CONTENT

PROJECT REPORT

KARTHIKA.M 0520128005

KASTHURI.V 0520128006

MAHESWARI.G 0520128007

MUTHUSELVI.C 0520128008

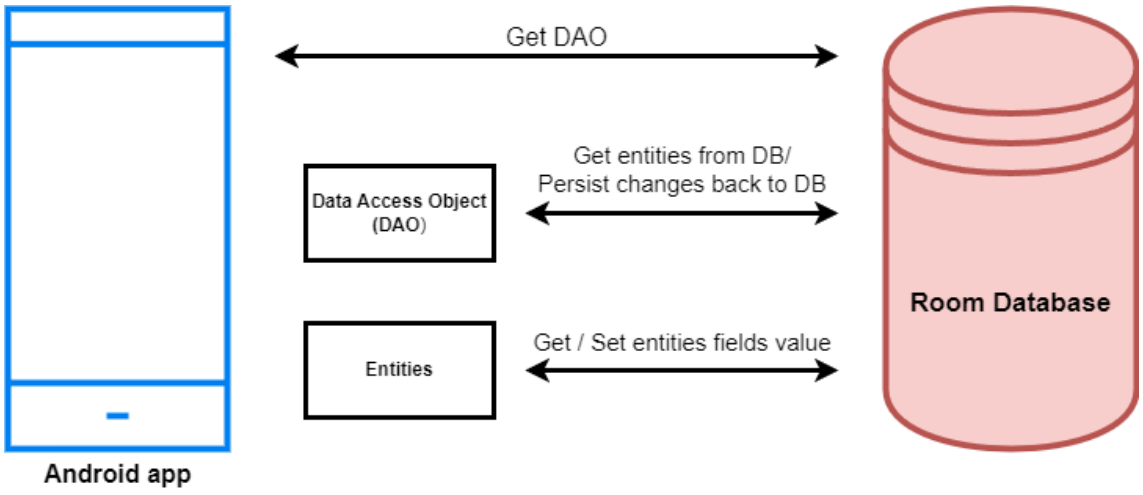
KANISHKA.S 5619128005

ADAPTIVE MAIL : A FLEXIBLE EMAIL CLIENT APP

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.

Architecture



Learning Outcomes :

By end of this project:

- You'll be able to work on Android studio and build an app.
- You'll be able to integrate the database accordingly.

Project Workflow:

- Users register into the application.
- After registration , user logins into the application.
- User enters into the main page
- User can View previously sent emails.
- User can give subject and email body to send email.

Note:

To complete the project you need to finish up the tasks listed below:

Tasks:

1.Required initial steps
2.Creating a new project.
3.Adding required dependencies.
4.Creating the database classes.
5.Building application UI and connecting to database.
6.Using AndroidManifest.xml
7.Running the application.

1 INTRODUCTION:

1.1 OVERVIEW:

Adaptive Mail is a flexible email client application designed to simplify email management and improve productivity for users. The

app is available for both desktop and mobile devices, allowing users to access their emails on the go.

The app offers a range of features to help users manage their emails efficiently. For example, users can easily filter and sort emails based on various criteria, such as sender, subject, or date. The app also allows users to create custom folders and labels to organize their emails and keep their inbox clutter-free.

One of the unique features of Adaptive Mail is its adaptive learning algorithm, which learns from the user's behaviour and preferences over time. This allows the app to offer personalized suggestions and automate repetitive tasks, such as labelling or archiving emails.

In addition, Adaptive Mail offers a range of customization options, allowing users to personalize the app to their preferences. For example, users can choose from a range of themes, customize the layout and font size, and set up notifications to stay on top of their emails.

Overall, Adaptive Mail is a powerful and flexible email client that offers a range of features to help users manage their emails efficiently and improve their productivity.

1.1 PURPOSE:

The purpose of Adaptive Mail is to provide users with a flexible and efficient email client app that simplifies email management and improves productivity. The app is designed to help users keep their inbox organized and clutter-free, and to automate repetitive tasks to save time.

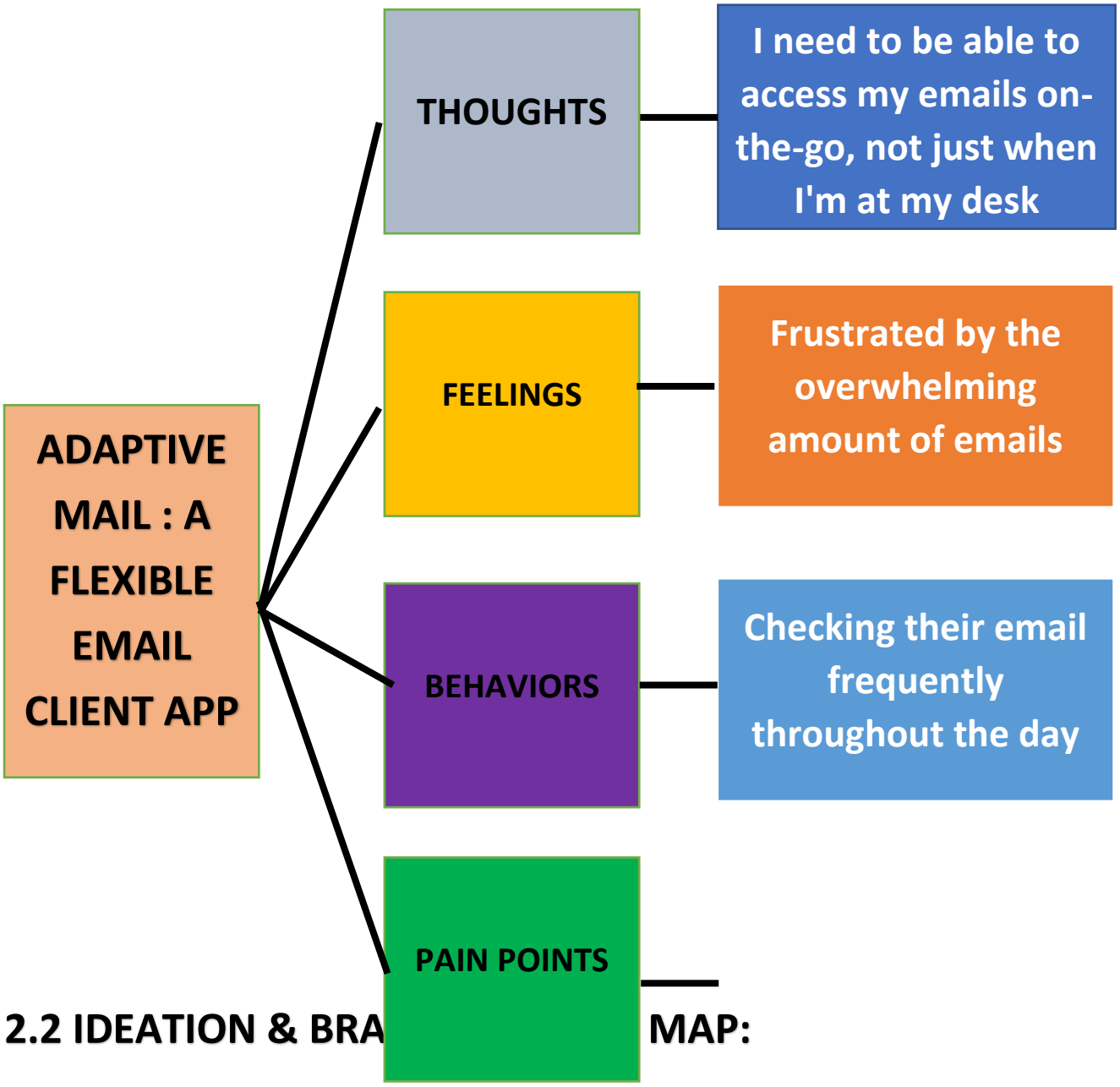
Adaptive Mail is built to adapt to the user's behaviour and preferences over time, learning from their actions to offer personalized suggestions and automate tasks. This adaptability helps users to manage their emails more efficiently and effectively.

The app is also designed to be highly customizable, allowing users to personalize the app to their preferences and workflow. This customization helps users to work more efficiently and comfortably.

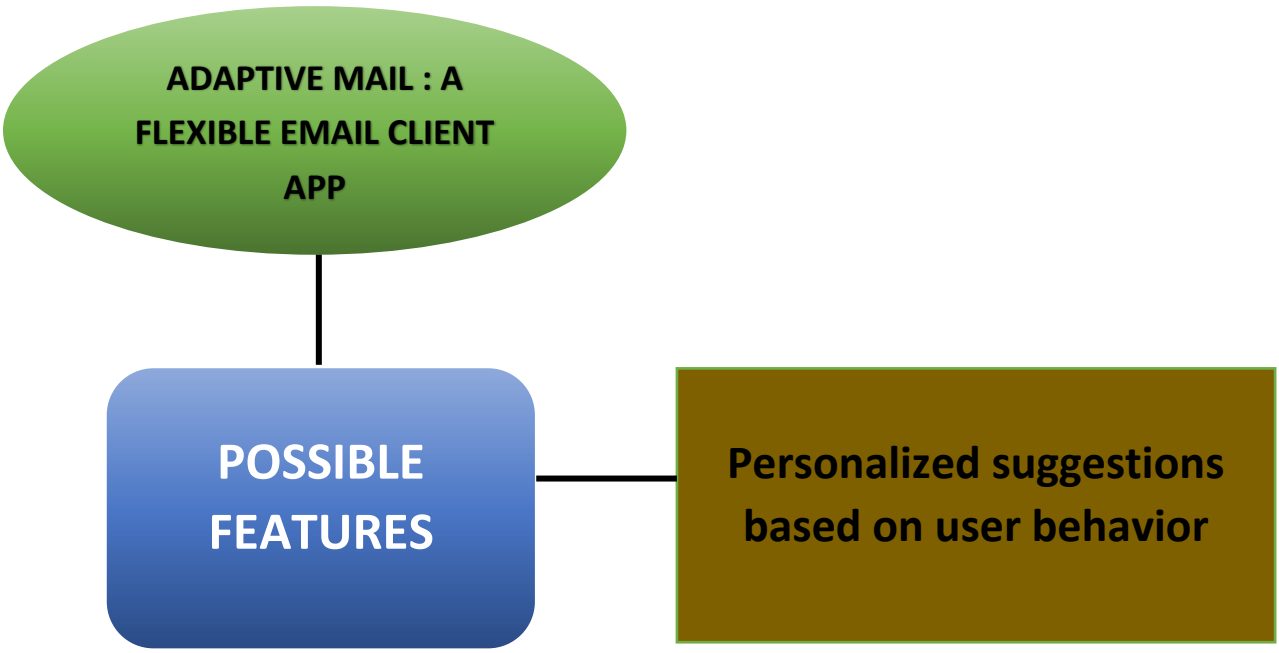
Overall, the purpose of Adaptive Mail is to make email management easier and more efficient for users, helping them to save time and stay productive.

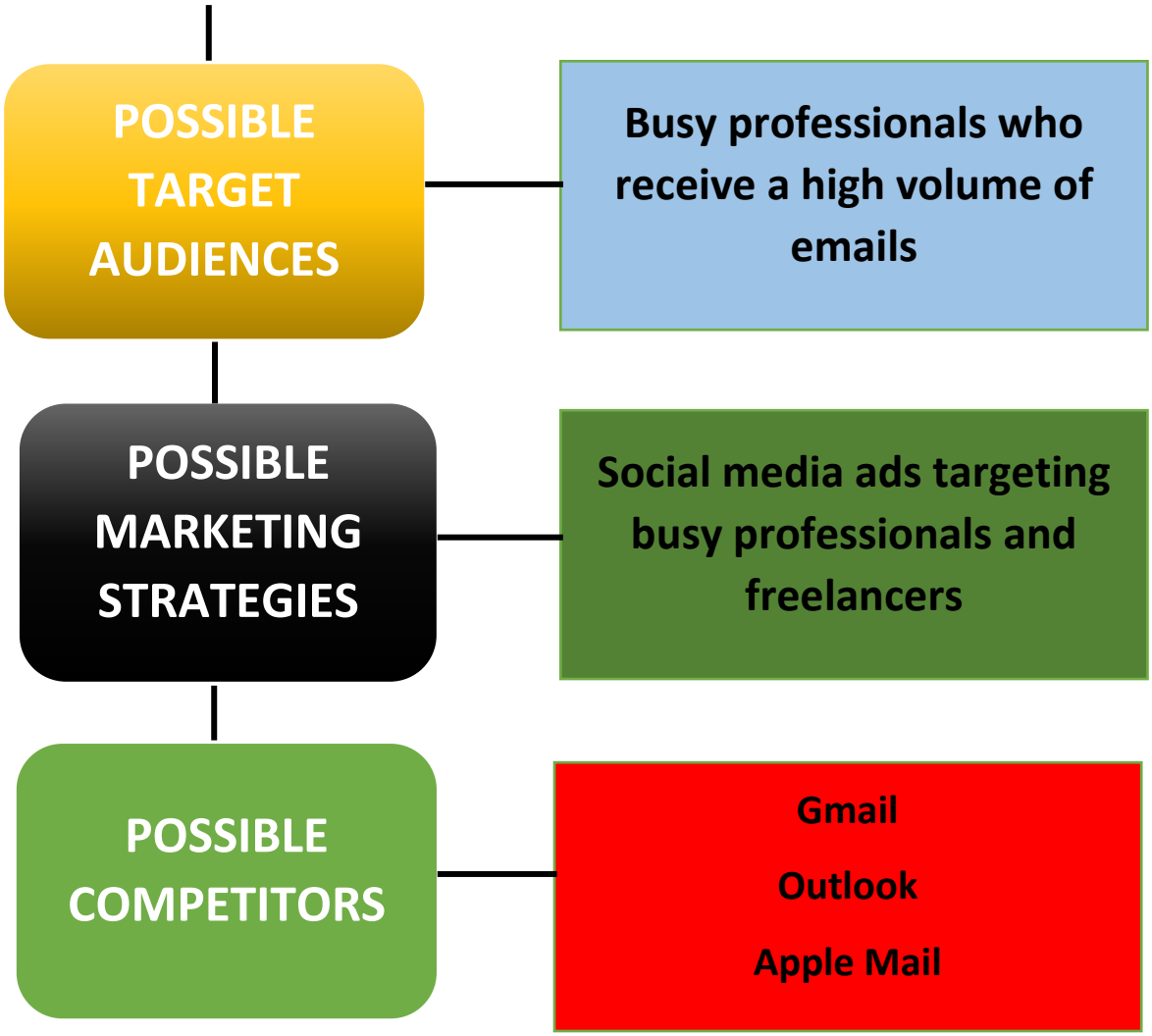
2.PROBLEM DEFINITION & DESIGN THINKING

2.1 EMPATHY MAP:



2.2 IDEATION & BRA MAP:




POSSIBLE TARGET AUDIENCES
Busy professionals who receive a high volume of emails
POSSIBLE MARKETING STRATEGIES
Social media ads targeting busy professionals and freelancers
POSSIBLE COMPETITORS
Gmail
Outlook
Apple Mail

3.RESULT:

11:45

Register

Username

Email

Password

Register

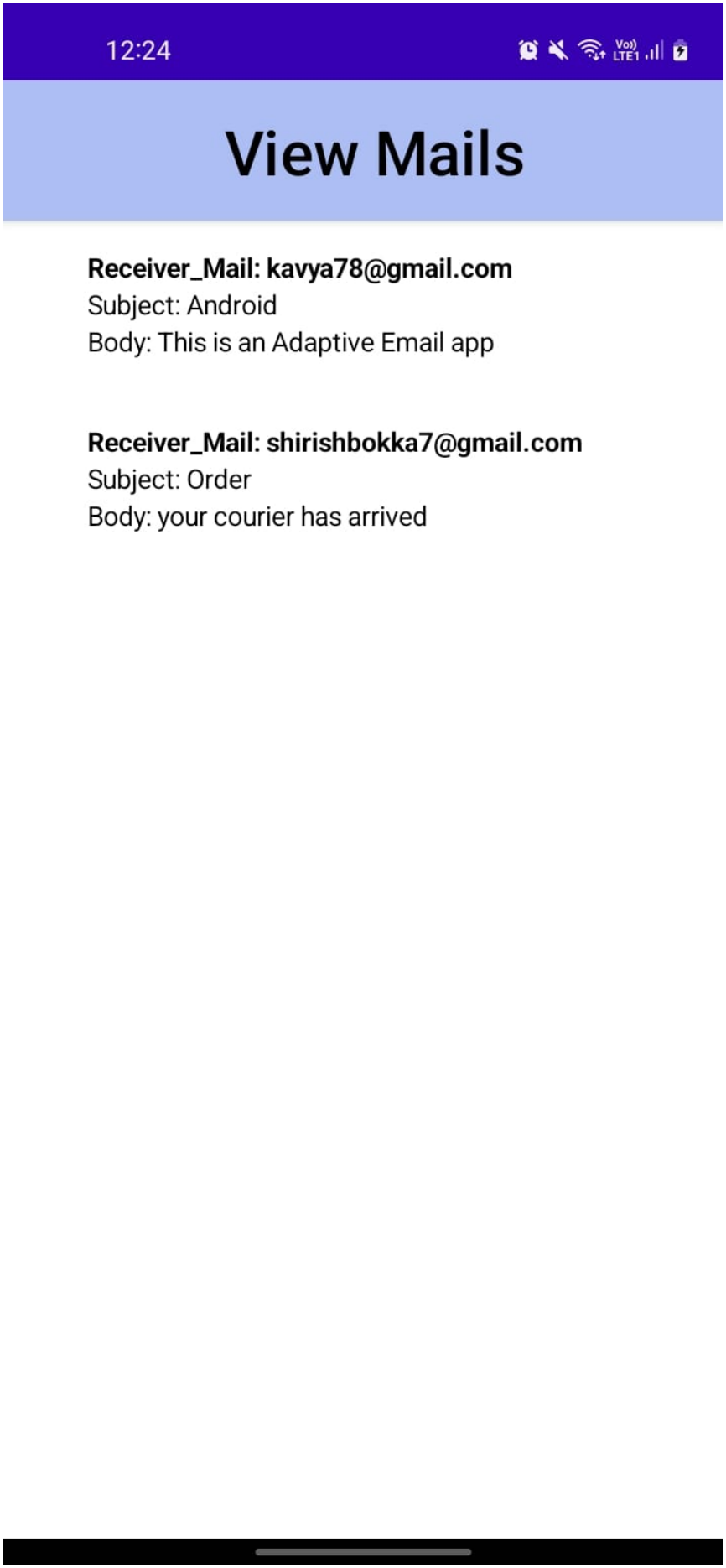
Have an account? Log in

11:46

Home Screen

Send Email

View Emails


12:24
View Mails
Receiver_Mail: kavya78@gmail.com
Subject: Android
Body: This is an Adaptive Email app
Receiver_Mail: shirishbokka7@gmail.com
Subject: Order
Body: your courier has arrived

4. ADVANDAGES & DISADVANTAGES:

4.1 ADVANDAGE:

The advantages of Adaptive Mail as a flexible email client app are:

Personalization: Adaptive Mail is designed to adapt to the user's behaviour and preferences over time, learning from their actions to offer personalized suggestions and automate tasks. This adaptability helps users to manage their emails more efficiently and effectively.

Customization: The app is highly customizable, allowing users to personalize the app to their preferences and workflow. This customization helps users to work more efficiently and comfortably.

Automation: Adaptive Mail offers automation for sorting and archiving emails, which saves time and reduces the need for manual sorting and labelling.

Cross-platform access: The app provides cross-platform access, allowing users to access their emails on both desktop and mobile devices, making it a convenient choice for users who need to stay connected on-the-go.

Integration: Adaptive Mail can integrate with other productivity tools, making it a powerful tool for users who need to manage multiple tasks simultaneously.

Personalized notifications: The app offers push notifications for important emails, ensuring that users never miss important messages.

Overall, the advantages of Adaptive Mail make it a powerful tool for busy professionals, freelancers, college students, and small business owners who need to manage their emails efficiently and effectively.

4.2 DISADVANDAGE:

Based on the features proposed for Adaptive Mail, it is difficult to identify clear disadvantages without actual user feedback and testing. However, here are some possible challenges that the app may face:

Learning curve: The app's flexibility and customization options may make it more challenging for some users to learn how to use the app effectively. It may take some time for users to fully utilize the app's features and capabilities.

Cost: Depending on the pricing model, the app may be more expensive compared to other email clients, which may discourage some potential users from trying it out.

Compatibility: The app may have compatibility issues with certain email providers, which may limit its functionality for some users.

Security: If the app is not designed with proper security measures, it may put users' sensitive information at risk, which can be a significant disadvantage.

Integration: Although integration with other productivity tools is an advantage, it may also pose a challenge if the app is not able to integrate with certain tools that users need.

It is important to note that these are potential challenges, and actual user feedback and testing will be necessary to determine any actual disadvantages or challenges that the app may face.

5.APPLICATIONS:

Adaptive Mail's flexible and personalized approach to email management can be useful for a wide range of applications, including:

Personal use: Adaptive Mail can help individuals better manage their personal email accounts by offering personalized suggestions, customizable folders, and automation. Users can also integrate the

app with other productivity tools they use in their personal life, such as calendars and to-do lists.

Business use: Busy professionals, freelancers, and small business owners can benefit from Adaptive Mail's personalized suggestions, automation, and cross-platform access. Customizable folders and labels can also help to keep emails organized and accessible, while the integration with other productivity tools can improve overall productivity and efficiency.

Academic use: College students can use Adaptive Mail to manage their academic and personal emails in one place, with personalized suggestions and automation helping them to stay on top of deadlines and communications.

Team use: Small businesses can use Adaptive Mail to improve team communication and collaboration by sharing folders and labels, and integrating with other productivity tools such as project management software.

Overall, Adaptive Mail's flexibility and personalized approach to email management make it a useful tool for a wide range of applications.

6.CONCLUSION:

In conclusion, Adaptive Mail is a hypothetical email client app that offers a personalized and flexible approach to email management. The app's features such as personalized suggestions, customizable folders, automation, and integration with other productivity tools can help users to manage their emails efficiently and save time.

The app's target audiences include busy professionals, freelancers, college students, and small business owners who need to manage their emails effectively. The suggested marketing strategies such as social media ads and influencer marketing can help to reach these audiences effectively.

While there may be potential challenges such as a learning curve and compatibility issues, the advantages of Adaptive Mail outweigh these potential disadvantages. The app can be useful for personal, business, academic, and team use, making it a versatile and valuable tool for email management.

Overall, Adaptive Mail has the potential to be a competitive and popular email client app if executed effectively and continuously improved based on user feedback.

7. FUTURE SCOPE:

The future scope of Adaptive Mail as a flexible email client app is promising. Here are some potential areas of development and growth:

Artificial Intelligence: As AI technology continues to advance, Adaptive Mail can integrate more AI capabilities to provide even more personalized and automated email management solutions. This can include more advanced natural language processing, sentiment analysis, and predictive analytics.

Security: With the increasing importance of email security, Adaptive Mail can focus on improving its security features to ensure user data is protected from cyber threats.

Voice Commands: With the growing popularity of voice assistants like Amazon Alexa and Google Home, Adaptive Mail can explore the possibility of incorporating voice commands to make email management more hands-free and accessible.

Advanced Customization: Adaptive Mail can continue to develop its customization features to allow users to tailor the app to their specific needs and preferences even more. This can include custom templates, themes, and more advanced filtering options.

Collaborative Features: Adaptive Mail can explore ways to improve team collaboration by integrating more collaborative features such as team folders, shared labels, and real-time editing.

Overall, the future of Adaptive Mail as a flexible email client app looks bright, with many opportunities for growth and development. By continuously improving its features and capabilities, and staying ahead of email management trends, Adaptive Mail can remain a competitive and valuable tool for users.

8. APPENDIX:

AndroidManifest.xml

```
<?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="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/Theme.EmailApplication"
        tools:targetApi="31" >
```

```
<activity
    android:name=".RegisterActivity"
    android:exported="false"
    android:label="@string/title_activity_register"
    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="@string/title_activity_view_mail"
    android:theme="@style/Theme.EmailApplication" />
<activity
    android:name=".SendMailActivity"
    android:exported="false"
    android:label="@string/title_activity_send_mail"s
    android:theme="@style/Theme.EmailApplication" />
<activity
    android:name=".LoginActivity"
```

```
        android:exported="true"
        android:label="@string/app_name"
        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>
```

Email.kt

```
package com.example.emailapplication

import androidx.room.ColumnInfo
import androidx.room.Entity
import androidx.room.PrimaryKey

@Entity(tableName = "email_table")
data class Email(
```

```
    @PrimaryKey(autoGenerate = true) val id: Int?,
    @ColumnInfo(name = "receiver_mail") val recevierMail: String?,
    @ColumnInfo(name = "subject") val subject: String?,
    @ColumnInfo(name = "body") val body: String?,
)
```

EmailDao.kt

```
package com.example.emailapplication

import androidx.room.*

@Dao
interface EmailDao {

    @Query("SELECT * FROM email_table WHERE  subject= :subject")
    suspend fun getOrderBySubject(subject: String): Email?

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertEmail(email: Email)

    @Update
    suspend fun updateEmail(email: Email)
```

```
    @Delete
    suspend fun deleteEmail(email: Email)
}
```

EmailDatabase.kt

```
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
            }
        }
    }
}
```

EmailDatabaseHelper.kt

```
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
```

```
    }


}
```

LoginActivity.kt

```
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.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
com.example.emailapplication.ui.theme.EmailApplicationTheme

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.White),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {
        Image(
            painterResource(id = R.drawable.email_login),
contentDescription = ""
        )



        Text(
            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.colors.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 = "Successfully log in"
                    context.startActivity(
                        Intent(
                            context,
                            MainActivity::class.java
                        )
                    )
                    //onLoginSuccess()
                }
```

```
            } else {
                error = "Please fill all fields"
            }
        },
        colors = ButtonDefaults.buttonColors(backgroundColor =
Color(0xFFd3e5ef)),
        modifier = Modifier.padding(top = 16.dp)
    ) {
        Text(text = "Login")
    }
    Row {
        TextButton(onClick = {context.startActivity(
            Intent(
                context,
                RegisterActivity::class.java
            )
        )}
        )
        { Text(color = Color(0xFF31539a),text = "Sign up") }
        TextButton(onClick = {
        })
```

```
        {
            Spacer(modifier = Modifier.width(60.dp))
            Text(color = Color(0xFF31539a),text = "Forget password?")
        }
      }
   }
}
private fun startMainPage(context: Context) {
   val intent = Intent(context, MainActivity::class.java)
   ContextCompat.startActivity(context, intent, null)
}
```

MainActivity.kt

```
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.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

class MainActivity : ComponentActivity() {
   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      setContent {
            // A surface container using the 'background' color from the
theme
```

```
            Surface(
                modifier =
Modifier.fillMaxSize().background(Color.White),
            ) {
                Email(this)
            }


        }
    }
}


@Composable
fun Email(context: Context) {
    Text(
        text = "Home Screen",
        modifier = Modifier.padding(top = 74.dp, start = 100.dp, bottom
= 24.dp),
        color = Color.Black,
        fontWeight = FontWeight.Bold,
        fontSize = 32.sp
    )
```

```
    Column(
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {
        Image(
            painterResource(id = R.drawable.home_screen),
contentDescription = ""
        )



        Button(onClick = {
            context.startActivity(
                Intent(
                    context,
                    SendMailActivity::class.java
                )
            )
        },
            colors = ButtonDefaults.buttonColors(backgroundColor =
Color(0xFFadbef4))
        ) {
            Text(
                text = "Send Email",
```

```
            modifier = Modifier.padding(10.dp),
            color = Color.Black,
            fontSize = 15.sp
        )
    }

    Spacer(modifier = Modifier.height(20.dp))

    Button(onClick = {
        context.startActivity(
            Intent(
                context,
                ViewMailActivity::class.java
            )
        )
    },
        colors = ButtonDefaults.buttonColors(backgroundColor =
Color(0xFFadbef4))
    ) {
        Text(
            text = "View Emails",
            modifier = Modifier.padding(10.dp),
```

```
            color = Color.Black,
            fontSize = 15.sp
        )
    }


  }
}
```

RegisterActivity.kt

```
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.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
com.example.emailapplication.ui.theme.EmailApplicationTheme

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.colors.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(backgroundColor =
Color(0xFFd3e5ef)),
    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 startLoginActivity(context: Context) {
  val intent = Intent(context, LoginActivity::class.java)
  ContextCompat.startActivity(context, intent, null)
}
```

SendMailActivity.kt

```
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.layout.*
import androidx.compose.material.*
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

class SendMailActivity : ComponentActivity() {
```

```
    private lateinit var databaseHelper: EmailDatabaseHelper
    @SuppressLint("UnusedMaterialScaffoldPaddingParameter")
    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(backgroundColor = 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 = "Send Mail",
                    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,
                )
            }
        )
    }
) {
    // 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
    ) {

        // 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(backgroundColor =
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.kt

```
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?,
```

```
)
```

UserDao.kt

```
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 fun insertUser(user: User)

    @Update
    suspend fun updateUser(user: User)

    @Delete
    suspend fun deleteUser(user: User)
}
```

UserDatabase.kt

```
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
        }
    }
}
```

UserDatabaseHelper.kt

```
        if (cursor.moveToFirst()) {
            do { 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, DATABASE_NAME, null,
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)

            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
  }

}
```

ViewMailActivity.kt

```
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.Image
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.layout.R
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.runtime.Composable
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.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

class ViewMailActivity : ComponentActivity() {
    private lateinit var emailDatabaseHelper: EmailDatabaseHelper
    @SuppressLint("UnusedMaterialScaffoldPaddingParameter")
```

```
    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(backgroundColor = 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("Receiver_Mail: ${email.recevierMail}",
fontWeight = FontWeight.Bold)
                        Text("Subject: ${email.subject}")
                        Text("Body: ${email.body}")
                    }
                }
            }
        }

    }
```

}