ANDROID APPLICATION DEVELOPMENT

Adaptive Email-A Flexible Email Client app

Team ID- NM2024TMID05533

Submitted by

Joshiga S(Team Leader)	-	8FC4CFD6157D0C29B4B483372C1A915C
Divya NV(Team Member)	-	B5E50DCD24B2494BF7C4F55C92FD480F
Madheshwari M(Team Member)-		80A7FD53005A1FAB2DA4C5AC1ECC5DD1
Christina Joice I(Team Member)-		4D947322EA2C52CA9154BEA3DA3EF2D6

SEMESTER-V

B.E COMPUTER SCIENCE AND ENGINEERING

ACADEMIC YEAR-2024-2025

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ANNA UNIVERSITY REGIONAL CAPMUS COIMBATORE

COIMBATORE-641046

NOVEMBER 2024

Table Of Contents

Abstract

The Adaptive Mail project aims to enhance email communication through intelligent, context-aware features that adapt to users' preferences, communication patterns, and priorities. Using machine learning and natural language processing (NLP), it categorizes and prioritizes emails, offers adaptive responses, and addresses inbox overload and missed messages by automating sorting, filtering, and summarizing.
Key features include personalized categorization, automatic prioritization, smart reply suggestions, and sentiment analysis to assess tone and urgency. The system provides insights and adaptive reminders to maintain productivity and ensure important messages aren't missed. This innovative approach transforms email management, creating a more efficient, personalized, and user-friendly experience.

Project Description

Adaptive Mail app is a sample project that demonstrates how to use the Android Compose UI toolkit to build a conversational UI. The app simulates a messaging interface, allowing the user to send and receive messages, and view a history of previous messages. It showcases some of the key features of the Compose UI toolkit, data management, and user interactions.

Objectives

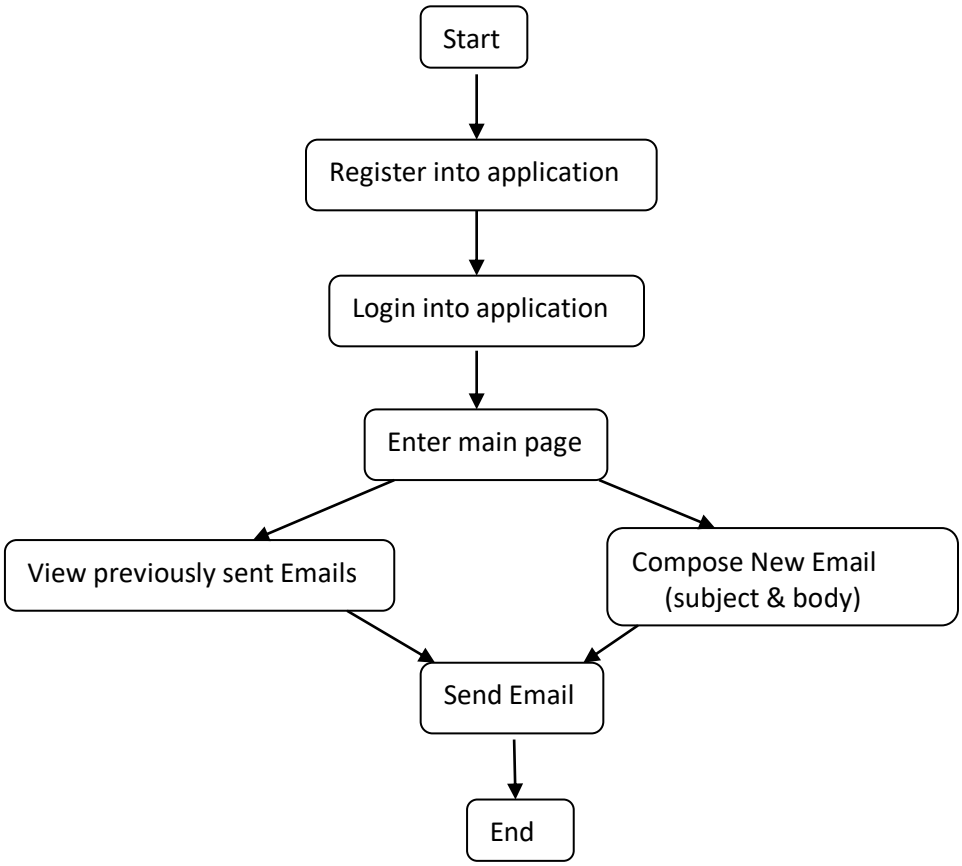
 Personalized Communication: Tailor emails dynamically to match recipients' preferences, behaviors, or demographics, increasing engagement and relevance.
 Real-time Adaptation: Enable content to update in real-time based on factors like location, time, or recent actions, ensuring the email content remains timely and accurate.
 Enhanced Responsiveness: Design emails that adapt to different devices and screen sizes, optimizing the reading experience across mobile, tablet, and desktop.
 Increased Conversion Rates: Boost conversion through targeted, context-aware messaging that appeals directly to individual recipient needs or current conditions.
 Data-driven Insights: Track and analyze recipient interactions with adaptive content to refine future email strategies, enhancing personalization and impact.
 Efficient Automation: Streamline the email creation process by using adaptive templates, reducing the need for manual adjustments while maintaining relevance.

Requirements

1. Development Environment
Android Studio: Primary IDE for Android development, supporting the Compose UI toolkit.Version: 2024.1.2.12
Android SDK: Essential development kit for building Android applications.
Kotlin: Programming language for Android development, compatible with Compose UI.
2. UI Toolkit
Jetpack Compose: Android’s modern UI toolkit for building declarative UIs, enabling the creation of adaptive, responsive layouts.
Material Design Components: For consistent design elements across the app, including buttons, cards, and list views.
3. Data Management and Storage
Room Database: For storing message history and local data persistently.
DataStore or SharedPreferences: For managing smaller sets of user preferences or app settings.
4. Machine Learning and Natural Language Processing (NLP)
TensorFlow Lite or ML Kit: For on-device machine learning tasks, such as message categorization, prioritization, and sentiment analysis.
Pre-trained NLP Models: Models for language understanding to enable adaptive responses and sentiment analysis (e.g., BERT or sentiment-specific models).
5. Backend and Networking
Firebase Realtime Database or Cloud Firestore: For storing user messages if cloud storage is needed, ensuring synchronization across devices.
RESTful API or GraphQL: For possible server communication if external data sources or remote storage are needed.
6. Libraries for Advanced Features
Coroutines and Flow: For handling asynchronous data flows and background tasks efficiently.
Accompanist Libraries: For enhanced Compose features, such as animation and navigation.

7. Testing and Debugging Tools
JUnit and Espresso: For unit testing, UI testing, and automated testing of user interactions.
Compose Testing Library: For testing Compose UI components and user interactions.
8. Productivity and Performance Optimization
Dependency Injection with Hilt or Dagger: For managing dependencies and making the codebase scalable.
LeakCanary: For memory leak detection to ensure efficient performance

Project Workflow



Testing and Debugging

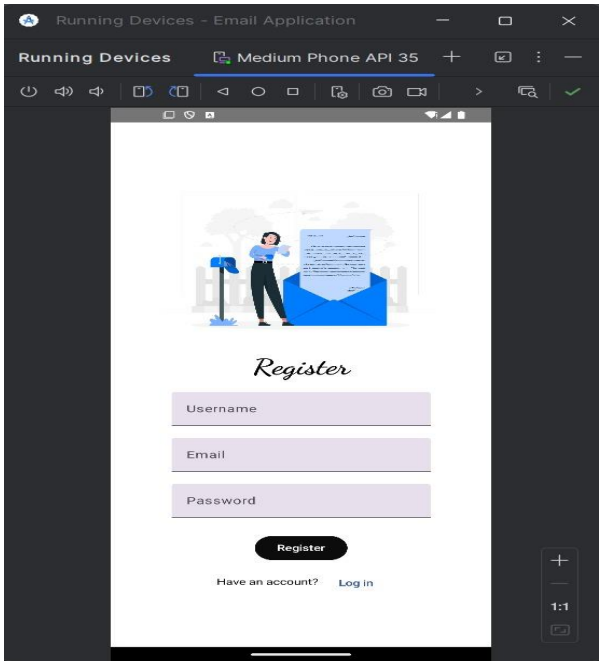
- Testing and debugging are essential to ensure the adaptive email client-server application performs reliably. The main testing methodologies include:
- Unit Testing: Verifies individual components, such as email parsing and network communication, with frameworks like JUnit and PyTest.
- Integration Testing: Checks client-server interactions and protocol communication (IMAP/SMTP) using tools like Postman.
- System Testing: Assesses the entire application flow to ensure end-to-end functionality and performance.
- Adaptive UI Testing: Uses tools like Selenium and Appium to validate UI responsiveness across different devices.

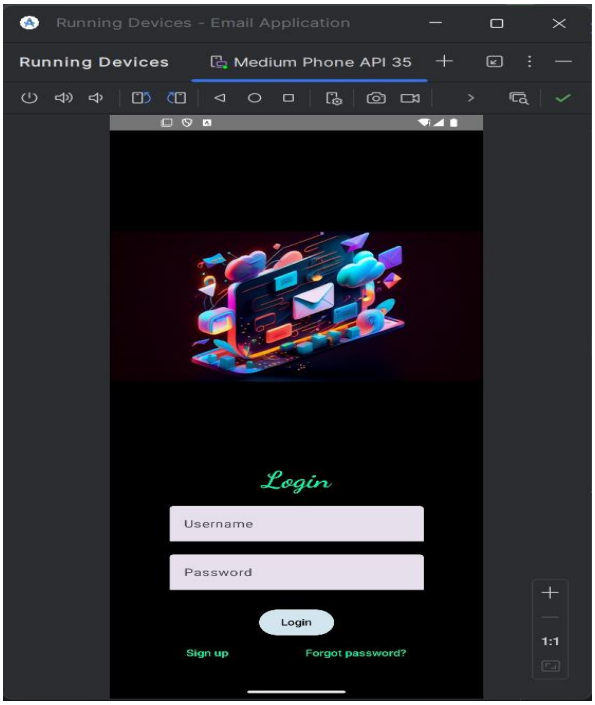
- User Acceptance Testing (UAT): Engages real users to confirm that the application meets expectations.
- Primary debugging techniques involve:
- Logging and Monitoring: Tools like the ELK Stack capture runtime data, helping to identify issues in adaptive behaviors.
- Automated Debugging and Profiling: GDB and PyCharm facilitate code tracing, while VisualVM

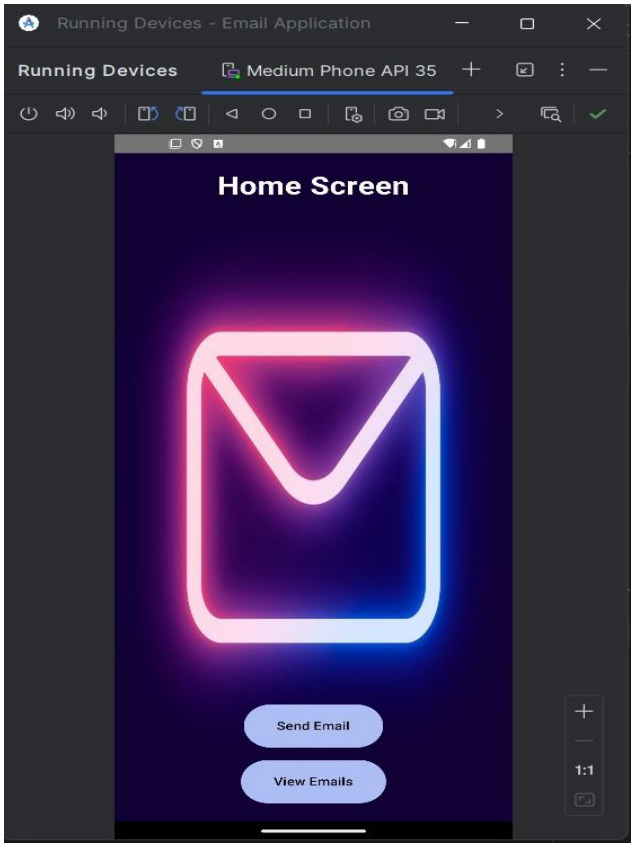
References

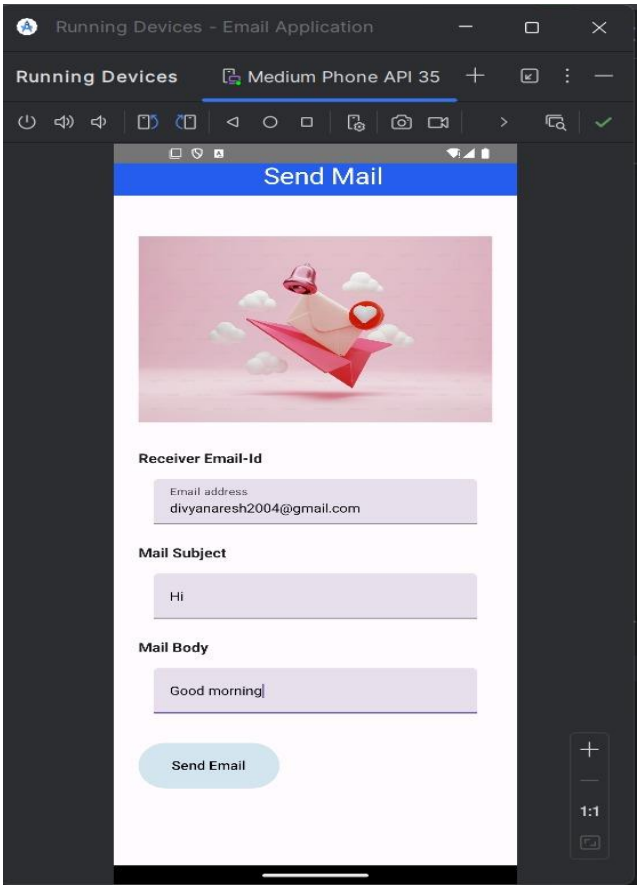
- RFC Standards and Protocols for Email Communication
- Core Concepts in Distributed and Data-Intensive Applications
- Adaptive and Scalable System Design Patterns
- Adaptive User Interfaces and Context-Aware Computing
- Machine Learning and Adaptive UI Design

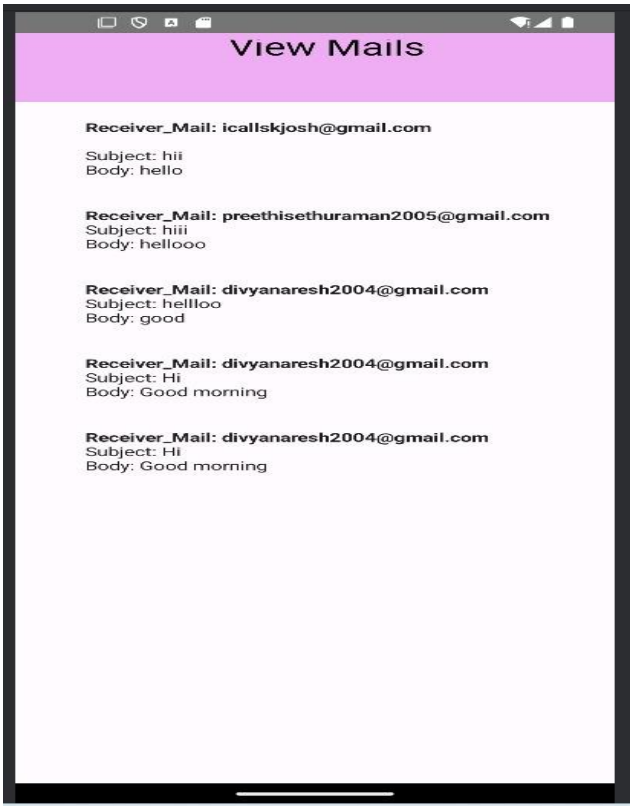
Output screenshot




Running Devices - Email Application
Running Devices
Medium Phone API 35
Login
Username
Password
Login
Sign up
Forgot password?
1:1


Running Devices - Email Application
Running Devices
Medium Phone API 35
Home Screen
Send Email
View Emails
1:1


Running Devices - Email Application
Running Devices
Medium Phone API 35
Send Mail
Receiver Email-Id
Email address
divyanaresh2004@gmail.com
Mail Subject
Hi
Mail Body
Good morning
Send Email
1:1


View Mails
Receiver_Mail: icallskjosh@gmail.com
Subject: hii
Body: hello
Receiver_Mail: preethisethuraman2005@gmail.com
Subject: hiii
Body: hellooo
Receiver_Mail: divyanaresh2004@gmail.com
Subject: hellloo
Body: good
Receiver_Mail: divyanaresh2004@gmail.com
Subject: Hi
Body: Good morning
Receiver_Mail: divyanaresh2004@gmail.com
Subject: Hi
Body: Good morning

Video Demo Link

https://drive.google.com/file/d/1a8g3PWkyUBRG2DE9uDkQWfdh1nsliz3r/view?usp=drive_link

Conclusion

The Adaptive Mail project demonstrates an intelligent, user-friendly email app built with the Android Compose UI toolkit. By leveraging machine learning for message prioritization, response suggestions, and personalized experiences, the app streamlines email management, enhancing productivity and reducing cognitive load. This project showcases the potential of Compose UI for modern interfaces and highlights the value of integrating AI to make digital communication more efficient and responsive.

APPENDIX-Source code

EMAIL DATABASE:

```
 package com.example.emailapplication
import android.content.Context
import androidx.room.Database
import androidx.room.Room
import androidx.room.RoomDatabase
@Database(entities = [Email::class], version = 1)
abstract class EmailDatabase : RoomDatabase() {
    abstract fun emailDao(): EmailDao
    companion object {
        @Volatile
        private var instance: EmailDatabase? = null
        fun getDatabase(context: Context): EmailDatabase {
            return instance ?: synchronized(this) {
                val newInstance = Room.databaseBuilder(
                    context.applicationContext,
                    EmailDatabase::class.java,
                    "email_database"
                ).build()
                instance = newInstance
                newInstance
            }
        }
    }
}
```

EMAIL DATABSE HELPER:

```
package com.example.emailapplication
import android.annotation.SuppressLint
import android.content.ContentValues
import android.content.Context
import android.database.Cursor
import android.database.sqlite.SQLiteDatabase
import android.database.sqlite.SQLiteOpenHelper
class EmailDatabaseHelper(context: Context) :
    SQLiteOpenHelper(context, DATABASE_NAME, null,DATABASE_VERSION){
    companion object {
        private const val DATABASE_VERSION = 1
        private const val DATABASE_NAME = "EmailDatabase.db"
        private const val TABLE_NAME = "email_table"
        private const val COLUMN_ID = "id"
        private const val COLUMN_RECEIVER_MAIL = "receiver_mail"
        private const val COLUMN_SUBJECT = "subject"
        private const val COLUMN_BODY = "body"
    }
    override fun onCreate(db: SQLiteDatabase?) {
        val createTable = "CREATE TABLE $TABLE_NAME (" +
                "${COLUMN_ID} INTEGER PRIMARY KEY AUTOINCREMENT, " +
                "${COLUMN_RECEIVER_MAIL} Text, " +
                "${COLUMN_SUBJECT} TEXT ," +
                "${COLUMN_BODY} TEXT " +
                ")"
        db?.execSQL(createTable)
    }
    override fun onUpgrade(db: SQLiteDatabase?, oldVersion: Int, newVersion: Int) {
        db?.execSQL("DROP TABLE IF EXISTS $TABLE_NAME")
        onCreate(db)
    }
    fun insertEmail(email: Email) {
        val db = writableDatabase
        val values = ContentValues()
        values.put(COLUMN_RECEIVER_MAIL, email.recevierMail)
        values.put(COLUMN_SUBJECT, email.subject)
        values.put(COLUMN_BODY, email.body)
        db.insert(TABLE_NAME, null, values)
        db.close()
    }
    @SuppressLint("Range")
    fun getEmailBySubject(subject: String): Email? {
```

```
        val db = readableDatabase
        val cursor: Cursor = db.rawQuery("SELECT * FROM $TABLE_NAME WHERE
$COLUMN_SUBJECT = ?", arrayOf(subject))
        var email: Email? = null
        if (cursor.moveToFirst()) {
            email = Email(
                id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),
                recevierMail =
cursor.getString(cursor.getColumnIndex(COLUMN_RECEIVER_MAIL)),
                subject = cursor.getString(cursor.getColumnIndex(COLUMN_SUBJECT)),
                body = cursor.getString(cursor.getColumnIndex(COLUMN_BODY)),
            )
        }
        cursor.close()
        db.close()
        return email
    }
    @SuppressLint("Range")
    fun getEmailById(id: Int): Email? {
        val db = readableDatabase
        val cursor: Cursor = db.rawQuery("SELECT * FROM $TABLE_NAME WHERE
$COLUMN_ID = ?", arrayOf(id.toString()))
        var email: Email? = null
        if (cursor.moveToFirst()) {
            email = Email(
                id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),
                recevierMail =
cursor.getString(cursor.getColumnIndex(COLUMN_RECEIVER_MAIL)),
                subject = cursor.getString(cursor.getColumnIndex(COLUMN_SUBJECT)),
                body = cursor.getString(cursor.getColumnIndex(COLUMN_BODY)),
            )
        }
        cursor.close()
        db.close()
        return email
    }
    @SuppressLint("Range")
    fun getAllEmails(): List<Email> {
        val emails = mutableListOf<Email>()
        val db = readableDatabase
        val cursor: Cursor = db.rawQuery("SELECT * FROM $TABLE_NAME", null)
        if (cursor.moveToFirst()) {
            do {
                val email = Email(
```

```
                id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),
                recevierMail =
cursor.getString(cursor.getColumnIndex(COLUMN_RECEIVER_MAIL)),
                subject = cursor.getString(cursor.getColumnIndex(COLUMN_SUBJECT)),
                body = cursor.getString(cursor.getColumnIndex(COLUMN_BODY)),
            )
            emails.add(email)
        } while (cursor.moveToNext())
    }
    cursor.close()
    db.close()
    return emails
  }

}
```

LOGIN ACTVITY:

```
package com.example.emailapplication
import android.content.Context
import android.content.Intent
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.Image
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.*
import androidx.compose.material.*
import androidx.compose.material3.Button
import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.text.font.FontFamily
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.input.PasswordVisualTransformation
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
```

```
import androidx.core.content.ContextCompat
import androidx. compose. material3.Text
import androidx. compose. material3.TextButton
import androidx.compose.material3.TextField
import androidx. compose. material3.TextButton
import com.example.emailapplication.ui.theme.EmailApplicationTheme
import com.example.emailapplication.UserDatabaseHelper
class LoginActivity : ComponentActivity() {
    private lateinit var databaseHelper: UserDatabaseHelper
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        databaseHelper = UserDatabaseHelper(this)
        setContent {
            LoginScreen(this, databaseHelper)
        }
    }
}
@Composable
fun LoginScreen(context: Context, databaseHelper: UserDatabaseHelper) {
    var username by remember { mutableStateOf("") }
    var password by remember { mutableStateOf("") }
    var error by remember { mutableStateOf("") }
    Column(
        modifier = Modifier.fillMaxSize().background(Color.Black),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {
        Image(
            painter = painterResource(id = R.drawable.email_login),
            contentDescription = "",
            modifier = Modifier
                .width(500.dp) // Replace 200.dp with the desired width
                .height(500.dp) // Replace 100.dp with the desired height
        )
        Text( color = Color(0xFF070707),
            fontSize = 36.sp,
            fontWeight = FontWeight.ExtraBold,
            fontFamily = FontFamily.Cursive,
            text = "Login"
        )
        Spacer(modifier = Modifier.height(10.dp))
        TextField(
            value = username,
            onValueChange = { username = it },
```

```
    label = { Text("Username") },
    modifier = Modifier.padding(10.dp)
        .width(280.dp)
)
TextField(
    value = password,
    onValueChange = { password = it },
    label = { Text("Password") },
    visualTransformation = PasswordVisualTransformation(),
    modifier = Modifier.padding(10.dp)
        .width(280.dp)
if (error.isNotEmpty()) {
    Text(
        text = error,
        color = MaterialTheme.colorScheme.error,
        modifier = Modifier.padding(vertical = 16.dp)
    )
}
Button(
    onClick = {
        if (username.isNotEmpty() && password.isNotEmpty()) {
            val user = databaseHelper.getUserByUsername(username)
            if (user != null && user.password == password) {
                error = ""  // Clear any previous errors
                // Navigate to MainActivity (send-mail page) on successful login
                context.startActivity(
                    Intent(
                        context,
                        MainActivity::class.java
                    )
                )
            } else {
                error = "Invalid username or password"
            }
        } else {
            error = "Please fill all fields"
        }
    },
    colors = ButtonDefaults.buttonColors(containerColor = Color(0xFFd3e5ef)),
    modifier = Modifier.padding(top = 16.dp)
) {
    Text(color = Color(0xFF070707),
        text = "Login")
}
```

```
        Row {
            TextButton(onClick = {context.startActivity(
                Intent(
                    context,
                    RegisterActivity::class.java
                )
            )}
            )
            { Text(color = Color(0xFF02FCAD),
                text = "Sign up") }
            TextButton(onClick = {
            })
            {
                Spacer(modifier = Modifier.width(60.dp))
                Text(color = Color(0xFF02FCAD),text = "Forgot password?")
            }
        }
    }
}
private fun startMainPage(context: Context) {
    val intent = Intent(context, MainActivity::class.java)
    context.startActivity(intent)
}
```

MAIN ACTIVITY:

```
package com.example.emailapplication
import android.content.Context
import android.content.Intent
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.Image
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.*
import androidx.compose.material.*
import androidx.compose.material3.Surface
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.res.painterResource
```

```
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.core.content.ContextCompat
import androidx.core.content.ContextCompat.startActivity
import com.example.emailapplication.ui.theme.EmailApplicationTheme
import androidx. compose. material3.Text
import androidx. compose. material3.Button
import androidx. compose. material3.ButtonDefaults
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            EmailApplicationTheme {
                // A surface container using the 'background' color from the theme
                Surface(
                    modifier = Modifier.fillMaxSize()
                ) {
                    Email(this)
                }
            }
        }
    }
}
@Composable
fun Email(context: Context) {
    Box(
        modifier = Modifier.fillMaxSize()
    ) {
        // Full-screen background image
        Image(
            painter = painterResource(id = R.drawable.home_screen),
            contentDescription = "",
            modifier = Modifier.fillMaxSize(),
            contentScale = ContentScale.FillBounds, // Scales the image to fill the screen without
stretching
        )
        // Overlay Column for Text at the top and Buttons at the bottom
        Column(
            modifier = Modifier.fillMaxSize(),
            horizontalAlignment = Alignment.CenterHorizontally
        ) {
            // Text at the top
```

```
Text(
    text = "Home Screen",
    modifier = Modifier
        .padding(top = 24.dp)
        .align(Alignment.CenterHorizontally), // Center text horizontally
    color = Color.White,
    fontWeight = FontWeight.Bold,
    fontSize = 32.sp
)
Spacer(modifier = Modifier.weight(1f)) // Spacer to push buttons to the bottom
// Buttons at the bottom
Column(
    horizontalAlignment = Alignment.CenterHorizontally,
    modifier = Modifier.padding(bottom = 24.dp)
) {
    Button(
        onClick = {
            context.startActivity(
                Intent(context, SendMailActivity::class.java)
            )
        },
        colors = ButtonDefaults.buttonColors(containerColor = Color(0xFFadbef4))
    ) {
        Text(
            text = "Send Email",
            modifier = Modifier.padding(10.dp),
            color = Color.Black,
            fontSize = 15.sp
        )
    }
    Spacer(modifier = Modifier.height(16.dp))
    Button(
        onClick = {
            context.startActivity(
                Intent(context, ViewMailActivity::class.java)
            )
        },
        colors = ButtonDefaults.buttonColors(containerColor = Color(0xFFadbef4))
    ) {
        Text(
            text = "View Emails",
            modifier = Modifier.padding(10.dp),
            color = Color.Black,
            fontSize = 15.sp
```

```
            )
        }
    }
  }
}
```

REGISTER ACTIVITY:

```
 package com.example.emailapplication
import android.content.Context
import android.content.Intent
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.Image
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.*
import androidx.compose.material.*
import androidx.compose.material3.Button
import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.material3.TextButton
import androidx.compose.material3.TextField
import androidx.compose.runtime.*
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.FontFamily
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.input.PasswordVisualTransformation
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.core.content.ContextCompat
class RegisterActivity : ComponentActivity() {
    private lateinit var databaseHelper: UserDatabaseHelper
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        databaseHelper = UserDatabaseHelper(this)
        setContent {
            RegistrationScreen(this, databaseHelper)
        }
```

```
    }
}
@Composable
fun RegistrationScreen(context: Context, databaseHelper: UserDatabaseHelper) {
    var username by remember { mutableStateOf("") }
    var password by remember { mutableStateOf("") }
    var email by remember { mutableStateOf("") }
    var error by remember { mutableStateOf("") }
    Column(
        modifier = Modifier.fillMaxSize().background(Color.White),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {
        Image(
            painterResource(id = R.drawable.email_signup), contentDescription = "",
            modifier = Modifier.height(300.dp)
        )
        Text(
            fontSize = 36.sp,
            fontWeight = FontWeight.ExtraBold,
            fontFamily = FontFamily.Cursive,
            text = "Register"
        )
        Spacer(modifier = Modifier.height(10.dp))
        TextField(
            value = username,
            onValueChange = { username = it },
            label = { Text("Username") },
            modifier = Modifier
                .padding(10.dp)
                .width(280.dp)
        )
        TextField(
            value = email,
            onValueChange = { email = it },
            label = { Text("Email") },
            modifier = Modifier
                .padding(10.dp)
                .width(280.dp)
        )
        TextField(
            value = password,
            onValueChange = { password = it },
            label = { Text("Password") },
```

```
    visualTransformation = PasswordVisualTransformation(),
    modifier = Modifier
        .padding(10.dp)
        .width(280.dp)
)
if (error.isNotEmpty()) {
    Text(
        text = error,
        color = MaterialTheme.colorScheme.error,
        modifier = Modifier.padding(vertical = 16.dp)
    )
}
Button(
    onClick = {
        if (username.isNotEmpty() && password.isNotEmpty() && email.isNotEmpty()) {
            val user = User(
                id = null,
                firstName = username,
                lastName = null,
                email = email,
                password = password
            )
            databaseHelper.insertUser(user)
            error = "User registered successfully"
            // Start LoginActivity using the current context
            context.startActivity(
                Intent(
                    context,
                    LoginActivity::class.java
                )
            )
        } else {
            error = "Please fill all fields"
        }
    },
    colors = ButtonDefaults.buttonColors(containerColor = Color(0xFF0C0C0C)),
    modifier = Modifier.padding(top = 16.dp)
) {
    Text(text = "Register")
}
Spacer(modifier = Modifier.width(10.dp))
Spacer(modifier = Modifier.height(10.dp))
Row() {
    Text(
```

```
            modifier = Modifier.padding(top = 14.dp), text = "Have an account?"
        )
        TextButton(onClick = {
            context.startActivity(
                Intent(
                    context,
                    LoginActivity::class.java
                )
            )
        })
        {
            Spacer(modifier = Modifier.width(10.dp))
            Text(color = Color(0xFF31539a),text = "Log in")
        }
    }
  }
}
private fun startMainPage(context: Context) {
  val intent = Intent(context, MainActivity::class.java)
  context.startActivity(intent)
}
```

SEND MAIL ACTIVITY:

```
package com.example.emailapplication
import android.annotation.SuppressLint
import android.content.Context
import android.content.Intent
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.Image
import androidx.compose.foundation.layout.*
import androidx.compose.material.*
import androidx.compose.material3.Scaffold
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.tooling.preview.Preview
```

```
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import com.example.emailapplication.ui.theme.EmailApplicationTheme
import androidx. compose. material3.TopAppBar
import androidx. compose. material3.Text
import androidx. compose. material3.TextField
import androidx. compose. material3.Button
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.MaterialTheme
import androidx. compose. material3.ButtonDefaults
import androidx. compose. material3.TopAppBarColors
import androidx.compose.material3.*
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.unit.dp
class SendMailActivity : ComponentActivity() {
    private lateinit var databaseHelper: EmailDatabaseHelper
    @OptIn(ExperimentalMaterial3Api::class)
    @SuppressLint("UnusedMaterialScaffoldPaddingParameter",
        "UnusedMaterial3ScaffoldPaddingParameter"
    )
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        databaseHelper = EmailDatabaseHelper(this)
        setContent {
            Scaffold(
                // in scaffold we are specifying top bar.
                topBar = {
                    // inside top bar we are specifying
                    // background color.
                    TopAppBar(
                        colors = TopAppBarDefaults.topAppBarColors(containerColor =
Color(0xFF255EEF)),
                        modifier = Modifier.height(40.dp),
                        title = {
                            // in the top bar we are specifying
                            // title as a text
                            Text(
                                // on below line we are specifying
                                // text to display in top app bar.
                                text = "Send Mail",
                                fontSize = 28.sp,
                                color = Color.White,
                                // on below line we are specifying
                                // modifier to fill max width.
```

```
                        modifier = Modifier.fillMaxWidth(),
                        // on below line we are
                        // specifying text alignment.
                        textAlign = TextAlign.Center,
                    )
                }
            )
        }
    ) {
        // on below line we are
        // calling method to display UI.
        openEmailer(this,databaseHelper)
    }
  }
}
@Composable
fun openEmailer(context: Context, databaseHelper: EmailDatabaseHelper)  {
  // in the below line, we are
  // creating variables for URL
  var recevierMail by remember {mutableStateOf("") }
  var subject by remember {mutableStateOf("") }
  var body by remember {mutableStateOf("") }
  var error by remember { mutableStateOf("") }
  // on below line we are creating
  // a variable for a context
  val ctx = LocalContext.current
  // on below line we are creating a column
  Column(
    // on below line we are specifying modifier
    // and setting max height and max width
    // for our column
    modifier = Modifier
      .fillMaxSize()
      .padding(top = 55.dp, bottom = 25.dp, start = 25.dp, end = 25.dp),
    horizontalAlignment = Alignment.Start
  ) {
    Image(
      painterResource(id = R.drawable.email_send), contentDescription = "",
      modifier = Modifier.height(300.dp)
    )
    // on the below line, we are
    // creating a text field.
    Text(text = "Receiver Email-Id",
```

```
        fontWeight = FontWeight.Bold,
        fontSize = 16.sp)
    TextField(
        // on below line we are specifying
        // value for our  text field.
        value = recevierMail,
        // on below line we are adding on value
        // change for text field.
        onValueChange = { recevierMail = it },
        // on below line we are adding place holder as text
        label = { Text(text = "Email address") },
        placeholder = { Text(text = "abc@gmail.com") },
        // on below line we are adding modifier to it
        // and adding padding to it and filling max width
        modifier = Modifier
            .padding(16.dp)
            .fillMaxWidth(),
        // on below line we are adding text style
        // specifying color and font size to it.
        textStyle = TextStyle(color = Color.Black, fontSize = 15.sp),
        // on below line we are
        // adding single line to it.
        singleLine = true,
    )
    // on below line adding a spacer.
    Spacer(modifier = Modifier.height(10.dp))
    Text(text = "Mail Subject",
        fontWeight = FontWeight.Bold,
        fontSize = 16.sp)
    // on the below line, we are creating a text field.
    TextField(
        // on below line we are specifying
        // value for our  text field.
        value = subject,
        // on below line we are adding on value change
        // for text field.
        onValueChange = { subject = it },
        // on below line we are adding place holder as text
        placeholder = { Text(text = "Subject") },
        // on below line we are adding modifier to it
        // and adding padding to it and filling max width
        modifier = Modifier
            .padding(16.dp)
            .fillMaxWidth(),
```

```
    // on below line we are adding text style
    // specifying color and font size to it.
    textStyle = TextStyle(color = Color.Black, fontSize = 15.sp),
    // on below line we are
    // adding single line to it.
    singleLine = true,
)
// on below line adding a spacer.
Spacer(modifier = Modifier.height(10.dp))
Text(text = "Mail Body",
    fontWeight = FontWeight.Bold,
    fontSize = 16.sp)
// on the below line, we are creating a text field.
TextField(
    // on below line we are specifying
    // value for our  text field.
    value = body,
    // on below line we are adding on value
    // change for text field.
    onValueChange = { body = it },
    // on below line we are adding place holder as text
    placeholder = { Text(text= "Body") },
    // on below line we are adding modifier to it
    // and adding padding to it and filling max width
    modifier = Modifier
        .padding(16.dp)
        .fillMaxWidth(),

    // on below line we are adding text style
    // specifying color and font size to it.
    textStyle = TextStyle(color = Color.Black, fontSize = 15.sp),
    // on below line we are
    // adding single line to it.
    singleLine = true,
)
// on below line adding a spacer.
Spacer(modifier = Modifier.height(20.dp))
// on below line adding a
// button to send an email
Button(onClick = {
    if( recevierMail.isNotEmpty() && subject.isNotEmpty() && body.isNotEmpty()) {
        val email = Email(
            id = null,
            recevierMail = recevierMail,
```

```
                subject = subject,
                body = body
            )
            databaseHelper.insertEmail(email)
            error = "Mail Saved"
        } else {
            error = "Please fill all fields"
        }
        // on below line we are creating
        // an intent to send an email
        val i = Intent(Intent.ACTION_SEND)
        // on below line we are passing email address,
        // email subject and email body
        val emailAddress = arrayOf(recevierMail)
        i.putExtra(Intent.EXTRA_EMAIL,emailAddress)
        i.putExtra(Intent.EXTRA_SUBJECT,subject)
        i.putExtra(Intent.EXTRA_TEXT,body)
        // on below line we are
        // setting type of intent
        i.setType("message/rfc822")
        // on the below line we are starting our activity to open email application.
        ctx.startActivity(Intent.createChooser(i,"Choose an Email client : "))
    },
        colors = ButtonDefaults.buttonColors(containerColor = Color(0xFFd3e5ef))
    ) {
        // on the below line creating a text for our button.
        Text(
            // on below line adding a text ,
            // padding, color and font size.
            text = "Send Email",
            modifier = Modifier.padding(10.dp),
            color = Color.Black,
            fontSize = 15.sp
        )
    }
  }
}
```

USER:

```
package com.example.emailapplication
import androidx.room.ColumnInfo
import androidx.room.Entity
import androidx.room.PrimaryKey
```

```
@Entity(tableName = "user_table")
data class User(
    @PrimaryKey(autoGenerate = true) val id: Int?,
    @ColumnInfo(name = "first_name") val firstName: String?,
    @ColumnInfo(name = "last_name") val lastName: String?,
    @ColumnInfo(name = "email") val email: String?,
    @ColumnInfo(name = "password") val password: String?,
    )
 package com.example.emailapplication
import androidx.room.*
@Dao
interface UserDao {
    @Query("SELECT * FROM user_table WHERE email = :email")
    suspend fun getUserByEmail(email: String): User?
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend un insertUser(user: User)
    @Update
    suspend fun updateUser(user: User)
    @Delete
    suspend fun deleteUser(user: User)
}
```

USER DATABASE :

```
 package com.example.emailapplication
import android.content.Context
import androidx.room.Database
import androidx.room.Room
import androidx.room.RoomDatabase
@Database(entities = [User::class], version = 1)
abstract class UserDatabase : RoomDatabase() {
    abstract fun userDao(): UserDao
    companion object {
        @Volatile
        private var instance: UserDatabase? = null
        fun getDatabase(context: Context): UserDatabase {
            return instance ?: synchronized(this) {
                val newInstance = Room.databaseBuilder(
                    context.applicationContext,
                    UserDatabase::class.java,
                    "user_database"
                ).build()
                instance = newInstance
                newInstance
            }
        }
```

```
    }
}
```

USER DATABASE HELPER:

```
 package com.example.emailapplication
import android.annotation.SuppressLint
import android.content.ContentValues
import android.content.Context
import android.database.Cursor
import android.database.sqlite.SQLiteDatabase
import android.database.sqlite.SQLiteOpenHelper
class UserDatabaseHelper(context: Context) :
    SQLiteOpenHelper(/* context = / context, / name = */
        DATABASE_NAME, /* factory = */
        null, /* version = */
        DATABASE_VERSION) {
    companion object {
        private const val DATABASE_VERSION = 1
        private const val DATABASE_NAME = "UserDatabase.db"
        private const val TABLE_NAME = "user_table"
        private const val COLUMN_ID = "id"
        private const val COLUMN_FIRST_NAME = "first_name"
        private const val COLUMN_LAST_NAME = "last_name"
        private const val COLUMN_EMAIL = "email"
        private const val COLUMN_PASSWORD = "password"
    }
    override fun onCreate(db: SQLiteDatabase?) {
        val createTable = "CREATE TABLE $TABLE_NAME (" +
                "$COLUMN_ID INTEGER PRIMARY KEY AUTOINCREMENT, " +
                "$COLUMN_FIRST_NAME TEXT, " +
                "$COLUMN_LAST_NAME TEXT, " +
                "$COLUMN_EMAIL TEXT, " +
                "$COLUMN_PASSWORD TEXT" +
                ")"
        db?.execSQL(createTable)
    }
    override fun onUpgrade(db: SQLiteDatabase?, oldVersion: Int, newVersion: Int) {
        db?.execSQL("DROP TABLE IF EXISTS $TABLE_NAME")
        onCreate(db)
    }
    fun insertUser(user: User) {
        val db = writableDatabase
        val values = ContentValues()
```

```
        values.put(COLUMN_FIRST_NAME, user.firstName)
        values.put(COLUMN_LAST_NAME, user.lastName)
        values.put(COLUMN_EMAIL, user.email)
        values.put(COLUMN_PASSWORD, user.password)
        db.insert(TABLE_NAME, null, values)
        db.close()
    }
    @SuppressLint("Range")
    fun getUserByUsername(username: String): User? {
        val db = readableDatabase
        val cursor: Cursor = db.rawQuery("SELECT * FROM $TABLE_NAME WHERE
$COLUMN_FIRST_NAME = ?", arrayOf(username))
        var user: User? = null
        if (cursor.moveToFirst()) {
            user = User(
                id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),
                firstName = cursor.getString(cursor.getColumnIndex(COLUMN_FIRST_NAME)),
                lastName = cursor.getString(cursor.getColumnIndex(COLUMN_LAST_NAME)),
                email = cursor.getString(cursor.getColumnIndex(COLUMN_EMAIL)),
                password = cursor.getString(cursor.getColumnIndex(COLUMN_PASSWORD)),
            )
        }
        cursor.close()
        db.close()
        return user
    }
    @SuppressLint("Range")
    fun getUserById(id: Int): User? {
        val db = readableDatabase
        val cursor: Cursor = db.rawQuery("SELECT * FROM $TABLE_NAME WHERE
$COLUMN_ID = ?", arrayOf(id.toString()))
        var user: User? = null
        if (cursor.moveToFirst()) {
            user = User(
                id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),
                firstName = cursor.getString(cursor.getColumnIndex(COLUMN_FIRST_NAME)),
                lastName = cursor.getString(cursor.getColumnIndex(COLUMN_LAST_NAME)),
                email = cursor.getString(cursor.getColumnIndex(COLUMN_EMAIL)),
                password = cursor.getString(cursor.getColumnIndex(COLUMN_PASSWORD)),
            )
        }
        cursor.close()
        db.close()
        return user
```

```
    }
    @SuppressLint("Range")
    fun getAllUsers(): List<User> {
        val users = mutableListOf<User>()
        val db = readableDatabase
        val cursor: Cursor = db.rawQuery("SELECT * FROM $TABLE_NAME", null)
        if (cursor.moveToFirst()) {
            do {
                val user = User(
                    id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),
                    firstName = cursor.getString(cursor.getColumnIndex(COLUMN_FIRST_NAME)),
                    lastName = cursor.getString(cursor.getColumnIndex(COLUMN_LAST_NAME)),
                    email = cursor.getString(cursor.getColumnIndex(COLUMN_EMAIL)),
                    password = cursor.getString(cursor.getColumnIndex(COLUMN_PASSWORD)),
                )
                users.add(user)
            } while (cursor.moveToNext())
        }
        cursor.close()
        db.close()
        return users
    }

}
```

VIEW MAIL ACTIVITY:

```
 package com.example.emailapplication
import android.annotation.SuppressLint
import android.os.Bundle
import android.util.Log
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.LazyRow
import androidx.compose.foundation.lazy.items
import androidx.compose.material.*
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.Scaffold
import androidx.compose.material3.TopAppBar
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
```

```
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx. compose. material3.TopAppBar
import androidx. compose. material3.Text
import androidx.compose.material3.TopAppBarDefaults
class ViewMailActivity : ComponentActivity() {
    private lateinit var emailDatabaseHelper: EmailDatabaseHelper
    @OptIn(ExperimentalMaterial3Api::class)
    @SuppressLint("UnusedMaterialScaffoldPaddingParameter",
        "UnusedMaterial3ScaffoldPaddingParameter"
    )
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        emailDatabaseHelper = EmailDatabaseHelper(this)
        setContent {
            Scaffold(
                // in scaffold we are specifying top bar.
                topBar = {
                    // inside top bar we are specifying
                    // background color.
                    TopAppBar(
                        colors = TopAppBarDefaults.topAppBarColors(containerColor =
Color(0xFFadbef4)),
                        modifier = Modifier.height(80.dp),
                        // along with that we are specifying
                        // title for our top bar.
                        title = {
                            // in the top bar we are specifying
                            // title as a text
                            Text(
                                // on below line we are specifying
                                // text to display in top app bar.
                                text = "View Mails",
                                fontSize = 32.sp,
                                color = Color.Black,
                                // on below line we are specifying
                                // modifier to fill max width.
                                modifier = Modifier.fillMaxWidth(),
                                // on below line we are
                                // specifying text alignment.
                                textAlign = TextAlign.Center,
                            )
```

```
                    }
                )
            }
        ) {
            val data = emailDatabaseHelper.getAllEmails();
            Log.d("swathi", data.toString())
            val email = emailDatabaseHelper.getAllEmails()
            ListListScopeSample(email)
        }
    }
  }
}
@Composable
fun ListListScopeSample(email: List<Email>) {
  LazyRow(
    modifier = Modifier
      .fillMaxSize(),
    horizontalArrangement = Arrangement.SpaceBetween
  ) {
    item {
      LazyColumn {
        items(email) { email ->
          Column(
            modifier = Modifier.padding(
              top = 16.dp,
              start = 48.dp,
              bottom = 20.dp
            )
          ) {
            Text(text = "Receiver_Mail: ${email.recevierMail}", fontWeight =
FontWeight.Bold)
            Text("Subject: ${email.subject}")
            Text("Body: ${email.body}")
          }
        }
      }
    }
  }
}
 <?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools" >

  <application
```

```
    android:allowBackup="true"
    android:dataExtractionRules="@xml/data_extraction_rules"
    android:fullBackupContent="@xml/backup_rules"
    android:icon="@mipmap/ic_launcher"
    android:label="Email Application"
    android:supportsRtl="true"
    android:theme="@style/Theme.EmailApplication"
    tools:targetApi="31" >
    <activity
      android:name=".RegisterActivity"
      android:exported="false"
      android:label="RegisterActivity"
      android:theme="@style/Theme.EmailApplication" />
    <activity
      android:name=".MainActivity"
      android:exported="false"
      android:label="MainActivity"
      android:theme="@style/Theme.EmailApplication" />
    <activity
      android:name=".ViewMailActivity"
      android:exported="false"
      android:label="ViewMailActivity"
      android:theme="@style/Theme.EmailApplication" />
    <activity
      android:name=".SendMailActivity"
      android:exported="false"
      android:label="SendMailActivity"
      android:theme="@style/Theme.EmailApplication" />
    <activity
      android:name=".LoginActivity"
      android:exported="true"
      android:label="Email Application"
      android:theme="@style/Theme.EmailApplication" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>

</manifest>
plugins {
  alias(libs.plugins.android.application)
```

```
    alias(libs.plugins.kotlin.android)
    alias(libs.plugins.kotlin.compose)
}
android {
    namespace = "com.example.emailapplication"
    compileSdk 35

    defaultConfig {
        applicationId = "com.example.emailapplication"
        minSdk = 21
        targetSdk = 34
        versionCode = 1
        versionName = "1.0"

        testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            minifyEnabled true
            proguardFiles(
                getDefaultProguardFile("proguard-android-optimize.txt"),
                "proguard-rules.pro"
            )
        }
    }
    compileOptions {
        sourceCompatibility = JavaVersion.VERSION_11
        targetCompatibility = JavaVersion.VERSION_11
    }
    kotlinOptions {
        jvmTarget = "11"
    }
    buildFeatures {
        compose = true
    }
}
dependencies {
    implementation(libs.androidx.core.ktx)
    implementation(libs.androidx.lifecycle.runtime.ktx)
    implementation(libs.androidx.activity.compose)
    implementation(platform(libs.androidx.compose.bom))
    implementation(libs.androidx.ui)
    implementation(libs.androidx.ui.graphics)
```

```
    implementation(libs.androidx.ui.tooling.preview)
    implementation(libs.androidx.material3)
    testImplementation(libs.junit)
    androidTestImplementation(libs.androidx.junit)
    androidTestImplementation(libs.androidx.espresso.core)
    androidTestImplementation(platform(libs.androidx.compose.bom))
    androidTestImplementation(libs.androidx.ui.test.junit4)
    debugImplementation(libs.androidx.ui.tooling)
    debugImplementation(libs.androidx.ui.test.manifest)

    implementation 'androidx.room:room-common:2.5.0'
    implementation 'androidx.room:room-ktx:2.5.0'
}
```