Adaptive Mail: A Flexible Email Client App

**A .NOOR MAISARA**

**K. SABEEKA**

**A. NOORUL FAIZAH**

**T. NUR HATHIJA**

**K. AYESHA NASRIN**

**1 INTRODUCTION**

* 1. **OVERVIEW**

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

**PROJECT FLOW**

**● Users register into the application.**

**● After registration , user logins into the application.**

**● User enters into the main page**

**● User can View previously sent emails.**

**● User can give subject and email body to send email**

* 1. PURPOSE

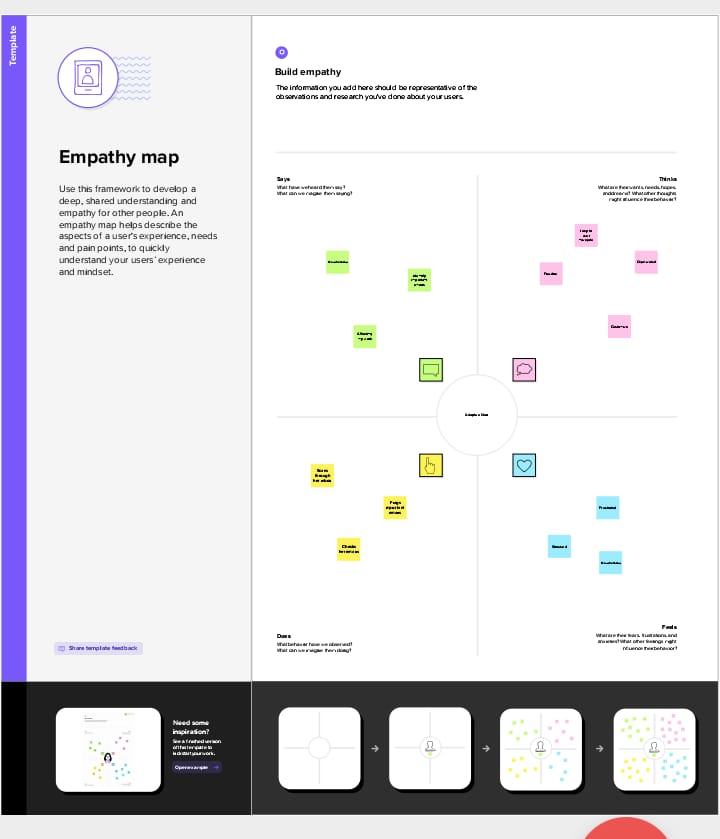
The purpose of an adaptive mail client app is to provide a flexible and customizable email experience to users. Unlike traditional email clients that offer limited customization options, adaptive mail clients can be tailored to meet the unique needs and preferences of individual users.

Adaptive mail apps may offer features such as customizable layouts, the ability to create custom tags and labels for organizing emails, and the ability to integrate with other productivity tools such as calendars and task managers. Additionally, adaptive mail clients may use machine learning algorithms to learn a user's email habits and prioritize messages accordingly, making it easier to manage a high volume of emails.

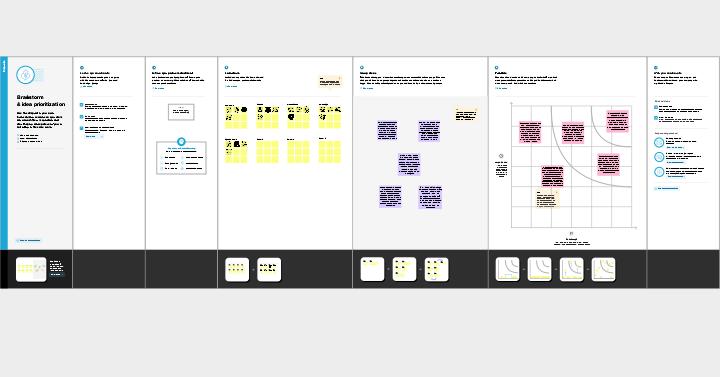
Overall, the goal of an adaptive mail client is to improve the email experience for users by providing them with greater control and customization over their inbox while also leveraging technology to streamline and automate certain aspects of email management.

2. PROBLEM DEFINITION & DESIGN THINKING

* 1. EMPATHY MAP

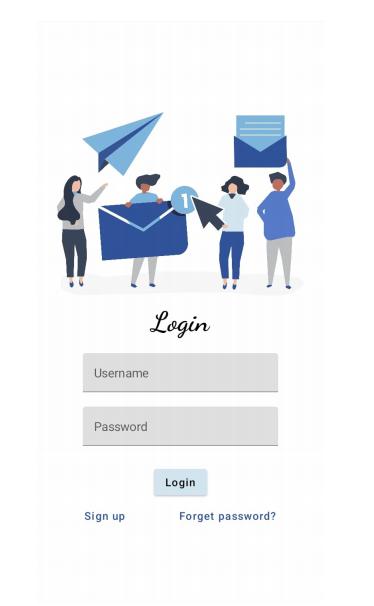


* 1. Ideation & brainstorming map

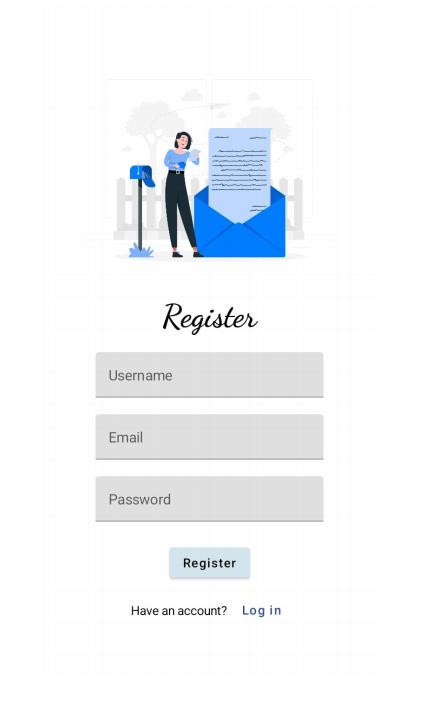


1. RESULT

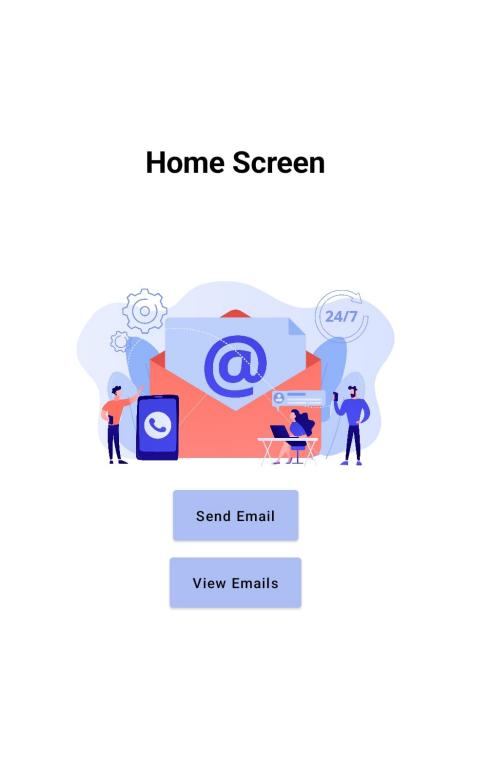
LOGIN PAGE:



REGISTER PAGE:



MAIN PAGE:



ADVANTAGES AND DISADVANTAGES

Advantages:

Customizable interface: Adaptive Mail allows users to customize the interface to suit their preferences, including changing the color scheme, layout, and font size.

Personalization: The app uses machine learning algorithms to personalize the email experience for each user, including categorizing emails based on importance and predicting which emails the user is likely to respond to.

Time-saving features: Adaptive Mail includes features that help users save time, such as quick reply options, automated email categorization, and the ability to schedule emails to be sent later.

Cross-platform compatibility: Adaptive Mail is available on multiple platforms, including iOS, Android, and desktop, making it easy for users to access their emails from anywhere.

Disadvantages:

Learning curve: Because Adaptive Mail is highly customizable and has many features, it may take some time for users to learn how to use all of its functions effectively.

Privacy concerns: Since the app uses machine learning algorithms to personalize the email experience, some users may be concerned about their data privacy.

Compatibility issues: Adaptive Mail may not be compatible with all email providers, so some users may need to switch email providers to use the app.

Cost: Adaptive Mail may not be a free app and may require a subscription, which may be a disadvantage for users who are looking for a free email client app.

1. APPLICATIONS

Personalized email management: An adaptive email client could use machine learning algorithms to learn a user's email management preferences and automatically categorize incoming messages into folders or tags. For example, if a user tends to file emails from certain senders or with certain keywords, the app could learn to do this automatically and save the user time.

Mobile optimization: Many people use their phones as their primary device for checking email, but not all email clients are optimized for mobile. An adaptive email client could adjust its user interface based on the device being used, making it easier to read and respond to emails on a small screen.

Multilingual support: If a user regularly receives emails in multiple languages, an adaptive email client could use natural language processing to automatically translate emails into the user's preferred language. This could save time and help ensure that important information isn't missed due to a language barrier.

Email automation: An adaptive email client could integrate with other productivity tools and automate certain email-related tasks. For example, it could automatically send follow-up emails after a certain amount of time, or automatically archive or delete messages that meet certain criteria.

Customizable interface: Different people have different preferences when it comes to email interfaces. An adaptive email client could allow users to customize the layout, color scheme, and other aspects of the app to suit their individual preferences.

1. CONCLUSION

In conclusion, an adaptive email client app has the potential to revolutionize the way we manage our email. By using machine learning algorithms and natural language processing, it can learn a user's preferences and adapt to their individual needs, making it easier and more efficient to manage emails. The app could also optimize for mobile devices, support multiple languages, automate certain email-related tasks, and allow for customization of the user interface. Overall, an adaptive email client app has the potential to save time, increase productivity, and improve the email experience for users.

1. FUTURE SCOPE

Integration with smart home devices: As more and more homes become equipped with smart devices, an adaptive email client app could integrate with these devices to allow users to manage their email using voice commands or other intuitive interfaces.

Enhanced security features: Email security is a growing concern, and an adaptive email client app could incorporate advanced security features such as two-factor authentication, encryption, and anti-phishing measures to help protect users from cyber threats.

Augmented reality (AR) integration: AR technology is becoming more prevalent in our daily lives, and an adaptive email client app could use AR to display email messages in an interactive and immersive way, making it easier to digest and understand complex information.

Collaboration tools: Many people use email as a means of collaborating with colleagues or working on projects with others. An adaptive email client app could incorporate collaboration tools such as shared calendars, project management features, and real-time editing capabilities to make it easier to work together.

Personalized content delivery: An adaptive email client app could use machine learning algorithms to deliver personalized content to users based on their interests and preferences. This could include curated newsletters, articles, and other content that is tailored specifically to the user's interests.

Overall, the future scope for an adaptive email client app is vast, and the potential for innovation and improvement is endless. As technology continues to advance and our needs and preferences evolve, an adaptive email client app has the potential to become an even more valuable and indispensable tool for managing our email.

1. APPENDIX
2. Source code

Database 1

Step 1: Create user data class

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?,

)

Step 2:

Create userdao interface

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)

}

Step 3: create an userdatabase class

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

}

}

}

Step 4: create an userdatabasehelper class

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

}

}

Database 2

Step 1: create email data class

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?,

)

Step 2: create emaildao class

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)

}

Step 3: create emaildatabase clas

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

}

}

}

}

Step 4: create databasehelper class

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

}

}

Step 1: creating loginactivity.kt with database

[3:32 pm, 14/04/2023] Hathija Clg: 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

}

}

[3:33 pm, 14/04/2023] Hathija Clg: 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)

}

Step 2: creating register activity.kt with database

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)

}

Step 3: 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

)

}

}

}

Step 4:

Creating sendmailactivity.kt file

mport 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

)

}

}

}

Step 5: creating viewmailactivity.kt file

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}")

}

}

}

}

}

}

MODIFYING ANDROIDMANIFEST.XML

<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:tools="http://schemas.android.com/tools" >

<application

android:allowBackup="true"

android:dataExtractionRules="@xml/data\_extraction\_rules"

android:fullBackupContent="@xml/backup\_rules"

android:icon="@mipmap/ic\_launcher"

android:label="@string/app\_name"

android:supportsRtl="true"

android:theme="@style/Theme.EmailApplication"

tools:targetApi="31" >

<activity

android:name=".RegisterActivity"

android:exported="false"

android:label="@string/title\_activity\_register"

android:theme="@style/Theme.EmailApplication" />

<activity

android:name=".MainActivity"

android:exported="false"

android:label="MainActivity"

android:theme="@style/Theme.EmailApplication" />

<activity

android:name=".ViewMailActivity"

android:exported="false"

android:label="@string/title\_activity\_view\_mail"

android:theme="@style/Theme.EmailApplication" />

<activity

android:name=".SendMailActivity"

android:exported="false"

android:label="@string/title\_activity\_send\_mail"

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>