PROJECT REPORT

Adaptive Mail : A Flexible Email Client App

1. INTRODUCTION

1.1 Overview

An email application, also known as an email client, is a software program that allows users to send, receive, and manage emails. These applications can be web-based or desktop-based and can be accessed on a range of devices including smartphones, tablets, and computers.

Email applications provide users with various features and functionalities, including composing and sending emails, organizing and managing emails, creating and managing contacts, and setting up filters and rules for incoming emails. They also offer the ability to attach files and documents to emails, and in some cases, schedule emails to be sent at a later time.

1.2. Purpose

The purpose of an email application is to allow users to send, receive, and manage electronic messages over the internet. Email applications can be used for personal communication, business communication, or any other purpose that requires the exchange of written information.

2. PROBLEM DEFINITION & DESIGN THINKING

2.1 Empathy Map

Template

Empathy map

Use this framework to develop a deep, shared understanding and empathy for other people. An empathy map helps describe the aspects of a user's experience, needs and pain points, to quickly understand your users' experience and mindset.

Share template feedback

Need some inspiration?

See a finished version of this template to kickstart your work.

Open example →

Build empathy

The information you add here should be representative of the observations and research you've done about your users.

Says

What have we heard them say?
What can we imagine them saying?

"Hello, would you like me to prioritize your inbox based on your most frequently contacted senders?"

"You have a new email from [sender's name]. Would you like me to read it to you?"

"Your email draft has been saved as a draft. Would you like me to remind you to finish it later?"

Thinks

What are their wants, needs, hopes, and dreams? What other thoughts might influence their behavior?

"Based on the user's email history, I think they would prioritize emails from these senders."

"The user often replies to emails from their phone. I think they would appreciate an easier-to-use mobile interface."

"This email has a lot of keywords related to a certain topic. I think the user might want to label it under that category."

Adaptive Mail : A Flexible Email Client App

Does

What behavior have we observed?
What can we imagine them doing?

Prioritizes the user's inbox based on frequently contacted senders.

Saves email attachments to the user's cloud storage.

Reads emails out loud to the user upon request.

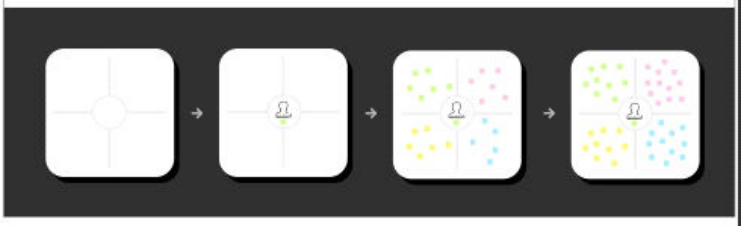
Feels

What are their fears, frustrations, and anxieties? What other feelings might influence their behavior?

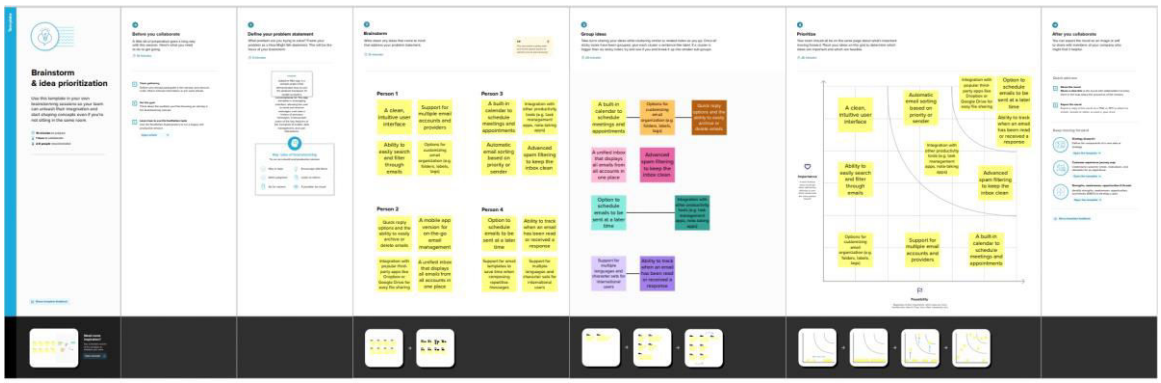
Efficient: Adaptive mail strives to make the user's email experience as streamlined as possible.

Helpful: Adaptive mail seeks to assist the user in any way possible.

Intuitive: Adaptive mail aims to anticipate the user's needs and tailor the email experience accordingly.



2.2 Ideation & Brainstorming Map



3.RESULT

Login Page

Login

Username

Password

Login

Sign up Forget password?

Register Page

Register

Username

Email

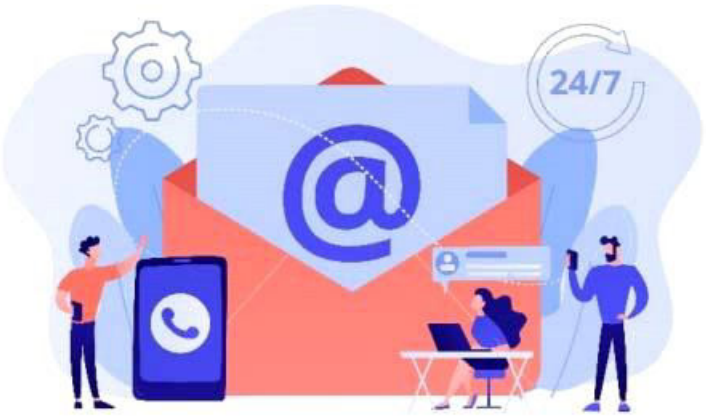
Password

Register

Have an account? Log in

Main News Headlines Page

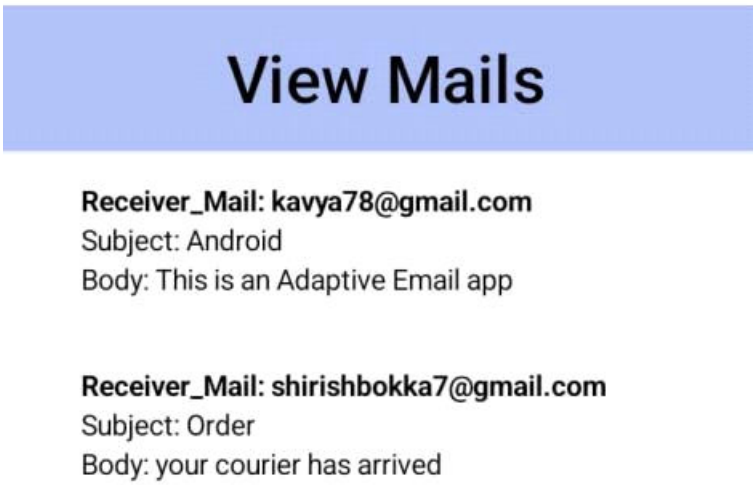
Home Screen



Send Email

View Emails

Display News Page



4. ADVANTAGES & DISADVANTAGES

- **Advantages**
    - Easy to use

- Email applications allow you to communicate instantly with people across the world, without having to wait for postal services to deliver your message.
- Email applications are typically free or low-cost, making them a cost-effective option for communication.

- **Disadvantages**
  - Email applications are vulnerable to security breaches and hacking.
  - Email applications can be inundated with spam emails.

• APPLICATIONS

The App can be used as personal app for individuals. The Email app are message app, many multimillion companies are using this email to share the information between them. The project will be useful to all the sections of the society.

• CONCLUSION

In conclusion, email applications are essential tools for communication in both personal and professional settings. They allow individuals to send and receive messages quickly and efficiently, organize their inbox, and automate repetitive tasks.

• FUTURE SCOPE

The future scope of email applications is vast and ever-evolving, as technology and user needs continue to advance.

The future of email applications looks promising, with advancements in technology and user needs driving innovation and improvements in the user experience.

• APPENDIX

// 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 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
    }

}
// 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 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
    }
```

```
}
// 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)
}

// 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)
}



// 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

            )

        }




    }

}



// 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}")
                    }
                }
            }
        }


    }
}
```

```
// AndroidManifest.kt

<?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"
    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>
```