ADAPTIVE MAIL

A FLEXIBLE EMAIL CLIENT APP

A PROJECT

Submitted by

K.ADHILAKSHMI: 20201231506202

J.BENCIYA: 20201231506207

R.DIVYA: 20201231506209

R.ESAKKIAMMAL: 20201231506211

MENTOR
Dr.T.ARUL RAJ M.Sc.,M.Phil.,Ph.D

in partial fulfillment of the requirements for the award of degree of

BACHELOR OF SCIENCE (COMPUTER SCIENCE)

SRI PARAMAKALYANI COLLEGE
ALWARKURICHI – 627 412
APRIL - 2023

PROJECT INDEX

1.INTRODUCTION

1.1 Overview

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

- Adaptive Mail is an email client application that is designed to be flexible and adaptable to the needs of individual users. The app allows users to customize the appearance and functionality of the app according to their preferences, making it easier for them to manage their email workflow.

- One of the key features of Adaptive Mail is its unified inbox, which aggregates all the emails from different accounts in one place. The app supports multiple email providers, including Gmail, Yahoo, and Microsoft Office365, among others.

- With Adaptive Mail, users can organize their emails into different folders, create filters to prioritize or exclude certain messages, and set up automatic replies or forwarding rules. The app also offers a range of customization options, such as choosing the color scheme, layout, and fonts, to make it more visually appealing and user-friendly.

- Furthermore, Adaptive Mail provides advanced search options that allow users to quickly find specific emails, attachments, or contacts with ease. The app also has strong security features, including encrypted connections and two-factor authentication for added protection against hacking or phishing attacks.

- Overall, Adaptive Mail offers a flexible and customizable email management solution that can help users streamline their workflow and increase their productivity.

1.2 Purpose

- The purpose of Adaptive mail, a flexible email client app, is to provide users with a customizable and user-friendly experience for managing their email accounts. The app is designed to adapt to the unique needs and preferences of each user, allowing them to tailor the app to fit their workflow and make it easier to manage their email.

- The app's integration with multiple email providers enables users to manage all of their email accounts in one place, which can help them stay organized and productive. The flexible user interface allows users to customize the app's layout and appearance, making it easier to find the features and tools they need.

- Adaptive mail also includes productivity tools, such as snooze emails, reminders, and custom filters and rules, which can help users manage their inbox more efficiently. Additionally, the app includes robust security features, such as end-to-end encryption and two-factor authentication, to protect users' sensitive information.

2.PROBLEM DEFINITION & DESIGN THINKING

2.1 Empathy Map

Says:

“I have so many emails to manage ,it’s overwhelming”.

“I need an app that integrates with all my email accounts”.

Thinks:

“I need to stay on top of my emails to be productive”.

“I don’t want to worry about security risks when using email”.

USER

Feels:

Overwhelmed by the number of emails in their box.

Frustrated with inefficient email management.

Does:

Tries to organize their inbox manually but it takes too much time.

Switches between different email apps to manage multiple accounts.

2.2 Ideation And Brainstorming Map

3. RESULT

Goal

- **Create an innovative client app**

Features

- **Productivity tools**
- **Robust Security**

Adaptive Mail A Flexible Email Client

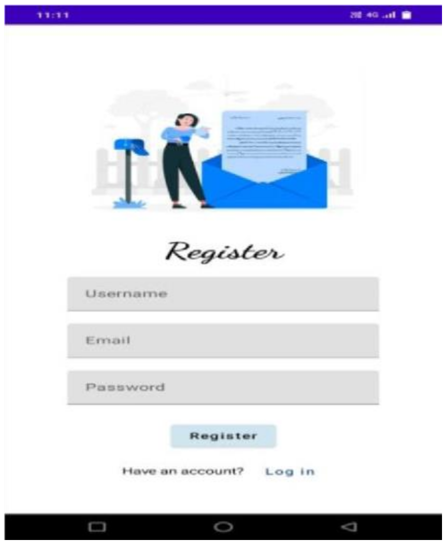
Ideas

- **Ability to schedule emails for sending later**

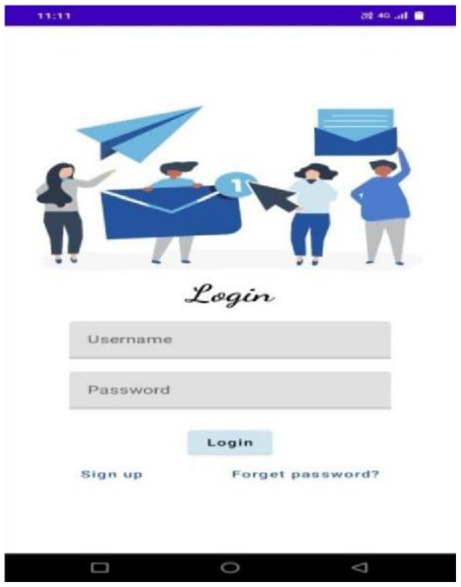
Potential Benefits

- **Better Organization and priotritization of emails**

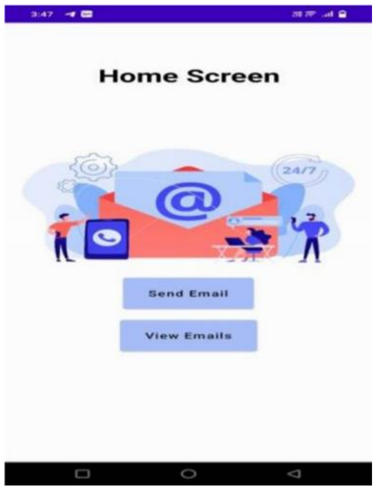
Register page



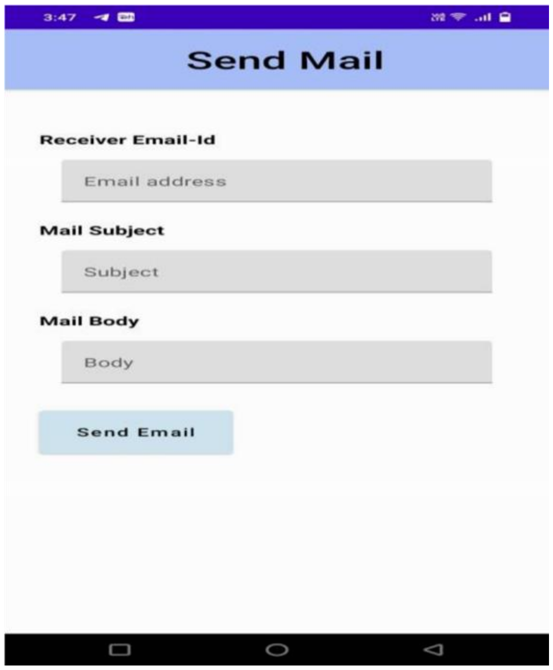
Login Page



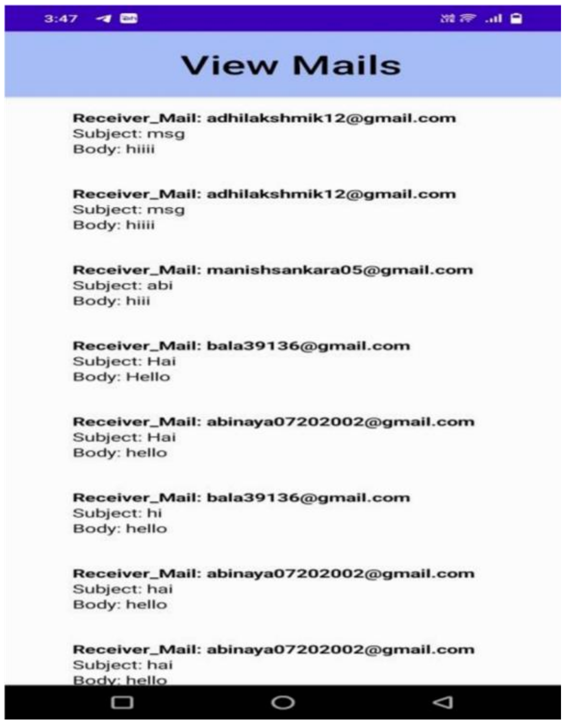
Home Screen



Send Mail Page



View Mails Page



Receiver_Mail: adhilakshmik12@gmail.com
Subject: msg
Body: hiiii

Receiver_Mail: adhilakshmik12@gmail.com
Subject: msg
Body: hiiii

Receiver_Mail: manishsankara05@gmail.com
Subject: abi
Body: hiii

Receiver_Mail: bala39136@gmail.com
Subject: Hai
Body: Hello

Receiver_Mail: abinaya07202002@gmail.com
Subject: Hai
Body: hello

Receiver_Mail: bala39136@gmail.com
Subject: hi
Body: hello

Receiver_Mail: abinaya07202002@gmail.com
Subject: hai
Body: hello

Receiver_Mail: abinaya07202002@gmail.com
Subject: hai
Body: hello

4. ADVANTAGES AND DISADVANTAGES

4.1 Advantages

- **Customize Interface :**
  Adaptive mail provides a customizable interface that can be personalized according to the user's preferences. users can choose the theme, layout, and features that suit their needs.

- **Compatibility:**
  Adaptive mail is compatible with various email services like gmail, yahoo, and outlook, which makes it easier for users to manage all their emails from a single app.

- **Advanced Filters:**

  Adaptive mail offers advanced filtering options that allow users to sort their emails based on various criteria like sender, subject, date, and size. this feature helps users quickly find the emails they need.

- **Security:**

  Adaptive mail takes security seriously and offers encryption, two-factor authentication, and other security features to protect users' emails from hackers and other threats.

- **Cross-Platform Compatibility:**

  Adaptive mail is available on multiple platforms like android, ios, windows, and mac, which makes it easy for users to access their emails from any device.

- **Integration:**

  Adaptive mail integrates with other productivity apps like trello, asana, and slack, which helps users manage their tasks and workflows more efficiently.

- **Artificial Intelligence:**

  Adaptive mail uses artificial intelligence to help users manage their emails more efficiently. the app can learn from the user's behavior and suggest replies, categorize emails, and perform other tasks to save time and increase productivity

4.2 Disadvantages

- **Learning Curve:**

  Because Adaptive mail offers many customizable features, there may be a learning curve for users who are not tech-savvy. It may take some time to learn how to use all of the features effectively.

- **Cost:**
  Adaptive mail may come with a cost to access some of its advanced features or to remove ads from the app. This may be a drawback for some users who are looking for a free email client app.

- **Privacy Concerns:**
  As with any email client app, there may be privacy concerns with Adaptive mail. Users need to ensure that their emails are secure and that their data is not being shared with third-party advertisers.

- **Compatibility:**
  While Adaptive mail is compatible with many email services, it may not be compatible with all of them. This could be a disadvantage for users who rely on a particular email service that is not supported.

- **Dependence on Internet Connection:**
  Since Adaptive mail is an online app, it requires a stable internet connection. Users may not be able to access their emails if they do not have an internet connection, which could be a significant disadvantage for users who are frequently on the go or in areas with poor connectivity.

5. APPLICATIONS

- **Personal Email Management:**
  Adaptive mail can be used to manage personal emails from various email services like Gmail, Yahoo, and Outlook. The advanced filtering options and artificial intelligence features can help users sort and manage their emails more efficiently.

- **Business Email Management:**

  Adaptive mail can also be used to manage business emails from multiple email accounts. The integration with other productivity apps like Trello, Asana, and Slack can help users manage their tasks and workflows more efficiently.

- **Marketing:**

  Adaptive mail can be used for marketing purposes, like sending newsletters and promotional emails to customers. The app's analytics features can help users track the performance of their email campaigns and make adjustments accordingly.

- **Customer Service:**

  Adaptive mail can be used for customer service purposes, like responding to customer inquiries and resolving issues. The app's artificial intelligence features can help automate responses and speed up the resolution process.

- **Education:**

  Adaptive mail can be used in educational settings, like for student-teacher communication and class announcements. The app's advanced filtering options can help educators sort and manage emails from multiple students more efficiently.

6. CONCLUSION

- In conclusion, Adaptive mail is a flexible email client app that offers several advantages for personal and business use. Its customizable interface, advanced filtering options, security features, cross-platform compatibility, integration with other apps, and artificial intelligence capabilities make it a highly efficient and productive tool for managing emails.

- However, there are also some potential disadvantages to using Adaptive mail, including a learning curve, cost, privacy concerns, compatibility issues, and dependence on an internet connection.

- Overall, Adaptive mail is a powerful tool that can be used for various applications, including personal and business email management, marketing, customer service, education, and research. It is important for users to weigh the pros and cons before deciding whether or not to use Adaptive mail and to take steps to ensure their emails are secure and protected.

7.FUTURE SCOPE

✞ The future scope of Adaptive mail, a flexible email client app, looks promising. Here are some potential future developments and advancements that could be made in the app:

- **Integration with AI :**

  The app's current AI features are already impressive, but there is room for even more advancement. As AI technology improves, Adaptive mail could incorporate even more sophisticated features, such as predicting which emails are most important to a user and automatically scheduling responses.
- **Further Customization Options:**

  Adaptive mail's current customization options are already vast, but there is always room for more. The app could continue to expand its customization options to allow users to tailor their email management experience even further.

- **Greater Collaboration Capabilities:**

  The app already offers integration with other productivity apps, but it could expand its collaboration capabilities even further. This could include features such as real-time collaboration on emails, shared email inboxes, and integration with video conferencing software.

- **Improved Security Features:**

  As email security continues to be a growing concern, Adaptive mail could improve its security features to ensure that emails are protected from phishing attacks, malware, and other online threats.

✞ The future scope of Adaptive mail looks promising, with potential advancements in AI, customization options, collaboration capabilities, security features, and integration with wearable devices. As technology continues to evolve, Adaptive mail will likely continue to adapt and improve, making it an even more valuable tool for managing emails efficiently and effectively.

8.APPENDIX

8.1 Source code

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

```
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)
        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
    }

}
```

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.backgro
und                                  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.ContentS
cale                             import
androidx.compose.ui.res.painterResou
rce                              import
androidx.compose.ui.text.font.FontFa
mily                             import
androidx.compose.ui.text.font.FontWe
ight
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)
}
```

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.isNotEmp
ty()                  &&
password.isNotEmpt
y() &&
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)
}
```

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.Previe
w import androidx.compose.ui.unit.dp import
androidx.compose.ui.unit.sp              import
androidx.core.content.ContextCompat  import
androidx.core.content.ContextCompat.startA
ctivity
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
            )
        }



    }
}
```

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
            )
        }
    }
}
```

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}")
               }
            }
          }
          }
      }
```