ChatConnect - A Real Time Chat and Communication App

1. Introduction

1.1 Project Overview

Initially, an UI for user registration and login. Users should be able to create accounts and sign in securely. This system should ensure the confidentiality and integrity of user data. Here for storing the data, we set up a firebase realtime database to store user information and messages. The data will be stored in JSON. The backend development involves creating a server-side application that manages user authentication, message storage, and delivery using APIs. We build an user interface for message composition and interaction. The interface should enable users to compose and send messages, view message history, and interact seamlessly within the app. We also build an user interface for contacts where user can see his recently chatted people, thus user don't need to enter receiver's username everytime he message them. Our system safeguard user credentials, data transmission, and prevent unauthorized access. We thoroughly test the app for functionality and debugging to identify and resolve any bugs or issues.

1.2 Purpose

Users can exchange messages in real-time, allowing quick and direct communication. The app fosters connections between individuals, enabling conversations, sharing information, and maintaining contact. It provides an easy and accessible platform for communication, allowing users to interact from anywhere, anytime. Users can also engage in group conversations, share media, and collaborate on various projects or discussions. It provides an easily accessible platform for communication, allowing users to connect from various devices and locations. It serves both personal and professional needs, from casual conversations to professional communication and team collaboration.

2. Literature Survey

2.1 Existing Problem

- The existence of numerous chat apps with varying features and platforms results in interoperability issues. Users often have to manage multiple apps to communicate across different platforms, leading to fragmentation and inconvenience.
- Handling a large volume of messages and multimedia files raises concerns about efficient data storage and management, especially in apps where data retention is a critical factor.
- Maintaining consistent real-time communication without delays or message loss remains a technical challenge, especially as user bases grow.
- Some apps face difficulties in providing a seamless, intuitive, and user-friendly interface, affecting ease of use and overall satisfaction.

2.2 References

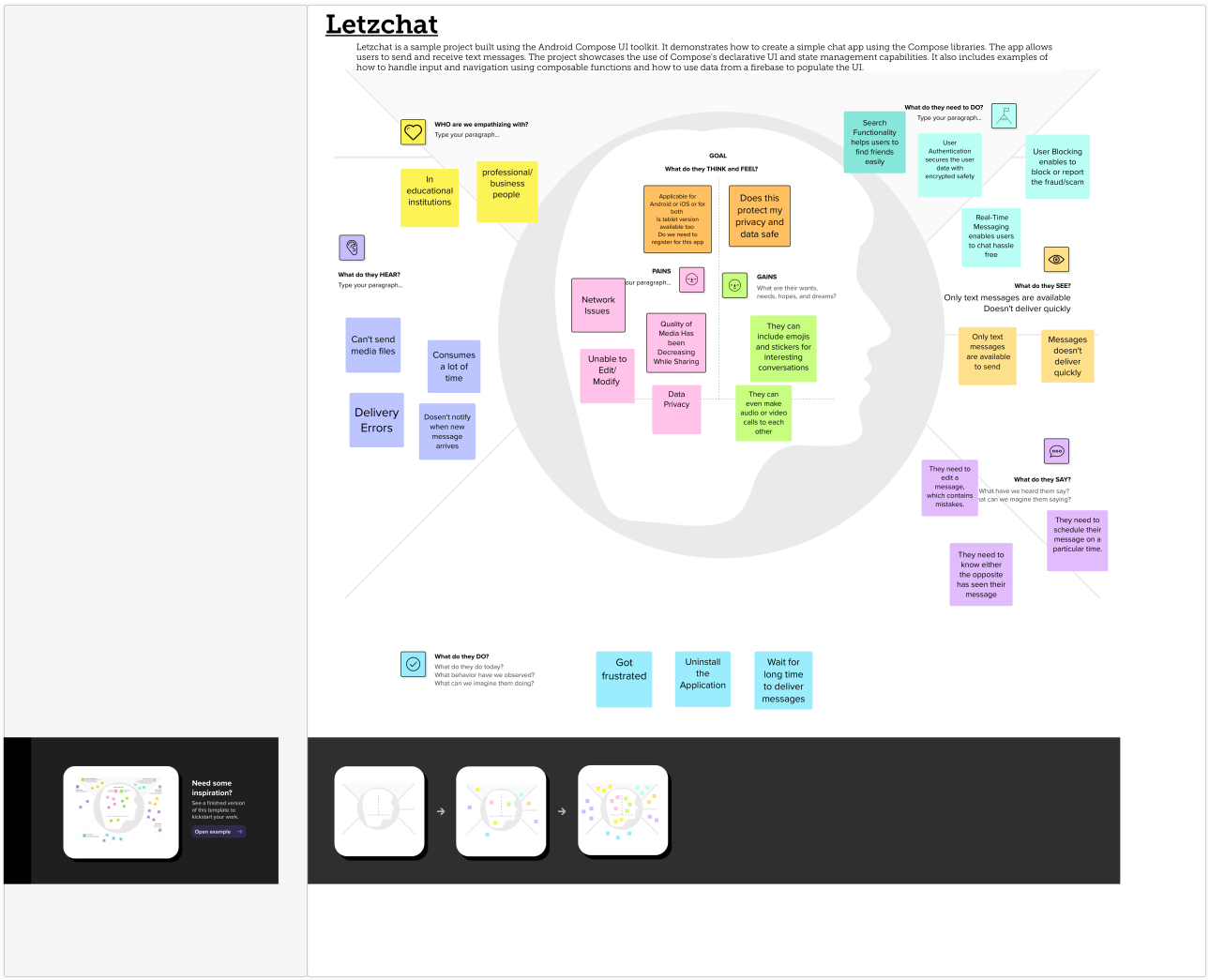
- "*Web-ChatLine: An Innovative Chatting Platform*" By Amit Kumar Goel, year 2022.
- "*Enhanced Chat Application*" By Avinash Bamane, year 2012.
- "*Reporting with WhatsApp: Mobile Chat Applications' Impact on Journalistic Practices*" By Tomás Dodds, year 2019.
- "*Social Network Chatting Apps Network Traffic Optimization*" By Pinial Khan Butt, year 2018.

2.3 Problem Statement Definitions

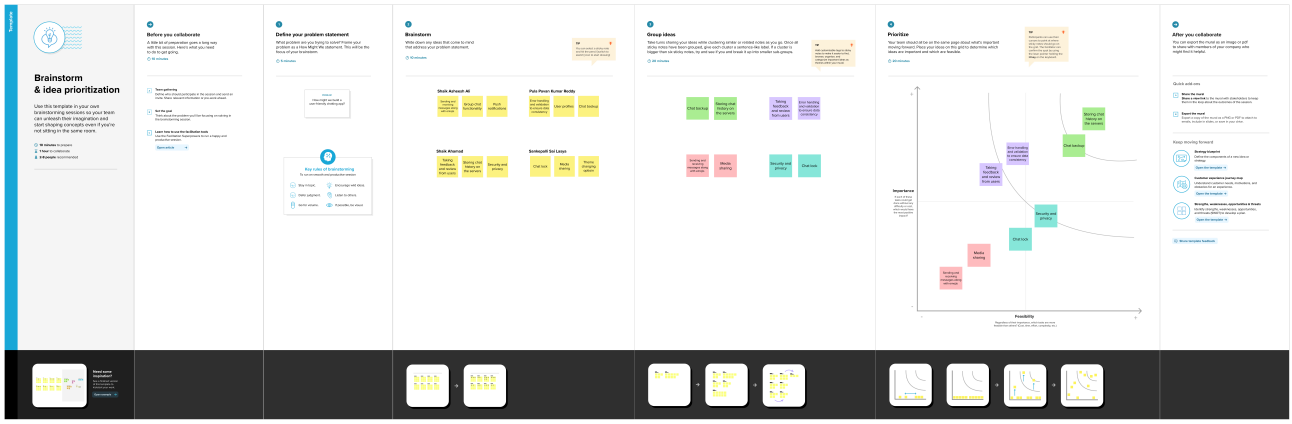
Design and develop a basic chatting application that enables real-time text messaging between users. The application should provide a user-friendly interface for mobile platform with some core features like User Authentication, Theme Preferences and Password changing facility.

3. Ideations and Proposed Solutions

3.1 Empathy Map Canvas



3.2 Ideation and Brainstorming



4. Requirement Analysis

4.1 Functional Requirement

1. User Registration and Authentication
   - Users should be able to create accounts with a unique username and password.
   - The app should have a secure authentication process to protect user accounts.
2. Contact Management
   - Users should be able to view contacts.
   - The app should provide a feature to search for and send friend requests.
3. Real-time Messaging
   - Users should be able to send text messages in real-time.
4. Message History
   - The app should store and display chat history.

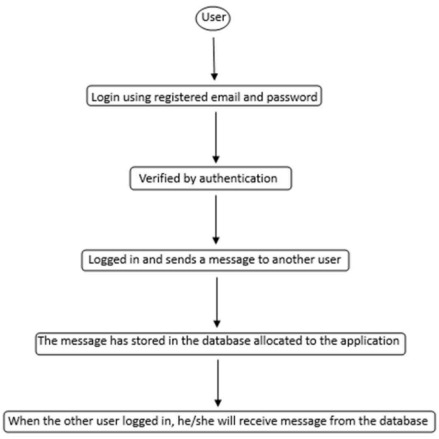
4.2 Non-Functional Requirements

1. Performance
   - The app should have low latency in delivering messages.
   - It should be able to handle a large number of concurrent users.
2. Scalability
   - The system should be able to scale seamlessly as the user base grows.
   - It should support additional users and features without significant performance degradation.
3. Reliability
   - The app should have high availability, minimizing downtime.
   - It should recover gracefully from failures and ensure data integrity.
4. Usability
   - The user interface should be intuitive and user-friendly.

5. Project Design

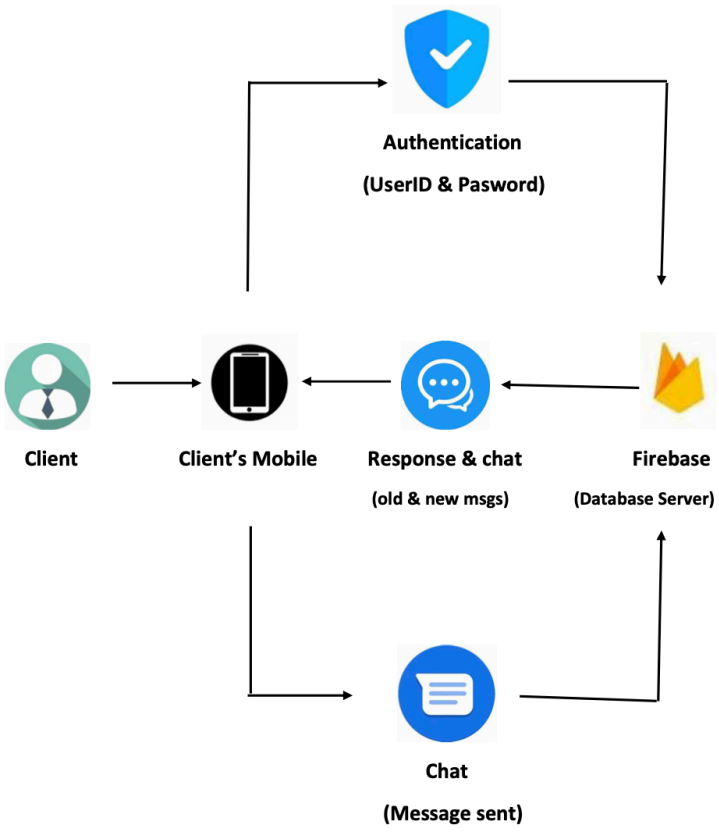
5.1 Data Flow Diagrams & User Stories

1. User login into the application using the registered credentials.
2. The credentials are verified using the authentication and allows if they are valid.
3. The user has the access to message to another user.
4. The message is stored in the firebase database and when another user logged in, then he/she will receive the message.
5. The message is stored in the database until the other user will receive the message.



User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Mobile user)	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	I can access my account / dashboard	High	Sprint-1
		USN-2	As a user, I will receive confirmation email once I have registered for the application	I can receive confirmation email & click confirm	High	Sprint-1
		USN-3	As a user, I can register for the application through Gmail	I can register & access the dashboard with Gmail Login	High	Sprint-1
	Login	USN-4	As a user, I can log into the application by entering email & password	I can login into the application using my registered email and password	High	Sprint-1
	Dashboard	USN-5	As a user, I can select one contact and can message them.	I can select a contact and can message them	Medium	Sprint-1
			I send a message to other person present in the list of the application.	I send a message to another user.	Medium	Sprint-1
Customer (Mobile user)	Login	USN-6	As a user, I can log into the application by entering my registered email and password.	I can login into the application using my registered email and password	High	Sprint-2
	Dashboard	USN-7	As a user, I received the message send by the other user.	I can receive the message sent by the other user to me.	Medium	Sprint-2
Administrator		ASN-1	As an administrator, I allocate the firebase database to store the user email and password.	I allocate the database to store the user's data.	High	Sprint-3
		ASN-2	As an administrator, I allocate the data base to store the message history between the users.	I allocate the database to store the chat history between the users.	High	Sprint-3

5.2 Solution Architecture

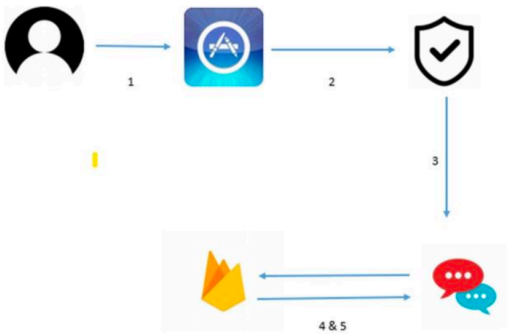


6. Project Planning and Scheduling

6.1 Technical Architecture

Technical Architecture:

The Deliverable shall include the architectural diagram as below and the information as per the table1 & table 2



Guidelines:

1. User login into the application using the registered credentials.
2. The credentials are verified using the authentication and allows if they are valid.
3. The user has the access to message to another user.
4. The message is stored in the firebase database and when another user logged in, then he/she will receive the message.
5. The message is stored in the database until the other user will receive the messa

6.2 Sprint Planning & Estimation

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	2	High	3
Sprint-2		USN-2	As a user, I will receive confirmation email once I have registered for the application	1	High	2
Sprint-1		USN-3	As a user, I can register for the application through gmail	1	high	1
Sprint-1	Login	USN-4	As a user, I can log into the application by entering email & password	1	High	2
Sprint-2	Dashboard	USN-5	As a user, I can select one contact and can message them	1	medium	3
Sprint-1	login	USN-6	As a user, I can log into the application by entering my registered email and password	1	high	2
Sprint-4	Dashboard	USN-7	As a user, I received the message send by the other user	1	medium	3

6.3 Sprint Delivery Schedule

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	20	6 Days	6 Oct 2023	11 Oct 2023	20	11 Oct 2023
Sprint-2	20	6 Days	11 Oct 2023	16 Oct 2023	20	16 Oct 2023
Sprint-3	20	6 Days	18 Oct 2023	23 Oct 2023	20	23 Oct 2023
Sprint-4	20	6 Days	25 Oct 2023	30 Oct 2023	20	30 Oct 2023

7. Coding and Solutioning

7.1 Change Theme

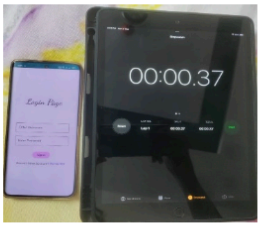
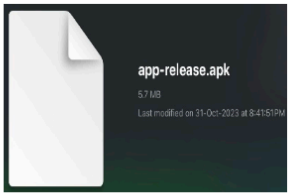
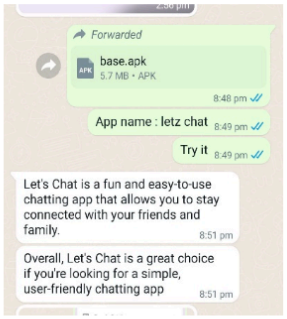
```
//Default Theme
var Theme = “light"

fun light() {
    Theme = "light"
    primary = Color(0xFFFFF0FF)
    primary2 = Color(0xFFFFE0FF)
    secondary = Color(0xFFFFC0FF)
    secondary2 = Color(0xFFFFA0FF)
    textcolor = Color.Black
}

fun dark() {
    Theme = "dark"
    primary = Color(0xFFCC80CC)
    primary2 = Color(0xFFDDA0DD)
    secondary = Color(0xFF883088)
    secondary2 = Color(0xFFAA40AA)
    textcolor = Color.White
}

//Onclick listener
if(Theme == "light")
    dark()
else
    light()
```

7.2 Internal Sharing

```
val file = File(context.applicationInfo.sourceDir)
val intent = Intent(Intent.ACTION_SEND)
intent.type = "application/vnd.android.package-archive"
intent.putExtra(Intent.EXTRA_STREAM, Uri.fromFile(file))
context.startActivity(Intent.createChooser(intent, "Share via”)
            .addFlags(Intent.FLAG_ACTIVITY_NEW_TASK))
```

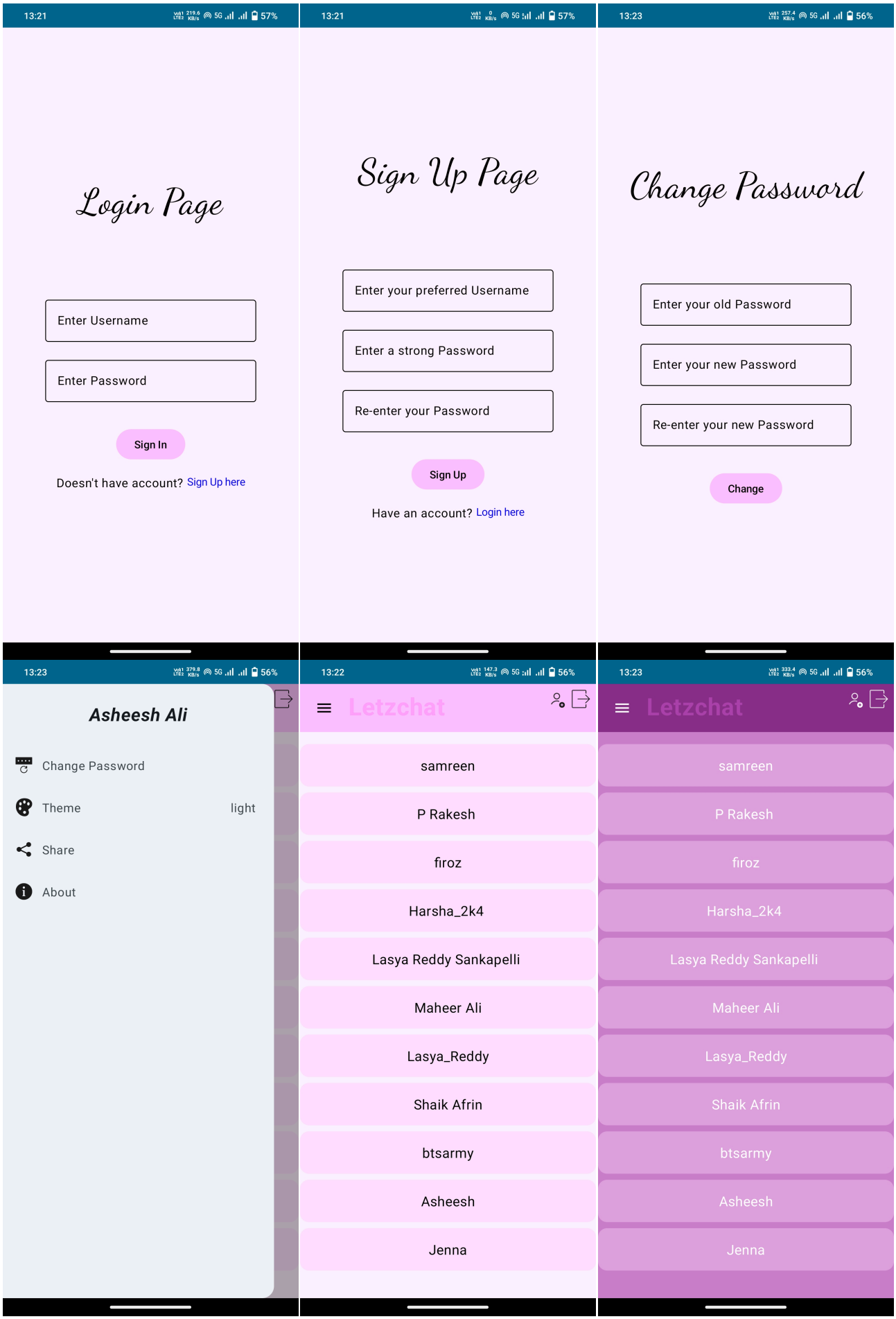
8. Performance Testing

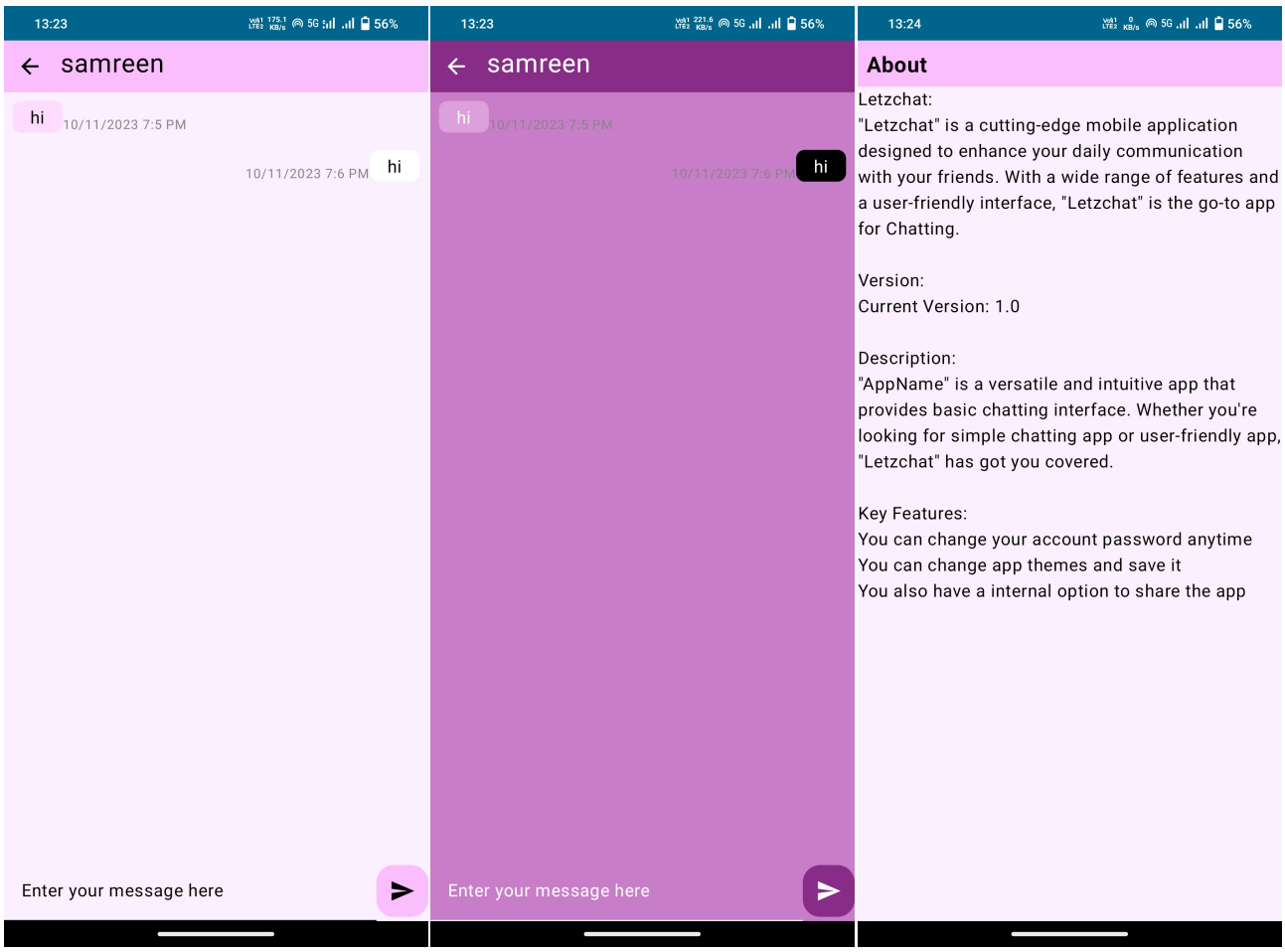
8.1 Performance Metrics

S.No.	Parameter	Values	Screenshot
	Metrics	App Launch Time-37ms Screen Render Time-40ms(130-135fps) Code Quality- the code quality of lets chat app is good, with proper use of coding standards. The app uses various frameworks To ensure the app runs efficiently and reliably.	 App launch time  Screen rendering time
	Usage	App Size-5.7mb Customer Experience-	  
	Performance	Error and Crash Rates-0 Database Query Performance- not applicable	none

9. Results

9.1 Output Screenshots





10. Advantages and Disadvantages

Pros:

1. Instant Communication
2. Global Connectivity
3. User Engagement
4. Access Anytime & Anywhere
5. Cost-Effective

Cons:

1. Privacy Concerns
2. Distraction and Overuse
3. Technical Issues
4. Security Risks
5. Dependency on Internet Connectivity

11. Conclusion

In conclusion, a chatting app offers numerous advantages in terms of instant communication, global connectivity, and cost-effectiveness. Users can engage in real-time conversations, fostering a dynamic and interactive communication experience. The accessibility of these apps across various devices enables users to stay connected anytime and anywhere.

However, it's essential to be mindful of the potential disadvantages associated with chatting apps. Privacy concerns, dependency on internet connectivity, and the risk of security issues underscore the need for robust measures to protect user data. Additionally, the potential for miscommunication and the lack of personal interaction can impact the quality of communication.

As technology continues to evolve, we(developers) shall focus on addressing these challenges through enhanced security features, improved user education, and thoughtful design that prioritizes both functionality and user experience. Ultimately, a well-designed basic chatting app has the potential to greatly enhance communication and connectivity, provided that it aligns with user expectations and maintains a balance between convenience and security.

12. Future Scope

The future scope for a chatting app is dynamic and expansive, driven by technological advancements and evolving user expectations. Here are several potential areas of development and enhancement for the future of chatting apps:

1. Cross-Platform Compatibility
   Further improvements in cross-platform compatibility will enable users to seamlessly switch between devices while maintaining a consistent and integrated experience.
2. Enhanced Security Measures
   As privacy concerns grow, there will likely be a continued focus on implementing robust security measures, such as end-to-end encryption and advanced authentication methods.
3. Smart Notifications and Filters
   Implementing intelligent notification systems and message filters can help users manage and prioritize their communication effectively.
4. Collaborative Tools
   Enhanced collaboration tools within chat apps can facilitate group projects, document sharing, and real-time collaboration, making them more than just communication tools.
5. Gesture and Voice Controls
   Integrating gesture and voice controls can provide a hands-free and intuitive interface, especially important in scenarios where users may not be able to use traditional input methods.
6. Personalization and Context Awareness
   Advanced personalization features and context-awareness can enable the app to adapt to users' preferences, making communication more tailored and efficient.

13. Appendix

Source Code

MainActivity.kt:

```
package com.example.letschat
```

```
import android.content.Context
import android.os.Bundle
```

```
import android.os.StrictMode
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.graphics.painter.Painter
import com.example.letschat.ui.theme.LetsChatTheme
import com.google.firebase.database.ktx.database
import com.google.firebase.ktx.Firebase
import androidx.navigation.compose.NavHost
import androidx.navigation.compose.composable
import androidx.navigation.compose.rememberNavController
import com.google.firebase.database.DataSnapshot
import com.google.firebase.database.DatabaseError
import com.google.firebase.database.DatabaseReference
import com.google.firebase.database.ValueEventListener
import com.google.firebase.database.ktx.getValue

val database = Firebase.database
var uname: String = ""
var myRef: DatabaseReference = database.getReference(uname)
var othersRef: DatabaseReference = database.getReference("")
var value: MutableList<messagedetails>? =
mutableListOf(messagedetails())
var bgvalue: MutableList<messagedetails>? =
mutableListOf(messagedetails())

var Theme = "light"
```

```
var primary = Color(0xFFFFF0FF)
var primary2 = Color(0xFFFFE0FF)
var secondary = Color(0xFFFFC0FF)
var secondary2 = Color(0xFFFFA0FF)
var textcolor = Color.Black

fun light() {
    Theme = "light"
    primary = Color(0xFFFFF0FF)
    primary2 = Color(0xFFFFE0FF)
    secondary = Color(0xFFFFC0FF)
    secondary2 = Color(0xFFFFA0FF)
    textcolor = Color.Black
}

fun dark() {
    Theme = "dark"
    primary = Color(0xFFCC80CC)
    primary2 = Color(0xFFDDA0DD)
    secondary = Color(0xFF883088)
    secondary2 = Color(0xFFAA40AA)
    textcolor = Color.White
}

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
```

```
        val context: Context = applicationContext
        val builder = StrictMode.VmPolicy.Builder()
        StrictMode.setVmPolicy(builder.build())
        builder.detectFileUriExposure()
        setContent {
            LetsChatTheme {
                val navController = rememberNavController()
                NavHost(
                    navController = navController,
                    startDestination = if(uname!="")
"contacts" else "login"
                ) {
                    composable("login") {
                        Login(navController)
                    }
                    composable("contacts") {
                        Contacts(navController, context)
                    }
                    composable("chat") {
                        Chat(navController)
                    }
                    composable("signup") {
                        Signup(navController)
                    }
                    composable("chpass") {
                        ChPass(navController)
                    }
```

```
                    composable("about") {
                        About(navController)
                    }
                }
            }
        }
    }
    override fun onStart() {
        super.onStart()
        val sharedPreferences =
getSharedPreferences("my_app_prefs", Context.MODE_PRIVATE)
        uname =
sharedPreferences.getString("uname","").toString()
        Theme =
sharedPreferences.getString("theme","light").toString()
        if(Theme == "light")
            light()
        else
            dark()
        if(uname!="") {
            myRef = database.getReference(uname)
            myRef.addValueEventListener(object :
ValueEventListener {

                override fun onDataChange(snapshot:
DataSnapshot) {
                    value =
snapshot.getValue<MutableList<messagedetails>>()
```

```
                }

                override fun onCancelled(error:
DatabaseError) {
                    println("Not yet implemented")
                }
            })
        }
    }

    override fun onStop() {
        super.onStop()
        val sharedPreferences =
getSharedPreferences("my_app_prefs", Context.MODE_PRIVATE)
        val editor = sharedPreferences.edit()
        editor.putString("uname", uname)
        editor.putString("theme", Theme)
        editor.apply()
    }
}

data class date(val year: Int = 0, val month: Int = 0, val
day: Int = 0, val hour: Int = 0, val minute: Int = 0, val
second: Int = 0, val meridiem: String = "")
data class messagedetails(val from: String = "", val to:
String = "", val date: date = date(), val message: String =
"", var pass: String = "")
data class navigationItem(val title:String, val icon:
Painter, val event: ()->Unit, val badge: String = "")
```

login.kt:

```
package com.example.letschat

import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.text.ClickableText
import androidx.compose.foundation.text.KeyboardActions
import androidx.compose.foundation.text.KeyboardOptions
import androidx.compose.material3.Button
import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.LocalTextStyle
import androidx.compose.material3.OutlinedTextField
import androidx.compose.material3.Text
import androidx.compose.material3.TextFieldDefaults
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
```

```
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.text.AnnotatedString
import androidx.compose.ui.text.SpanStyle
import androidx.compose.ui.text.font.FontFamily
import androidx.compose.ui.text.input.ImeAction
import androidx.compose.ui.text.input.KeyboardType
import
androidx.compose.ui.text.input.PasswordVisualTransformation
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.navigation.NavHostController
import com.google.firebase.database.DataSnapshot
import com.google.firebase.database.DatabaseError
import com.google.firebase.database.ValueEventListener
import com.google.firebase.database.ktx.getValue
```

```
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun Login(navController: NavHostController) {
    if(uname!="")
        navController.navigate("contacts")
    Column(modifier = Modifier
        .fillMaxSize()
        .background(color = primary), horizontalAlignment =
Alignment.CenterHorizontally, verticalArrangement =
Arrangement.Center) {
```

```
        Text(text = "Login Page", fontFamily =
FontFamily.Cursive, fontSize = 48.sp, color = textcolor)
        Spacer(modifier = Modifier.padding(32.dp))
        var text by remember {
            mutableStateOf("")
        }
        Text(text = text, color = Color.Red)
        Spacer(modifier = Modifier.padding(4.dp))
        var name by remember {
            mutableStateOf("")
        }
        OutlinedTextField(value = name, onValueChange =
{ name = it }, label = { Text(text = "Enter Username", color
= textcolor)}, keyboardOptions = KeyboardOptions(imeAction =
ImeAction.Next), textStyle =
LocalTextStyle.current.copy(textcolor), colors =
TextFieldDefaults.outlinedTextFieldColors(focusedBorderColor
= textcolor, unfocusedBorderColor = textcolor))
        Spacer(modifier = Modifier.padding(8.dp))
        var pass by remember {
            mutableStateOf("")
        }
        var checking by remember {
            mutableStateOf(false)
        }
        var i = false
        fun checking(name: String, pass: String) {
            myRef = database.getReference(name)
```

```
            myRef.addValueEventListener(object :
ValueEventListener {

                override fun onDataChange(snapshot:
DataSnapshot) {
                    value =
snapshot.getValue<MutableList<messagedetails>>()
                    if (i) {
                        if (value != null) {
                            if (value?.get(0)?.pass == pass)
{
                                uname = name

navController.navigate("contacts")
                            } else
                                text = "Username or Password
is doesn't match"
                        } else
                            text = "Username or Password is
doesn't match"
                        i = false
                    }
                    checking = false
                }

                override fun onCancelled(error:
DatabaseError) {
                    println("Not yet implemented")
                }
```

```
            })
            checking = true
        }
        fun go() {
            text = ""
            if(!checking) {
                if (name.length >= 3) {
                    if (pass.length >= 8) {
                        i = true
                        checking(name, pass)
                    } else
                        text = "Password must contain at
least 8 characters"
                } else
                    text = "Username must contain at least 3
characters"
                name = ""
                pass = ""
            }
        }
        OutlinedTextField(value = pass, onValueChange =
{ pass = it }, label = {Text("Enter Password", color =
textcolor)}, keyboardOptions = KeyboardOptions(keyboardType =
KeyboardType.Password, imeAction = ImeAction.Go),
keyboardActions = KeyboardActions( onGo = { go() } ),
visualTransformation = PasswordVisualTransformation(),
textStyle = LocalTextStyle.current.copy(textcolor), colors =
TextFieldDefaults.outlinedTextFieldColors(focusedBorderColor
= textcolor, unfocusedBorderColor = textcolor))
        Spacer(modifier = Modifier.padding(16.dp))
```

```
        Button(onClick = {
            go()
        }, colors = ButtonDefaults.buttonColors(secondary)) {
            Text(text = if(checking) "Validating.." else
"Sign In", color = textcolor)
        }
        Spacer(modifier = Modifier.padding(8.dp))
        Row {
            Text(text = "Doesn't have account? ", color =
textcolor)
            ClickableText(text = AnnotatedString("Sign Up
here", spanStyle = SpanStyle(color = Color.Blue)), onClick =
{ navController.navigate("signup")})
        }
    }
}
```

signup.kt:

```
package com.example.letschat

import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.text.ClickableText
import androidx.compose.foundation.text.KeyboardActions
```

```
import androidx.compose.foundation.text.KeyboardOptions
import androidx.compose.material3.Button
import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.LocalTextStyle
import androidx.compose.material3.OutlinedTextField
import androidx.compose.material3.Text
import androidx.compose.material3.TextFieldDefaults
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.text.AnnotatedString
import androidx.compose.ui.text.SpanStyle
import androidx.compose.ui.text.font.FontFamily
import androidx.compose.ui.text.input.ImeAction
import androidx.compose.ui.text.input.KeyboardType
import
androidx.compose.ui.text.input.PasswordVisualTransformation
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.navigation.NavHostController
import com.google.firebase.database.DataSnapshot
```

```
import com.google.firebase.database.DatabaseError
import com.google.firebase.database.ValueEventListener
import com.google.firebase.database.ktx.getValue
import java.text.SimpleDateFormat
import java.util.Date

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun Signup(navController: NavHostController) {
    if(uname!="")
        navController.navigate("contacts")
    Column(modifier = Modifier
        .fillMaxSize()
        .background(primary), horizontalAlignment =
Alignment.CenterHorizontally, verticalArrangement =
Arrangement.Center) {
        Text(text = "Sign Up Page", fontFamily =
FontFamily.Cursive, fontSize = 48.sp, color = textcolor)
        Spacer(modifier = Modifier.padding(32.dp))
        var text by remember {
            mutableStateOf("")
        }
        Text(text = text, color = Color.Red)
        Spacer(modifier = Modifier.padding(4.dp))
        var name by remember {
            mutableStateOf("")
        }
```

```
        OutlinedTextField(value = name, onValueChange =
{ name = it }, label = { Text(text = "Enter your preferred
Username", color = textcolor) }, keyboardOptions =
KeyboardOptions(imeAction = ImeAction.Next), textStyle =
LocalTextStyle.current.copy(textcolor), colors =
TextFieldDefaults.outlinedTextFieldColors(focusedBorderColor
= textcolor, unfocusedBorderColor = textcolor))
        Spacer(modifier = Modifier.padding(8.dp))
        var pass1 by remember {
            mutableStateOf("")
        }
        OutlinedTextField(value = pass1, onValueChange =
{ pass1 = it }, label = { Text("Enter a strong Password",
color = textcolor) }, keyboardOptions =
KeyboardOptions(keyboardType = KeyboardType.Password,
imeAction = ImeAction.Next), visualTransformation =
PasswordVisualTransformation(), textStyle =
LocalTextStyle.current.copy(textcolor), colors =
TextFieldDefaults.outlinedTextFieldColors(focusedBorderColor
= textcolor, unfocusedBorderColor = textcolor))
        Spacer(modifier = Modifier.padding(8.dp))
        var pass2 by remember {
            mutableStateOf("")
        }
        var creating by remember {
        mutableStateOf(false)
        }
        var i = false
        fun creating(name: String, pass: String) {
            myRef = database.getReference(name)
```

```
            myRef.addValueEventListener(object:
ValueEventListener {

                override fun onDataChange(snapshot:
DataSnapshot) {
                    value =
snapshot.getValue<MutableList<messagedetails>>()
                    if(i) {
                        if (value == null) {
                            val sdf = SimpleDateFormat("yyyy
M dd hh mm ss a")
                            val currentDate =
sdf.format(Date()).split(" ")
                            val temp = messagedetails(
                                name,
                                "",
                                date(
                                    currentDate[0].toInt(),
                                    currentDate[1].toInt(),
                                    currentDate[2].toInt(),
                                    currentDate[3].toInt(),
                                    currentDate[4].toInt(),
                                    currentDate[5].toInt(),
                                    currentDate[6]
                                ),
                                "",
                                pass
                            )
```

```
                                value = mutableListOf(temp)
                                myRef.setValue(value)
                                uname = name

navController.navigate("contacts")
                            } else
                                text = "$name is already taken.
Please give another one"
                            i = false
                        }
                        creating = false
                    }

                    override fun onCancelled(error:
DatabaseError) {
                        println("Not yet implemented")
                    }
                })
                creating = true
            }
            fun go() {
                text = ""
                if(!creating) {
                    if (name.length >= 3) {
                        if (pass1 == pass2) {
                            if (pass1.length >= 8) {
                                i = true
                                creating(name, pass1)
```

```
                    } else
                        text = "Passwords must contain at
least 8 characters"
                } else
                    text = "Passwords should match"
            } else
                text = "Username must contain at least 3
characters"
                name = ""
                pass1 = ""
                pass2 = ""
            }
        }
        OutlinedTextField(value = pass2, onValueChange =
{ pass2 = it }, label = { Text("Re-enter your Password",
color = textcolor) }, keyboardOptions =
KeyboardOptions(keyboardType = KeyboardType.Password,
imeAction = ImeAction.Go), keyboardActions = KeyboardActions(
onGo= { go() } ), visualTransformation =
PasswordVisualTransformation(), textStyle =
LocalTextStyle.current.copy(textcolor), colors =
TextFieldDefaults.outlinedTextFieldColors(focusedBorderColor
= textcolor, unfocusedBorderColor = textcolor))
        Spacer(modifier = Modifier.padding(16.dp))
        Button(onClick = {
            go()
        }, colors = ButtonDefaults.buttonColors(secondary)) {
            Text(text = if(creating) "Creating.." else "Sign
Up", color = textcolor)
        }
```

```
        Spacer(modifier = Modifier.padding(8.dp))
        Row {
            Text(text = "Have an account? ", color =
textcolor)
            ClickableText(text = AnnotatedString("Login
here", spanStyle = SpanStyle(color = Color.Blue)), onClick =
{ navController.navigate("login")})
        }
    }
}
```

contacts.kt:

```
package com.example.letschat

import android.content.Context
import android.content.Intent
import android.net.Uri
import android.os.Handler
import androidx.compose.foundation.Image
import androidx.compose.foundation.background
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
```

```
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.width
import androidx.compose.foundation.text.KeyboardOptions
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.ArrowForward
import androidx.compose.material.icons.filled.Menu
import androidx.compose.material3.Card
import androidx.compose.material3.CardDefaults
import androidx.compose.material3.DrawerValue
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.LocalTextStyle
import androidx.compose.material3.ModalDrawerSheet
import androidx.compose.material3.ModalNavigationDrawer
import androidx.compose.material3.NavigationDrawerItem
import androidx.compose.material3.OutlinedTextField
import androidx.compose.material3.Text
import androidx.compose.material3.TextFieldDefaults
import androidx.compose.material3.rememberDrawerState
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.rememberCoroutineScope
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
```

```
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.text.font.FontStyle
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.input.ImeAction
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.navigation.NavHostController
import com.google.firebase.database.DataSnapshot
import com.google.firebase.database.DatabaseError
import com.google.firebase.database.ValueEventListener
import com.google.firebase.database.ktx.getValue
import kotlinx.coroutines.launch
import java.io.File

var sname = ""
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun Contacts(navController: NavHostController, context:
Context) {
    if(uname=="")
        navController.navigate("login")
    var contacts by remember {
        mutableStateOf(mutableListOf<String>())
    }
    var temp = mutableListOf<String>()
```

```
    for (x in value!!) {
        if(x.to!=""&&x.from!="") {
            if(x.from==uname) {
                if (temp.contains(x.to))
                    temp.remove(x.to)
                temp.add(0, x.to)
            } else {
                if (temp.contains(x.from))
                    temp.remove(x.from)
                temp.add(0, x.from)
            }
        }
    }
    contacts = temp
    val drawerstate = rememberDrawerState(initialValue =
DrawerValue.Closed)
    val scope = rememberCoroutineScope()
    val items = listOf(
        navigationItem(
            title = "Change Password",
            icon = painterResource(id =
R.drawable.changepassword),
            event = {
                scope.launch {
                    drawerstate.close()
                    navController.navigate("chpass")
                }
```

```
            }
        ),
        navigationItem(
            title = "Theme",
            icon = painterResource(id = R.drawable.theme),
            event = {
                if(Theme == "light")
                    dark()
                else
                    light()
                scope.launch {
                    drawerstate.close()
                    navController.navigate("contacts")
                }
            },
            badge = Theme
        ),
        navigationItem(
            title = "Share",
            icon = painterResource(id = R.drawable.share),
            event = {
                scope.launch {
                    drawerstate.close()
                    val file =
File(context.applicationInfo.sourceDir)
                    val intent = Intent(Intent.ACTION_SEND)
```

```
                    intent.type = "application/
vnd.android.package-archive"
                    intent.putExtra(Intent.EXTRA_STREAM,
Uri.fromFile(file))

context.startActivity(Intent.createChooser(intent, "Share
via").addFlags(Intent.FLAG_ACTIVITY_NEW_TASK))
                }
            }
        ),
        navigationItem(
            title = "About",
            icon = painterResource(id = R.drawable.info),
            event = {
                scope.launch {
                    drawerstate.close()
                    navController.navigate("about")
                }
            }
        )
    )
    ModalNavigationDrawer(
        drawerContent = {
            ModalDrawerSheet {
                Row(modifier = Modifier
                    .fillMaxWidth()
                    .padding(24.dp), horizontalArrangement =
Arrangement.Center) {
```

```
                    Text(
                        text = uname,
                        fontWeight = FontWeight.Bold,
                        fontSize = 24.sp,
                        fontStyle = FontStyle.Italic
                    )
                }
                items.forEach { item ->
                    NavigationDrawerItem(
                        label = { Text(text = item.title) },
                        selected = false,
                        onClick = item.event,
                        icon = { Image(painter = item.icon,
contentDescription = item.title, modifier = Modifier
                            .width(24.dp)
                            .height(24.dp)) },
                        badge = { Text(text = item.badge) }
                    )
                }
            }
        },
        drawerState = drawerstate
    ) {
        var add by remember {
            mutableStateOf(false)
        }
        Column (modifier = Modifier
```

```
            .fillMaxSize()
            .background(color = primary), horizontalAlignment
= Alignment.CenterHorizontally) {
            Row(modifier = Modifier
                .fillMaxWidth()
                .background(color = secondary)
                .padding(8.dp), horizontalArrangement =
Arrangement.SpaceBetween) {
                Row {
                    IconButton(onClick = {
                        scope.launch {
                            drawerstate.open()
                        }
                    }) {
                        Icon(
                            imageVector = Icons.Default.Menu,
                            contentDescription = "Navigation
Drawer Icon",
                            tint = textcolor
                        )
                    }
                    Spacer(modifier = Modifier.padding(4.dp,
0.dp, 4.dp, 0.dp))
                    Text(
                        text = "Letzchat",
                        color = secondary2,
                        fontSize = 32.sp,
                        fontWeight = FontWeight.Bold
```

```
                    )
                }
                Row {
                    /*Image(
                        painter = painterResource(id =
if(Theme == "light") R.drawable.reload_light else
R.drawable.reload_dark),
                        contentDescription = "add people",
                        modifier = Modifier
                            .clickable {

navController.navigate("contacts")
                            }
                            .height(24.dp)
                            .width(24.dp))
                    Spacer(modifier = Modifier.padding(4.dp,
0.dp, 4.dp, 0.dp))*/
                    Image(
                        painter = painterResource(id =
if(Theme == "light") R.drawable.addperson_black else
R.drawable.addperson_white),
                        contentDescription = "add people",
                        modifier = Modifier
                            .clickable {
                                add = !add
                            }
                            .height(24.dp)
                            .width(24.dp))
```

```
                    Spacer(modifier = Modifier.padding(4.dp,
0.dp, 4.dp, 0.dp))
                    Image(
                        painter = painterResource(id =
if(Theme == "light") R.drawable.logout_black else
R.drawable.logout_white),
                        contentDescription = "Logout",
                        modifier = Modifier
                            .clickable {
                                uname = ""

navController.navigate("login")
                            }
                            .height(24.dp)
                            .width(24.dp)
                    )
                }
            }
            if(add) {
                var name by remember {
                    mutableStateOf("")
                }
                var text by remember {
                    mutableStateOf("")
                }
                var i: Boolean
                Text(text = text, color = Color.Red)
                OutlinedTextField(
```

```
                    value = name,
                    onValueChange = { name = it },
                    label = { Text("receiver's username",
color = textcolor) },
                    textStyle =
LocalTextStyle.current.copy(textcolor),
                    colors =
TextFieldDefaults.outlinedTextFieldColors(
                        focusedBorderColor = textcolor,
                        unfocusedBorderColor = textcolor),
                    keyboardOptions =
KeyboardOptions(imeAction = ImeAction.Done),
                    trailingIcon = {
                        Icon(
                            imageVector =
Icons.Default.ArrowForward,
                            contentDescription = "Go",
                            tint = textcolor,
                            modifier = Modifier.clickable {
                                if(name.length>=3) {
                                    i = true
                                    othersRef =
database.getReference(name)

othersRef.addValueEventListener(object : ValueEventListener {

                                    override fun
onDataChange(snapshot: DataSnapshot) {
```

```
                                                bgvalue =
snapshot.getValue<MutableList<messagedetails>>()
                                                if(i) {
                                                    if (bgvalue !
= null) {
                                                        sname =
name

navController.navigate("chat")
                                                } else
                                                    text =
"No person with that username"
                                                    i = false
                                                }
                                            }

                                            override fun
onCancelled(error: DatabaseError) {
                                                println("Not yet
implemented")
                                            }
                                        })
                                    } else
                                        text = "Username must
contain atleast 3 characters"
                                }
                            )
                        }
                    )
```

```
            }
            Spacer(modifier = Modifier.padding(8.dp))
            var j: Boolean
            contacts.forEach { s ->
                Card(modifier = Modifier
                    .fillMaxWidth()
                    .clickable {
                        j = true
                        sname = s
                        othersRef =
database.getReference(sname)

othersRef.addValueEventListener(object : ValueEventListener {

                            override fun
onDataChange(snapshot: DataSnapshot) {
                                bgvalue =
snapshot.getValue<MutableList<messagedetails>>()
                                if (j) {

navController.navigate("chat")
                                    j = false
                                }
                            }

                            override fun onCancelled(error:
DatabaseError) {
                                println("Not yet
implemented")
```

```
                            }
                        })
                    },
                    colors =
CardDefaults.cardColors(primary2)) {
                    Row(modifier = Modifier
                        .fillMaxWidth()
                        .padding(16.dp),
horizontalArrangement = Arrangement.Center) {
                        Text(text = s, color = textcolor,
fontSize = 18.sp)
                    }
                }
                Spacer(modifier = Modifier.padding(4.dp))
            }
        }
    }
    fun loop() {
        val handler = Handler()
        handler.postDelayed({
            temp = mutableListOf()
            for (x in value!!) {
                if (x.to != "" && x.from != "") {
                    if (x.from == uname) {
                        if (temp.contains(x.to))
                            temp.remove(x.to)
                        temp.add(0, x.to)
                    } else {
```

```
                        if (temp.contains(x.from))
                            temp.remove(x.from)
                        temp.add(0, x.from)
                    }
                }
            }
            contacts = temp
            loop()
        }, 500)
    }
    loop()
}
```

chat.kt:

```
package com.example.letschat

import android.os.Handler
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.wrapContentHeight
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.foundation.lazy.rememberLazyListState
```

```
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.foundation.text.KeyboardActions
import androidx.compose.foundation.text.KeyboardOptions
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.ArrowBack
import androidx.compose.material.icons.filled.Send
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.LocalTextStyle
import androidx.compose.material3.Text
import androidx.compose.material3.TextField
import androidx.compose.material3.TextFieldDefaults
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.text.input.ImeAction
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.navigation.NavHostController
import java.text.SimpleDateFormat
```

```
import java.util.Date

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun Chat(navController: NavHostController) {
    if(uname=="")
        navController.navigate("login")
    var chats by remember {
        mutableStateOf(mutableListOf<messagedetails>())
    }
    var temp = mutableListOf<messagedetails>()
    for(x in value!!) {
        if(uname==x.from&&sname==x.to||
uname==x.to&&sname==x.from)
            temp.add(x)
    }
    chats = temp
    val lazyListState = rememberLazyListState()
    val currentDate = SimpleDateFormat("yyyy M
dd").format(Date()).split(" ")
    Column (modifier = Modifier
        .fillMaxSize()
        .background(color = primary)) {
        Row(modifier = Modifier
            .fillMaxWidth()
            .background(color = secondary)) {
            IconButton(onClick =
{ navController.navigate("contacts") }) {
```

```
                Icon(imageVector = Icons.Default.ArrowBack,
contentDescription = "go back", tint = textcolor)
            }
            Text(text = sname, modifier =
Modifier.padding(4.dp), fontSize = 24.sp, color = textcolor)
        }
        LazyColumn(modifier = Modifier.weight(1f), state =
lazyListState) {
            var temp_date = ""
            var temp_text = ""
            items(chats?.toList()!!) {
                if(temp_date != it.date.day.toString()
+"/"+it.date.month.toString()+"/"+it.date.year.toString()) {
                    temp_date = it.date.day.toString() + "/"
+ it.date.month.toString() + "/" + it.date.year.toString()
                    temp_text =
if(currentDate[0].toInt()==it.date.year&&currentDate[1].toInt
()==it.date.month&&currentDate[2].toInt()==it.date.day)
                        "today"
                    else if
(currentDate[0].toInt()==it.date.year&&currentDate[1].toInt()
==it.date.month&&currentDate[2].toInt()-1==it.date.day)
                        "yesterday"
                    else
                        temp_date
                    Row(modifier =
Modifier.wrapContentHeight().fillMaxWidth(),
horizontalArrangement = Arrangement.Center) {
                        Text(text = temp_text, color =
textcolor, fontSize = 12.sp, modifier =
```

```
Modifier.background(primary2,
RoundedCornerShape(4.dp)).padding(4.dp))
                    }
                }
                if(it.to==sname) {
                    Row(modifier = Modifier
                        .fillMaxWidth()
                        .padding(8.dp), horizontalArrangement
= Arrangement.End, verticalAlignment = Alignment.Bottom) {
                        Text(text = it.date.hour.toString()
+":"+it.date.minute.toString()+" "+it.date.meridiem, color =
Color.Gray, fontSize = 12.sp)
                        Text(text = it.message, color =
textcolor, modifier = Modifier.background(color = if (Theme
== "light") Color.White else Color.Black, shape =
RoundedCornerShape(8.dp)).padding(16.dp, 4.dp))
                    }
                } else {
                    Row(modifier = Modifier
                        .fillMaxWidth()
                        .padding(8.dp), horizontalArrangement
= Arrangement.Start, verticalAlignment = Alignment.Bottom) {
                        if(it.message.length>=30) {
                            Text(
                                text = it.message, color =
textcolor, modifier = Modifier
                                    .background(color =
primary2, shape = RoundedCornerShape(8.dp))
                                    .padding(16.dp, 4.dp)
                                    .weight(1f)
```

```
                            )
                        } else {
                            Text(
                                text = it.message, color =
textcolor, modifier = Modifier
                                    .background(color =
primary2, shape = RoundedCornerShape(8.dp))
                                    .padding(16.dp, 4.dp)
                            )
                        }
                        Text(text = it.date.hour.toString()
+":"+it.date.minute.toString()+" "+it.date.meridiem, color =
Color.Gray, fontSize = 12.sp)
                    }
                }
            }
        }
        Row(modifier = Modifier.fillMaxWidth(),
horizontalArrangement = Arrangement.Center, verticalAlignment
= Alignment.CenterVertically) {
            var msg by remember {
                mutableStateOf("")
            }
            TextField(
                value = msg,
                onValueChange = { msg = it },
                label = { Text("Enter your message here",
color = textcolor) },
```

```
                textStyle =
LocalTextStyle.current.copy(textcolor),
                colors =
TextFieldDefaults.outlinedTextFieldColors(
                    focusedBorderColor = textcolor,
                    unfocusedBorderColor = textcolor
                ),
                keyboardOptions = KeyboardOptions(imeAction =
ImeAction.Send),
                keyboardActions = KeyboardActions( onSend = {
send(msg); msg = "" } ),
                modifier = Modifier.weight(1f)
            )
            IconButton(onClick = {
                send(msg)
                msg = ""
            }, modifier = Modifier.background(color =
secondary, shape = RoundedCornerShape(16.dp))) {
                Icon(imageVector = Icons.Default.Send,
contentDescription = "Send", tint = textcolor)
            }
        }
    }
    LaunchedEffect(Unit) {
        lazyListState.scrollToItem(chats.size - 1)
    }
    fun loop() {
        val handler = Handler()
```

```
        handler.postDelayed({
            temp = mutableListOf()
            for (x in value!!) {
                if (uname == x.from && sname == x.to || uname
== x.to && sname == x.from)
                    temp.add(x)
            }
            chats = temp
            loop()
        }, 100)
    }
    loop()
}
```

```
fun send(msg: String) {
    val sdf = SimpleDateFormat("yyyy M dd hh mm ss a")
    val currentDate = sdf.format(Date()).split(" ")
    if(msg!="") {
        val temp = messagedetails(
            uname,
            sname,
            date(
                currentDate[0].toInt(),
                currentDate[1].toInt(),
                currentDate[2].toInt(),
                currentDate[3].toInt(),
                currentDate[4].toInt(),
```

```
                currentDate[5].toInt(),
                currentDate[6]
            ),
            msg
        )
        value?.add(temp)
        bgvalue?.add(temp)
        myRef.setValue(value)
        othersRef.setValue(bgvalue)
    }
}
```

ChPass.kt:

```
package com.example.letschat

import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.text.KeyboardActions
import androidx.compose.foundation.text.KeyboardOptions
import androidx.compose.material3.Button
import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.LocalTextStyle
import androidx.compose.material3.OutlinedTextField
```

```
import androidx.compose.material3.Text
import androidx.compose.material3.TextFieldDefaults
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.text.font.FontFamily
import androidx.compose.ui.text.input.ImeAction
import androidx.compose.ui.text.input.KeyboardType
import
androidx.compose.ui.text.input.PasswordVisualTransformation
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.navigation.NavHostController

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun ChPass(navController: NavHostController) {
Column(modifier = Modifier
    .fillMaxSize()
    .background(primary), horizontalAlignment =
Alignment.CenterHorizontally, verticalArrangement =
Arrangement.Center) {
```

```
    Text(text = "Change Password", fontFamily =
FontFamily.Cursive, fontSize = 48.sp, color = textcolor)
    Spacer(modifier = Modifier.padding(32.dp))
    var text by remember {
        mutableStateOf("")
    }
    Text(text = text, color = Color.Red)
    Spacer(modifier = Modifier.padding(4.dp))
    var oldpass by remember {
        mutableStateOf("")
    }
    OutlinedTextField(value = oldpass, onValueChange =
{ oldpass = it }, label = { Text("Enter your old Password",
color = textcolor) }, keyboardOptions =
KeyboardOptions(keyboardType = KeyboardType.Password,
imeAction = ImeAction.Next), visualTransformation =
PasswordVisualTransformation(), textStyle =
LocalTextStyle.current.copy(textcolor), colors =
TextFieldDefaults.outlinedTextFieldColors(focusedBorderColor
= textcolor, unfocusedBorderColor = textcolor))
    Spacer(modifier = Modifier.padding(8.dp))
    var newpass1 by remember {
        mutableStateOf("")
    }
    OutlinedTextField(value = newpass1, onValueChange =
{ newpass1 = it }, label = { Text("Enter your new Password",
color = textcolor) }, keyboardOptions =
KeyboardOptions(keyboardType = KeyboardType.Password,
imeAction = ImeAction.Next), visualTransformation =
PasswordVisualTransformation(), textStyle =
LocalTextStyle.current.copy(textcolor), colors =
```

```
TextFieldDefaults.outlinedTextFieldColors(focusedBorderColor
= textcolor, unfocusedBorderColor = textcolor))
    Spacer(modifier = Modifier.padding(8.dp))
    var newpass2 by remember {
        mutableStateOf("")
    }
    fun changing(old: String, new: String) {
        if(old == value?.get(0)?.pass){
            val temp = value?.get(0)!!
            value?.removeAt(0)
            temp.pass = new
            value?.add(temp)
            myRef.setValue(value)
            text = "Password has changed"
        } else
            text = "Password is incorrect"
    }
    fun go() {
        text = ""
```

```
if(oldpass.length>=8&&newpass1.length>=8&&newpass2.length>=8)
{
            if(newpass1==newpass2) {
                changing(oldpass, newpass1)
            } else
                text = "Both new passwords aren't matching"
        } else
```

```
            text = "Password must contain at least 8
characters"
        oldpass = ""
        newpass1 = ""
        newpass2 = ""
    }
    OutlinedTextField(value = newpass2, onValueChange =
{ newpass2 = it }, label = { Text("Re-enter your new
Password", color = textcolor) }, keyboardOptions =
KeyboardOptions(keyboardType = KeyboardType.Password,
imeAction = ImeAction.Go), keyboardActions = KeyboardActions(
onGo = { go() } ), visualTransformation =
PasswordVisualTransformation(), textStyle =
LocalTextStyle.current.copy(textcolor), colors =
TextFieldDefaults.outlinedTextFieldColors(focusedBorderColor
= textcolor, unfocusedBorderColor = textcolor))
    Spacer(modifier = Modifier.padding(16.dp))
    Button(onClick = { go() }, colors =
ButtonDefaults.buttonColors(secondary)) {
        Text(text = "Change", color = textcolor)
    }
}
}
```

About.kt:

```
package com.example.letschat

import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.fillMaxSize
```

```
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.navigation.NavHostController

@Composable
fun About(navController: NavHostController) {
    Column(modifier = Modifier
        .fillMaxSize()
        .background(color = primary)) {
        Row(modifier = Modifier
            .fillMaxWidth()
            .background(color = secondary)
            .padding(8.dp)) {
            Text(text = "About", fontSize = 20.sp, fontWeight
= FontWeight.Bold, color = textcolor)
        }
        Text(text = "Letzchat:\n" +
                "\"Letzchat\" is a cutting-edge mobile
application designed to enhance your daily communication with
your friends. With a wide range of features and a user-
friendly interface, \"Letzchat\" is the go-to app for
Chatting.\n" +
```

```
                "\n" +
                "Version:\n" +
                "Current Version: 1.0\n" +
                "\n" +
                "Description:\n" +
                "\"AppName\" is a versatile and intuitive app
that provides basic chatting interface. Whether you're
looking for simple chatting app or user-friendly app,
\"Letzchat\" has got you covered.\n" +
                "\n" +
                "Key Features:\n" +
                "You can change your account password
anytime\n" +
                "You can change app themes and save it\n" +
                "You also have a internal option to share the
app", color = textcolor)
    }
}
```

GitHub Link: https://github.com/smartinternz02/SI-GuidedProject-587108-1696856914/tree/main/Project%20Development%20Phase/LetsChat

Project Demo Link: https://drive.google.com/file/d/1S4OeHoQJy4XmCEAgWXFwUD2U5eX2y-Oa/view?usp=drive_link