities
- Smart Cast
- Compilation time
- Tool-Friendly
- User-Friendly

## CHAPTER – II

# REQUIREMENTS SPECIFICATION

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. It is available for download on Windows, macOS and Linux-based operating systems. The first stable build was released in December 2014. Android Studio provides many excellent features that enhance productivity when building Android apps, such as a blended environment where one can develop for all Android devices, apply changes to push code and resource changes to the running app without restarting the app, a flexible Gradle-based build system, a fast and feature-rich emulator, GitHub and Code template integration to assist you to develop common app features and import sample code, extensive testing tools and frameworks, C++ and NDK support, and many more.

### 2.1 Hardware Requirements

- 64-bit Microsoft Windows 8/10
- x86\_64 CPU architecture; 2nd generation Intel Core or newer, or AMD CPU with support for a Windows Hypervisor.
- 8 GB RAM or more.
- 8 GB of available disk space minimum (IDE + Android SDK + Android Emulator).
- 1280 x 800 minimum screen resolution.

### 2.2 Software Requirements

- Java Development Kit 8 or 11
- Android Studio
- Android SDK
- Java Runtime Environment (JRE) 6
- Android Development Tool kit (ADT kit)
- Android Emulator or a Physical device

## CHAPTER – III

### ABOUT THE PROJECT

#### 3.1 Overview

The purpose of the chat application is to allow users to chat with each other. The users will be able to chat with each other, mostly only from one user to another user. Users can register themselves on to the app or log in if already registered. Users can choose their username and profile image from their local storage. After logging in, the user can see a list of registered users. On tapping any user, they are redirected to a private chat log. Users can also see the latest messages they received on the home page of the application.

The need new way of communicating with each other due to the increase in disruption of privacy by big cooperation's in the world, it has the potential to change our lives unimaginably. And the main importance of our project is since it is an android based chat app it is compatible with most android phones used these days, and already the mobile phone has revolutionized the way we live and communicate, and hence many want to stay in touch with each other.

#### 3.2 Benefits of the application

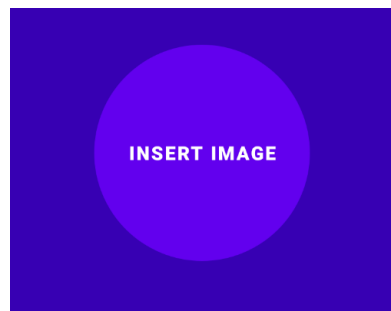
- **One Channel:** You never have to direct conversation between any of the users registered in the application.
- **Privacy:** Users don't have to worry about anyone seeing their private information. The application offers data protection and privacy with no worries.
- **Real-Time Communication:** Chat logs are constantly being updated for new messages; this ensures that the users get their messages as soon as the person on the other side sends the message without any delay in-between.
- **Easy to Access:** The application is effortless to set up and use and can be used on many android phones.
- **Good User Interface:** The application constitutes a satisfactory user interface which is an impressive design and easy-to-use UI. Apart from published user lists, the app provides the user profile image data, captivating the users even further.

## CHAPTER – IV

# SYSTEM DESIGN

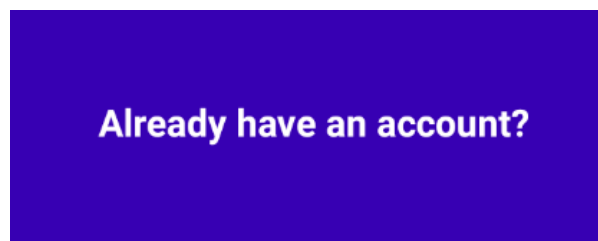
### 4.1 List of Components

- **Button:** Android Button represents a push-button. The `android.widget.Button` is subclass of `TextView` class and `CompoundButton` is the subclass of `Button` class. There are different types of buttons in android such as `RadioButton`, `ToggleButton`, `ImageButton` etc.



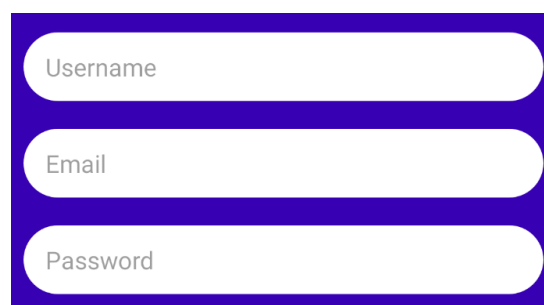
**Fig 4.1 Button**

- **TextView:** A `TextView` displays text to the user and optionally allows them to edit it. A `TextView` is a complete text editor, however the basic class is configured not to allow editing.



**Fig 4.2 TextView**

- **EditText:** A `EditText` is an overlay over `TextView` that configures itself to be editable. It is the predefined subclass of `TextView` that includes rich editing capabilities.



**Fig 4.3 EditText**

- **ImageView:** An Image View is used to display any kind of image in the application. There is also an image button where the component is a button with image on it.



**Fig 4.4 ImageView**

- **Android Option Menus:** are the primary menus of android. They can be used for settings, search, delete item, logout etc.



**Fig 4.5 Menu Options**

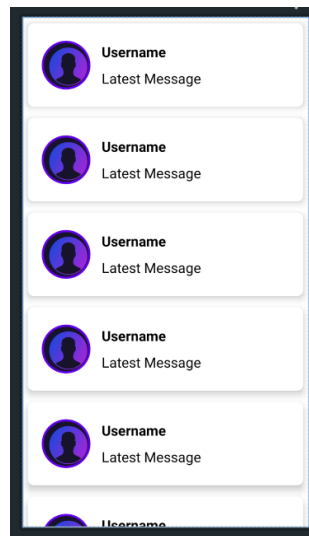
- **RecyclerView:** The RecyclerView class is a more advanced and flexible version of ListView. It is a container for displaying large data sets that can be scrolled very efficiently by maintaining a limited number of views.

To display the data in a RecyclerView, we need the following parts:

- **Data:** It doesn't matter where the data comes from. You can create the data locally, as you do in the practical, get it from a database on the device as you will do in a later practical, or pull it from the cloud.
- **A RecyclerView:** The scrolling list that contains the list items. An instance of RecyclerView as defined in your activity's layout file to act as the container for the views.
- **Layout for one item of data:** All list items look the same, so you can use the same layout for all of them. The item layout has to be created separately from the activity's layout, so that one item view at a time can be created and filled with data.
- **A layout manager:** The layout manager handles the organization (layout) of user interface components in a view. All view groups have layout managers. For the LinearLayout, the Android system handles the layout for you. RecyclerView requires an explicit layout manager to manage the arrangement of list items contained within it. This layout could be vertical, horizontal, or a grid. The layout manager is an instance of Recyclerview.LayoutManager to organize the layout of the items in the RecyclerView



- **A view holder:** The view holder extends the ViewHolder class. It contains the view information for displaying one item from the item's layout
- **An adapter:** The adapter connects your data to the RecyclerView. It prepares the data and how will be displayed in a view holder. When the data changes, the adapter updates the contents of the respective list item view in the RecyclerView. And an adapter is an extension of RecyclerView.Adapter. The adapter uses a ViewHolder to hold the views that constitute each item in the RecyclerView, and to bind the data to be displayed into the views that display it.



**Fig 4.6 RecyclerView**

## CHAPTER – V

# IMPLEMENTATION

These are the following Kotlin files used in the creation of the application.

- RegisterActivity.kt
- LoginActivity.kt
- LatestMessagesActivity.kt
- NewContactActivity.kt
- ChatLogActivity.kt
- UserClass
- ChatMessageClass

### 5.1 Kotlin Code

#### RegisterActivity.kt

```
package com.example.demochat

import android.app.ProgressDialog
import android.content.Intent
import android.net.Uri
import android.os.Bundle
import android.util.Log
import android.widget.*

import androidx.activity.result.contract.ActivityResultContracts
import androidx.appcompat.app.AppCompatActivity
import com.example.demochat.model.UserClass
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.database.FirebaseDatabase
import com.google.firebase.storage.FirebaseStorage
import java.util.*
```

```
class RegisterActivity : AppCompatActivity() {

    private lateinit var username: EditText
    private lateinit var email: EditText
    private lateinit var password: EditText
    private lateinit var register: Button
    private lateinit var haveAccount: TextView
    private lateinit var imageInsert: Button
    private lateinit var imageView: ImageView
    private val tag: String = "RegisterActivity"
    private var selectedPhotoUri: Uri? = null
    private lateinit var dialog: ProgressDialog

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_register)
        supportActionBar?.hide()

        username = findViewById(R.id.editTextUsername)
        email = findViewById(R.id.editTextTextEmailAddress)
        password = findViewById(R.id.editTextTextPassword)
        register = findViewById(R.id.buttonRegister)
        haveAccount = findViewById(R.id.textViewAlready_have_an_account)
        imageInsert = findViewById(R.id.buttonInsertImage)
        imageView = findViewById(R.id.imageViewProfileImage)

        val getImage = registerForActivityResult(
            ActivityResultContracts.GetContent()
        ) {
            selectedPhotoUri = it
            imageView.setImageURI(it)}
    }
```

```
if (selectedPhotoUri != null) {  
    imageInsert.alpha = 0f  
}  
  
imageInsert.setOnClickListener {  
    getImage.launch("image/*")  
}  
  
register.setOnClickListener {  
    dialog = ProgressDialog(this)  
    dialog.setMessage("Registering User")  
    dialog.show()  
    performRegister()  
}  
  
haveAccount.setOnClickListener {  
    val intent = Intent(this@RegisterActivity, LoginActivity::class.java)  
    startActivity(intent)  
}  
}  
} // onCreate  
  
private fun performRegister() {  
    if (email.text.isNullOrEmpty() || password.text.isNullOrEmpty() || username.text.isNullOrEmpty() ||  
    imageView.drawable == null) {  
        Toast.makeText(this, "Fill the above details", Toast.LENGTH_SHORT).show()  
        return  
    }  
  
    FirebaseAuth.getInstance()  
        .createUserWithEmailAndPassword(email.text.toString(), password.text.toString())  
        .addOnCompleteListener {
```

```
if (!it.isSuccessful) return@addOnCompleteListener
    Log.d(tag, "uid = ${it.result?.user?.uid}")

uploadImageToFirebase()
    }
    .addOnFailureListener {
        dialog.dismiss()
        Toast.makeText(this, "${it.message}", Toast.LENGTH_SHORT).show()
        Log.d(tag, "${it.message}")
    }
} //performRegister

private fun uploadImageToFirebase() {
    if (selectedPhotoUri == null) return
    val fileName = UUID.randomUUID().toString() + ".jpg"

    val refStorage = FirebaseStorage.getInstance().reference.child("/images/$fileName")

    refStorage.putFile(selectedPhotoUri!!)
        .addOnSuccessListener {
            Log.d(tag, "image uploaded")

            refStorage.downloadUrl
                .addOnSuccessListener {
                    saveUserToFirebaseDatabase(it.toString())
                }
        }
        .addOnFailureListener {
            dialog.dismiss()
            Log.d(tag, "${it.message}")
        }
}
```

```
        Toast.makeText(this, "${it.message}", Toast.LENGTH_SHORT).show()
    }
}

} //uploadImageToFirebase

private fun saveUserToFirebaseDatabase(profileImageUrl: String) {
    val uid = FirebaseAuth.getInstance().uid ?: ""
    val ref = FirebaseDatabase.getInstance().getReference("/users/$uid")

    val user = UserClass(uid, username.text.toString(), profileImageUrl)
    ref.setValue(user)

    .addOnSuccessListener {
        Log.d(tag, "uploaded to DB")
        dialog.dismiss()
        val intent = Intent(this, LatestMessageActivity::class.java)
        intent.flags =
Intent.FLAG_ACTIVITY_CLEAR_TASK.or(Intent.FLAG_ACTIVITY_NEW_TASK)
        startActivity(intent)
    }

    .addOnFailureListener {
        dialog.dismiss()
        Toast.makeText(this, "${it.message}", Toast.LENGTH_SHORT).show()
        Log.d(tag, "${it.message}")
    }
}

} //saveUserToFirebaseDatabase

} //RegisterActivity
```

## **CHAPTER – VI**

### **CONCLUSION**

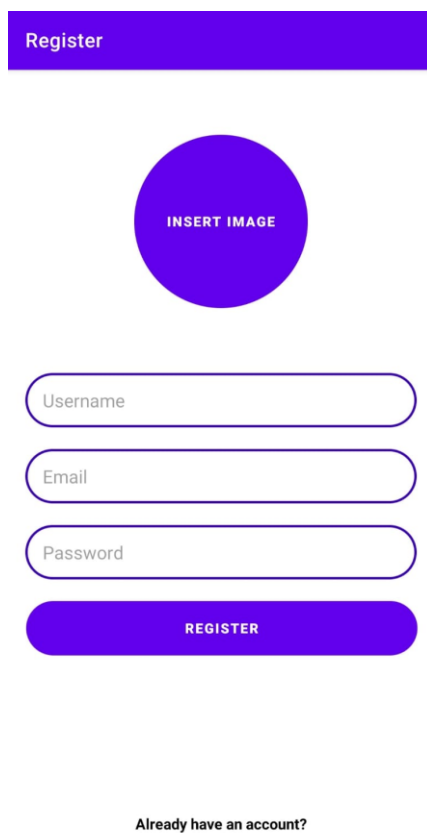
There is always room for improvements in any apps. Right at the moment, we are just dealing with text communication. Several Android apps serve a similar purpose as this project, but these apps were a bit difficult to use and provide confusing interfaces. A positive first impression is essential in a human relationship as well as in human-computer interaction. This project hopes to develop a chat service Android app with a high-quality user interface.

#### **6.1 Future Enhancement**

- File Transfer
- Voice Message
- Group Message
- Profile Customization
- Video Message

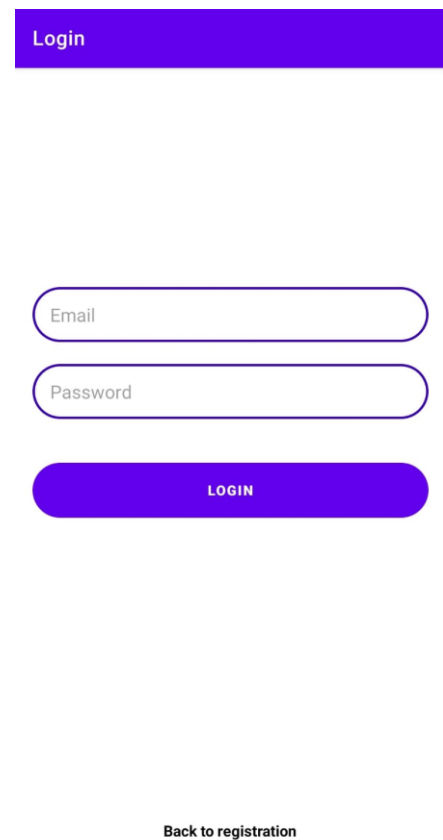
## CHAPTER – VII

### SNAPSHOTS



The screenshot shows the 'Register' screen of a chat application. At the top, there is a blue header bar with the word 'Register' in white. Below the header, there is a large blue circle with the text 'INSERT IMAGE' in white. Underneath the circle, there are three white input fields with blue borders, labeled 'Username', 'Email', and 'Password'. Below these fields is a large blue button with the word 'REGISTER' in white. At the bottom of the screen, there is a link that says 'Already have an account?'.

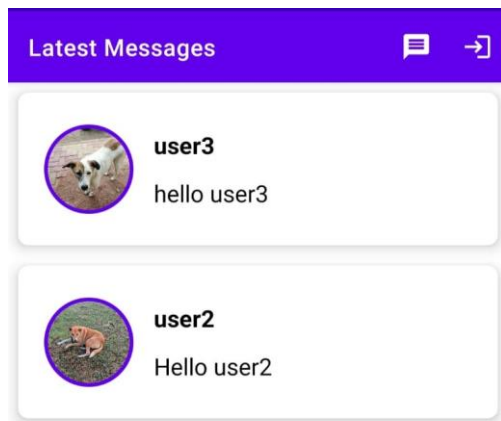
**Fig 7.1 Register Activity**



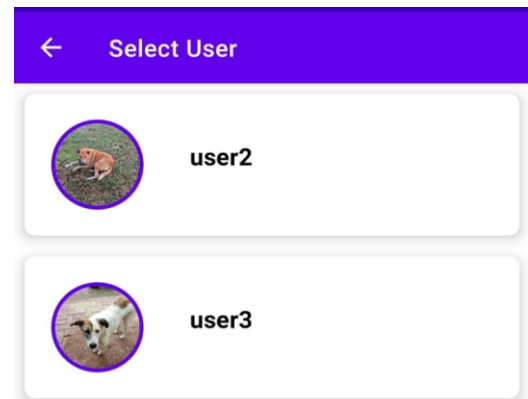
The screenshot shows the 'Login' screen of a chat application. At the top, there is a blue header bar with the word 'Login' in white. Below the header, there are two white input fields with blue borders, labeled 'Email' and 'Password'. Below these fields is a large blue button with the word 'LOGIN' in white. At the bottom of the screen, there is a link that says 'Back to registration'.

**Fig 7.2 Login Activity**





**Fig 7.3 Latest Message Activity**



**Fig 7.4 List of Users**



**Fig 7.5 Chat log Activity**

## CHAPTER – VIII

### BIBLIOGRAPHY

- Google Developer Training, "Android Developer Fundamentals Course – Concept Reference", Google Developer Training Team, 2017.

#### 8.1 Web References:

- [www.google.com](http://www.google.com)
- [www.youtube.com](http://www.youtube.com)
- [www.github.com](http://www.github.com)
- [www.letsbuildthatapp.com](http://www.letsbuildthatapp.com)
- [www.stackoverflow.com](http://www.stackoverflow.com)