Project Report Template

1.INTRODUCTION

* 1. Overview

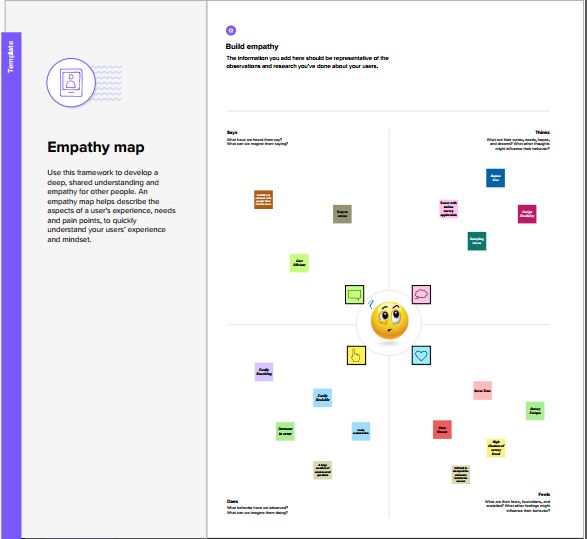
Adaptive Mail app is a sample project that demonstrates how to use the Android Compose UI toolkit to build a conversational UI. The app simulates a messaging interface, allowing the user to send and receive messages, and view a history of previous messages.

* 1. Purpose

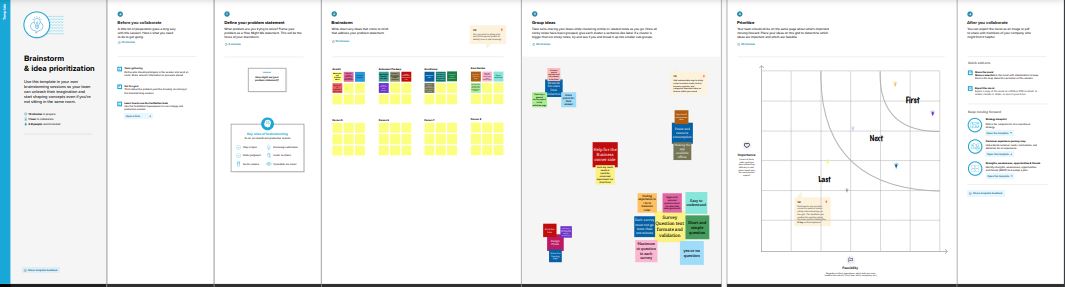
Email is used for many different purposes, including contacting friends, communicating with professors and supervisors, requesting information, and applying for jobs, internships, and scholarships. Depending on your purposes, the messages you send will differ in their formality, intended audience, and desired outcomes.

2.Problem Definition & Design Thinking

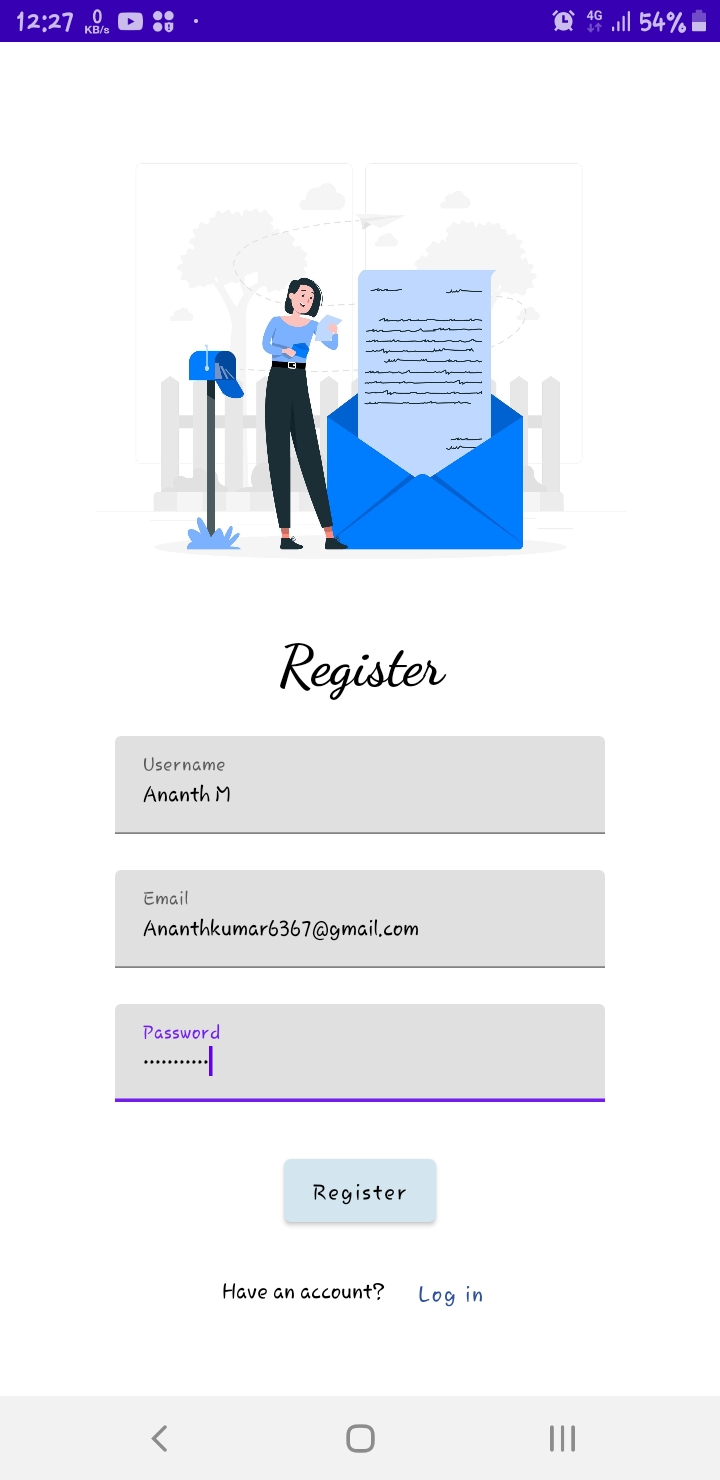
2.1 Empathy Map

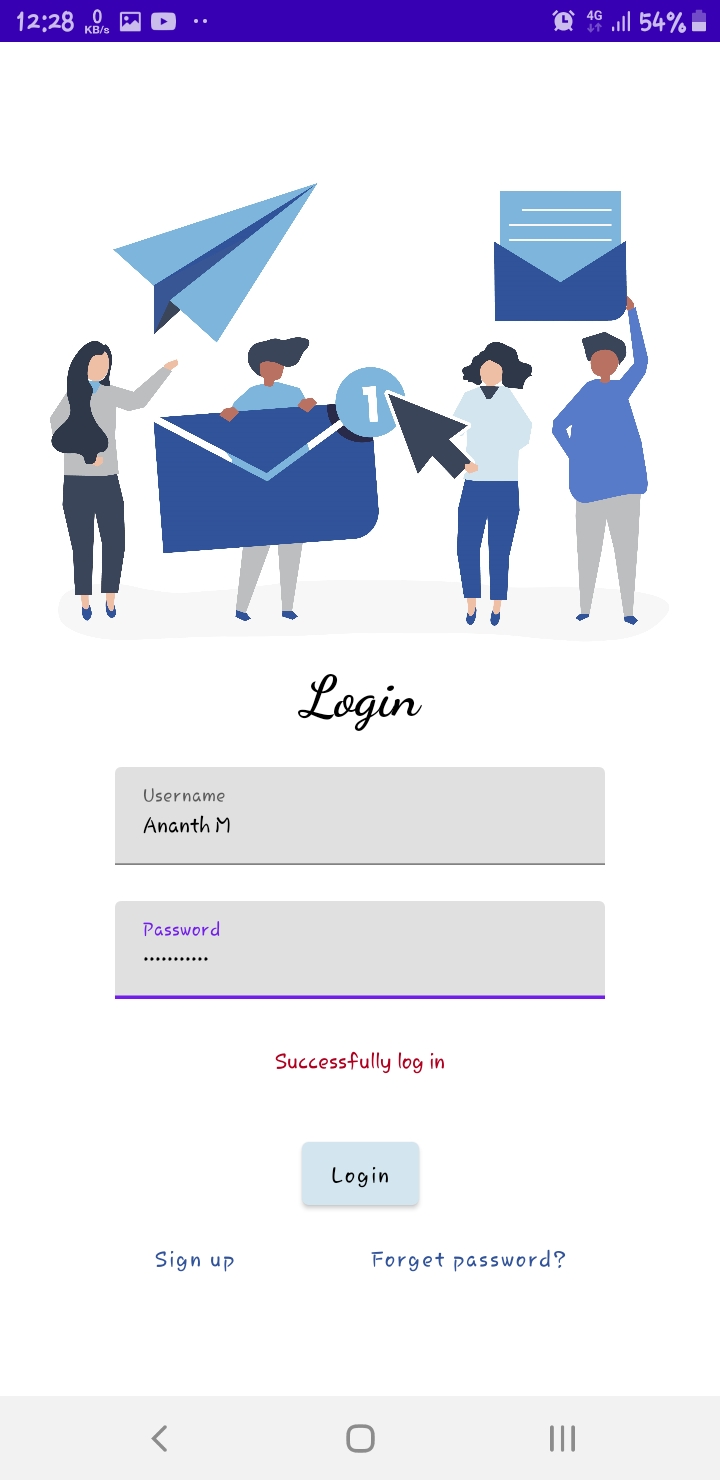
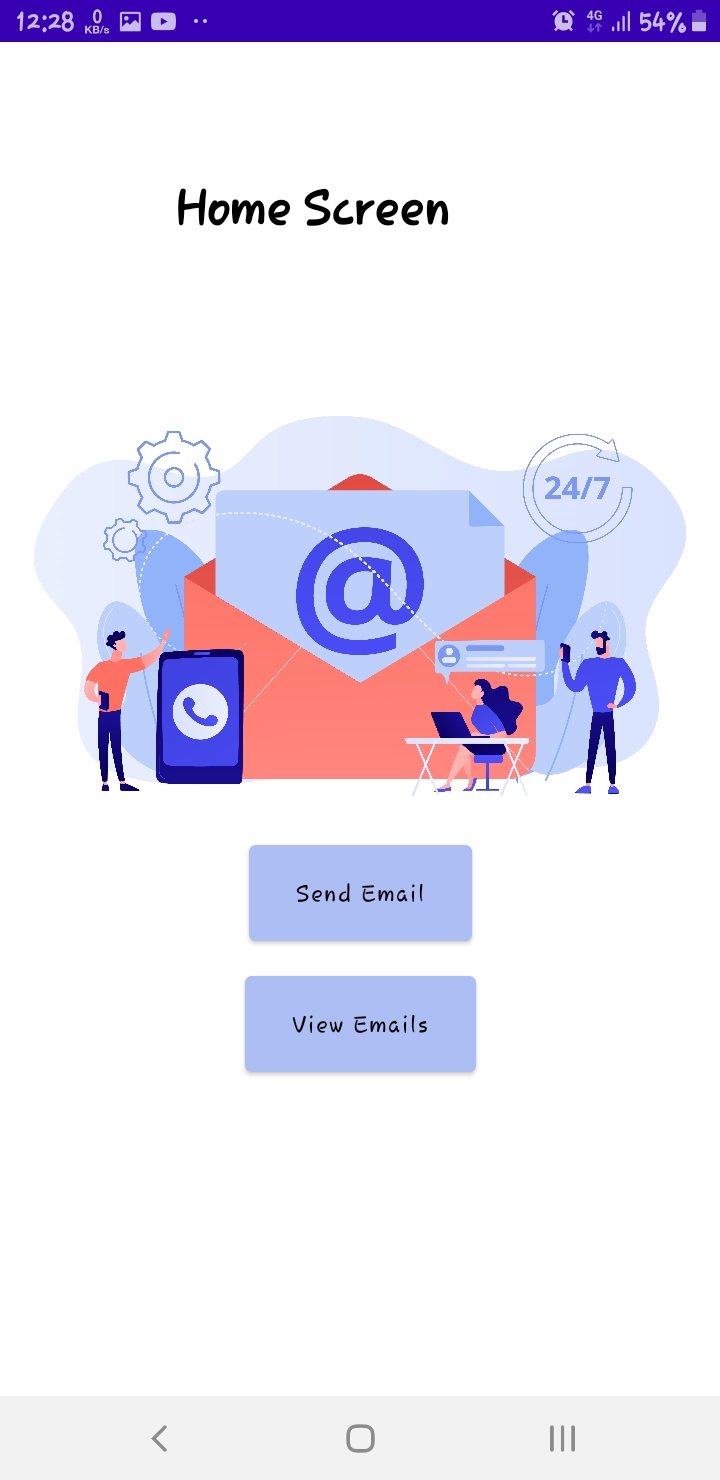


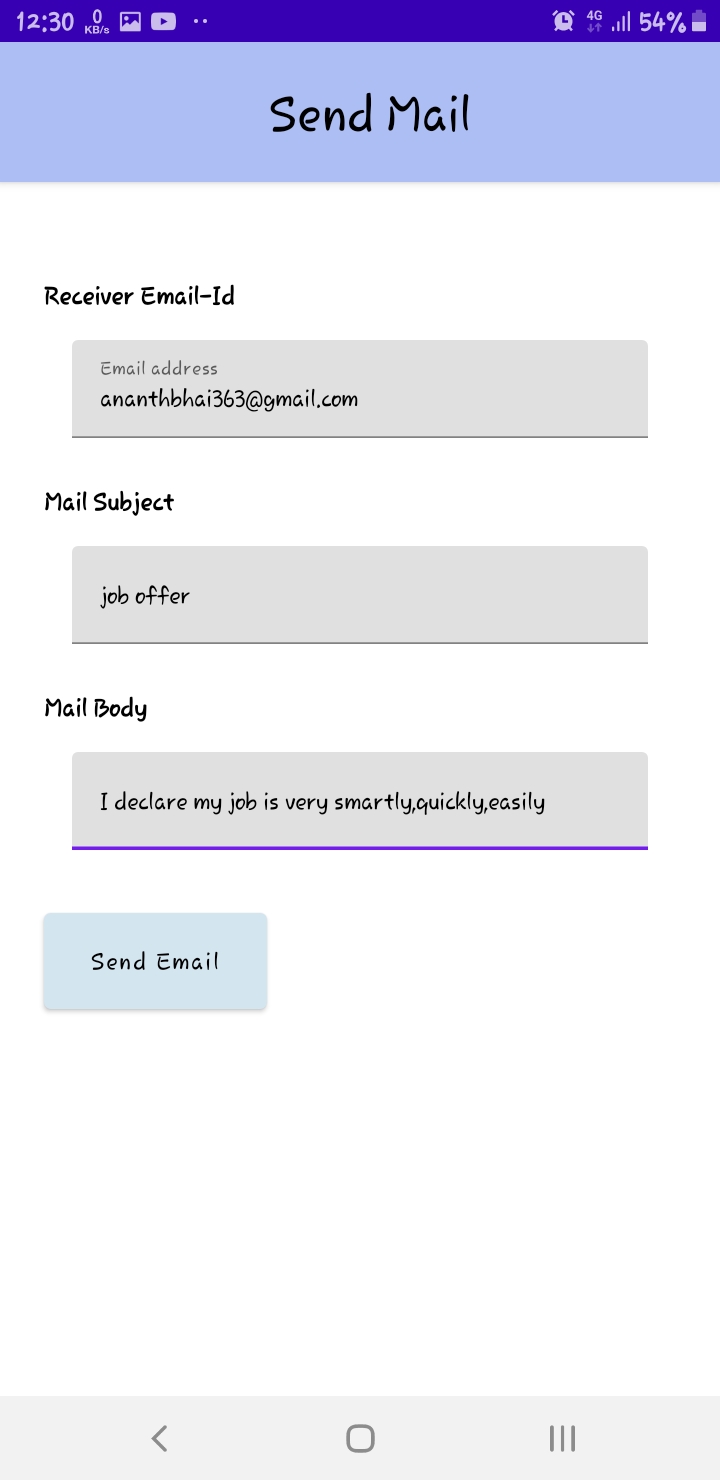
2.2 Ideation & Brainstorming Map

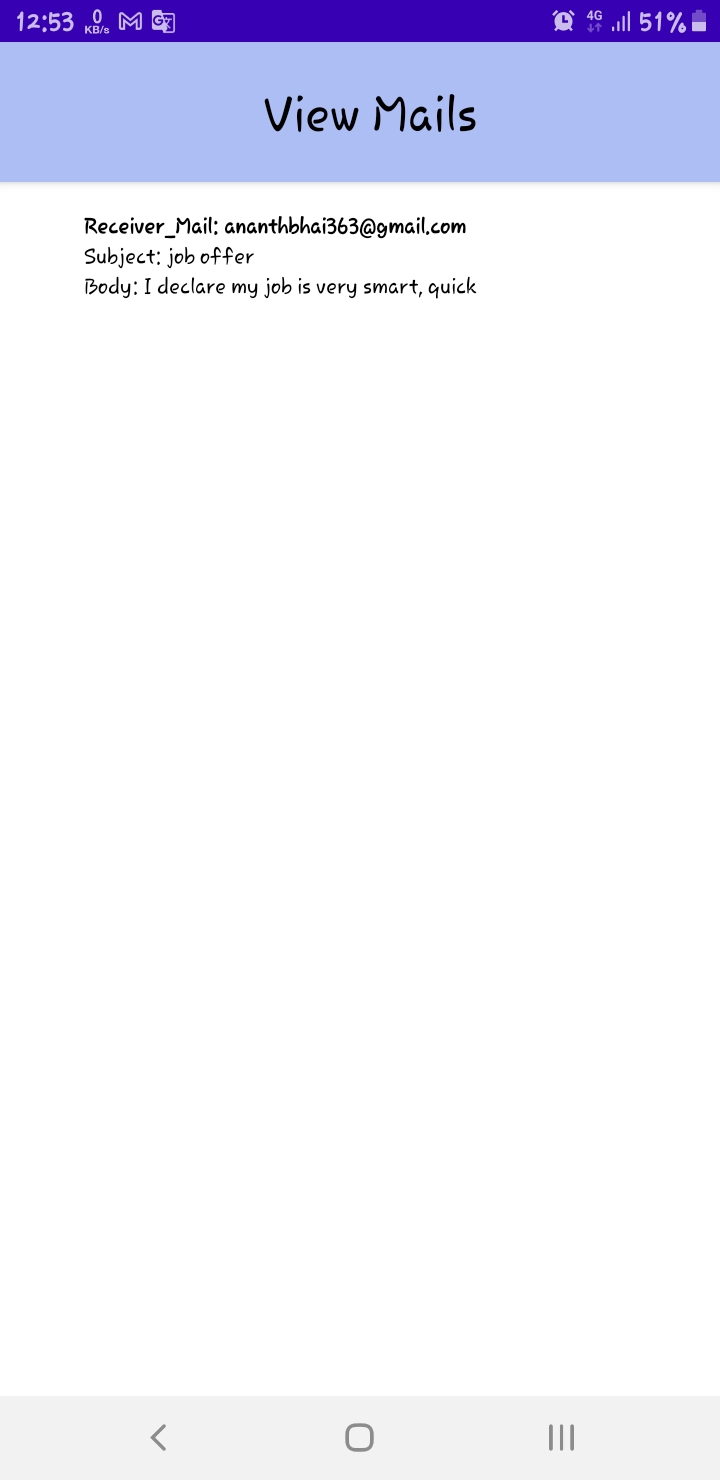


3.Result







4.ADVANTAGES & DISADVANTAGES

4.1 Advantages

* E-mails provides faster and easy mean of communication. One can send message to any person at any place of world by just clicking mouse.
* Various folders and sub-folders can be created within inbox of mail, so it provide management of messages.
* It is effective and cheap means of communication because single message can be send to multiple people at same time.
* E-mails are very easy to filter. User according to his/her priority can prioritize e-mail by specifying subject of e-mail.
* E-mail is not just only for textual message. One can send any kind of multimedia within mail.
* E-mail can be send at any hour of day, thus ensures timeliness of message.
* It is secure and reliable method to deliver our message.
* It also provide facility for edition and formatting of textual messages.
* There is also facility of auto-responders in e-mail i.e. to send automated e-mails with certain text.
* To write an e-mail there is no need of any kind of paper, thus it is environment friendly.

4.2 Disadvantages

* It is source of viruses. It is capable to harm one’s computer and read out user’s e-mail address book and send themselves to number of people around the world.
* It can be source of various spams. These spam mails can fill up inbox and to deletion of these mail consumes lot of time.
* It is informal method of communication. The documents those require signatures are not managed by e-mail.
* To use facility of e-mail, user must have an access to internet and there are many parts of world where people does not have access to Internet.
* Sometimes, e-mails becomes misunderstood as it is not capable of expressing emotions.
* To be updated, user have to check inbox from time-to-time.

5. Application

Electronic mail (e-mail) is a computer-based application for the exchange of messages between users. A worldwide e-mail network allows people to exchange e-mail messages very quickly. E-mail is the electronic equivalent of a letter, but with advantages in timeliness and flexibility.

6. Conclusion

At the end of the day, email is still one of the most effective forms of marketing. If you use it correctly, you can target prospective customers with relevant information, at precisely the right time - gaining brand loyalists for life.

7. Future Scope

### 7.1 Hyper-Personalization will be the need of Great Importance

* Personalization is just the same old thing, yet it’s ready to rule the future of Email Marketing by incorporating new means.
* Customizing subject lines is not going to help in the future. Personalized use of list segmentation will be the key. It will assist you in putting your clients into records in light of client socioeconomics, preferences, and other rich information accessible to you about your existing and potential customers.
* This can make it less demanding for you to send content to prospects that is significant to their interests. This will increase the odds that the recipient will open that mail.
* In case you are not fitting your messages to singular needs of your targets, at that point you’re not making the real effort to understand your audiences. Concerning emails, individuals tend to sift through anything that is not of their use or interest. Along these lines, it turns out to be extremely troublesome for brands to convert.
* Hyper personalization of your emails will have an immense effect.
* In future, you may see emails whose content will change relying upon the time of opening the email. Some factors that will play a key role in the personalization of emails are-

#### Interactivity of Emails

* Future marketers need search for chances to make their messages intuitive and more interactive.
* This can help in an assortment of ways. Clients can rapidly put orders, advertisers can accumulate better information, it can help abbreviate the buy pipeline, et cetera.
* These emails should empower beneficiaries to personalize their orders, add things to a shopping basket and have a better affair in the whole process.
* Use of Chatbots also ensures the increase in email interactivity.

#### Use of Chatbots

* Future of Email Marketing will see more use of [Chatbots](https://medium.com/swlh/what-is-a-chatbot-and-how-to-use-it-for-your-business-976ec2e0a99f) to gain traction. This will quicken and empower high quality, productive discussions between businesses and beneficiaries.
* If used rightly, Chatbots can give extraordinary incentive to both your business and clients.
* As an advertiser, you’ll have the capacity to utilize particular information from your clients to make a more appealing experience.
* For the users, they will apparently have an individual at the tip of their fingers to help them out in some situations.

### 7.2 Artificial Intelligence (AI) will be a Deciding Factor

* 2017 was the time when (AI) was more prominently discussed.
* What’s more, 2018 is probably the year when advertisers see it in real life.
* This technology is going to decide the future of Email Marketing. Future trends in email marketing will revolve around the use of Artificial Intelligence. The innovation can encourage measure and comprehend innumerable metrics on how clients associated with the content, generally in ways; advertisers could never comprehend without anyone else.
* Chatbots is one such example we just discussed. They are basically AI-powered tools intended to streamline complex predefined undertakings without the requirement for a human to be included straightforwardly. They’re additionally changing the way email advertising is being completed.
* Another example can be [Siri](https://www.apple.com/ios/siri/)– Presumably the most well-known bot of all.
* Facebook Messenger has also come up with its own Chatbot, enabling organizations to collaborate with their clients in a propelled situation.
* Actually, artificial intelligence, with their capacity to track singular client conduct, can send logically significant and customized emails and hence can help brands associate with their clients on a more individual level and connect with them the correct way that can eventually help support ROI.
* With an ever-increasing number of clients utilizing numerous advanced platforms and needing data in a hurry, AI-powered features enable brands to get more information and discuss on different channels consistently. This additionally gives another vital help to advertisers in helping them keep a tab on the key metrics that can aid in streamlining and keeping track on changing business sector patterns.
* By using AI, brands can eliminate the cost and time expected to recognize fragments and openings. This will enable them to create better, more pertinent substance for their audiences.

### 7.3 Utilization of Visuals will Increase

* As specified, the interactivity of email is one of the most important future trends in Email Marketing.
* This implies advertisers need to deploy new resources and systems in their messages to make their emails more interactive. Using visuals in emails is going to be one of the most important practices to make emails have the needed ability to interact.
* Even the Future of Email Marketing in 2018 will likely to observe more use of GIFs, HTML5, and the inclusion of video in email messages to keep beneficiaries locked in.
* Be that as it may, GIFs are not everything. Significant changes to email administrations like Gmail and updates on how Internet Service Providers render messages will empower email advertisers and planners to be more innovative with the emails they send to their targets.
* So, be ready to see more visual and intuitive messages from highly savvy brands in near future. If you also want to be in the same very league, you should also start utilizing more visual elements in your emails.

### 7.4 Machine-to-Machine Correspondence will Rise

* The scope of Email Marketing is experiencing a great shift.
* Do not get surprised, if, in future, you likely won’t need to stress over the opening and sending emails.
* Specialists anticipate that with the quickly propelling of different innovations and advancing needs of clients, emails will soon get involved in expanding machine-to-machine correspondence.
* Just imagine a scenario when a grocery thing in your refrigerator is getting consumed, at that point software in your refrigerator will instantly have the capacity to refresh your shopping list on the grocery application on your smartphone. It must be seeming like a thing of future now but for sure, Email Marketing communication between client and businesses can also be done using this way in future.
* Email Marketers need to be a great degree technically savvy while at the same time they need to adopt and adapt well with the new tools that can incorporate more of automation in their communications.

8.Appendix

Source code:-

Code

Email data base helper.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

}

}

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?,

)

Login activity.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)

}

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)

}

Main Activity.kt

package com.example.emailapplication

import android.content.Context

import android.content.Intent

import android.os.Bundle

import androidx.activity.ComponentActivity

import androidx.activity.compose.setContent

import androidx.compose.foundation.Image

import androidx.compose.foundation.background

import androidx.compose.foundation.layout.\*

import androidx.compose.material.\*

import androidx.compose.runtime.Composable

import androidx.compose.ui.Alignment

import androidx.compose.ui.Modifier

import androidx.compose.ui.graphics.Color

import androidx.compose.ui.layout.ContentScale

import androidx.compose.ui.res.painterResource

import androidx.compose.ui.text.font.FontWeight

import androidx.compose.ui.tooling.preview.Preview

import androidx.compose.ui.unit.dp

import androidx.compose.ui.unit.sp

import androidx.core.content.ContextCompat

import androidx.core.content.ContextCompat.startActivity

import com.example.emailapplication.ui.theme.EmailApplicationTheme

class MainActivity : ComponentActivity() {

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContent {

// A surface container using the 'background' color from the theme

Surface(

modifier = Modifier.fillMaxSize().background(Color.White),

) {

Email(this)

}

}

}

}

@Composable

fun Email(context: Context) {

Text(

text = "Home Screen",

modifier = Modifier.padding(top = 74.dp, start = 100.dp, bottom = 24.dp),

color = Color.Black,

fontWeight = FontWeight.Bold,

fontSize = 32.sp

)

Column(

horizontalAlignment = Alignment.CenterHorizontally,

verticalArrangement = Arrangement.Center

) {

Image(

painterResource(id = R.drawable.home\_screen), contentDescription = ""

)

Button(onClick = {

context.startActivity(

Intent(

context,

SendMailActivity::class.java

)

)

},

colors = ButtonDefaults.buttonColors(backgroundColor = Color(0xFFadbef4))

) {

Text(

text = "Send Email",

modifier = Modifier.padding(10.dp),

color = Color.Black,

fontSize = 15.sp

)

}

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

Button(onClick = {

context.startActivity(

Intent(

context,

ViewMailActivity::class.java

)

)

},

colors = ButtonDefaults.buttonColors(backgroundColor = Color(0xFFadbef4))

) {

Text(

text = "View Emails",

modifier = Modifier.padding(10.dp),

color = Color.Black,

fontSize = 15.sp

)

}

}

}

RegisterActivity.kt

package com.example.emailapplication

import android.content.Context

import android.content.Intent

import android.os.Bundle

import androidx.activity.ComponentActivity

import androidx.activity.compose.setContent

import androidx.compose.foundation.Image

import androidx.compose.foundation.background

import androidx.compose.foundation.layout.\*

import androidx.compose.material.\*

import androidx.compose.runtime.\*

import androidx.compose.ui.Alignment

import androidx.compose.ui.Modifier

import androidx.compose.ui.graphics.Color

import androidx.compose.ui.layout.ContentScale

import androidx.compose.ui.res.painterResource

import androidx.compose.ui.text.font.FontFamily

import androidx.compose.ui.text.font.FontWeight

import androidx.compose.ui.text.input.PasswordVisualTransformation

import androidx.compose.ui.tooling.preview.Preview

import androidx.compose.ui.unit.dp

import androidx.compose.ui.unit.sp

import androidx.core.content.ContextCompat

import com.example.emailapplication.ui.theme.EmailApplicationTheme

class RegisterActivity : ComponentActivity() {

private lateinit var databaseHelper: UserDatabaseHelper

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

databaseHelper = UserDatabaseHelper(this)

setContent {

RegistrationScreen(this, databaseHelper)

}

}

}

@Composable

fun RegistrationScreen(context: Context, databaseHelper: UserDatabaseHelper) {

var username by remember { mutableStateOf("") }

var password by remember { mutableStateOf("") }

var email by remember { mutableStateOf("") }

var error by remember { mutableStateOf("") }

Column(

modifier = Modifier.fillMaxSize().background(Color.White),

horizontalAlignment = Alignment.CenterHorizontally,

verticalArrangement = Arrangement.Center

) {

Image(

painterResource(id = R.drawable.email\_signup), contentDescription = "",

modifier = Modifier.height(300.dp)

)

Text(

fontSize = 36.sp,

fontWeight = FontWeight.ExtraBold,

fontFamily = FontFamily.Cursive,

text = "Register"

)

Spacer(modifier = Modifier.height(10.dp))

TextField(

value = username,

onValueChange = { username = it },

label = { Text("Username") },

modifier = Modifier

.padding(10.dp)

.width(280.dp)

)

TextField(

value = email,

onValueChange = { email = it },

label = { Text("Email") },

modifier = Modifier

.padding(10.dp)

.width(280.dp)

)

TextField(

value = password,

onValueChange = { password = it },

label = { Text("Password") },

visualTransformation = PasswordVisualTransformation(),

modifier = Modifier

.padding(10.dp)

.width(280.dp)

)

if (error.isNotEmpty()) {

Text(

text = error,

color = MaterialTheme.colors.error,

modifier = Modifier.padding(vertical = 16.dp)

)

}

Button(

onClick = {

if (username.isNotEmpty() && password.isNotEmpty() && email.isNotEmpty()) {

val user = User(

id = null,

firstName = username,

lastName = null,

email = email,

password = password

)

databaseHelper.insertUser(user)

error = "User registered successfully"

// Start LoginActivity using the current context

context.startActivity(

Intent(

context,

LoginActivity::class.java

)

)

} else {

error = "Please fill all fields"

}

},

colors = ButtonDefaults.buttonColors(backgroundColor = Color(0xFFd3e5ef)),

modifier = Modifier.padding(top = 16.dp)

) {

Text(text = "Register")

}

Spacer(modifier = Modifier.width(10.dp))

Spacer(modifier = Modifier.height(10.dp))

Row() {

Text(

modifier = Modifier.padding(top = 14.dp), text = "Have an account?"

)

TextButton(onClick = {

context.startActivity(

Intent(

context,

LoginActivity::class.java

)

)

})

{

Spacer(modifier = Modifier.width(10.dp))

Text(color = Color(0xFF31539a),text = "Log in")

}

}

}

}

private fun startLoginActivity(context: Context) {

val intent = Intent(context, LoginActivity::class.java)

ContextCompat.startActivity(context, intent, null)

}

SendMailActivity.kt

package com.example.emailapplication

import android.annotation.SuppressLint

import android.content.Context

import android.content.Intent

import android.os.Bundle

import androidx.activity.ComponentActivity

import androidx.activity.compose.setContent

import androidx.compose.foundation.layout.\*

import androidx.compose.material.\*

import androidx.compose.runtime.\*

import androidx.compose.ui.Alignment

import androidx.compose.ui.Modifier

import androidx.compose.ui.graphics.Color

import androidx.compose.ui.platform.LocalContext

import androidx.compose.ui.text.TextStyle

import androidx.compose.ui.text.font.FontWeight

import androidx.compose.ui.text.style.TextAlign

import androidx.compose.ui.tooling.preview.Preview

import androidx.compose.ui.unit.dp

import androidx.compose.ui.unit.sp

import com.example.emailapplication.ui.theme.EmailApplicationTheme

class SendMailActivity : ComponentActivity() {

private lateinit var databaseHelper: EmailDatabaseHelper

@SuppressLint("UnusedMaterialScaffoldPaddingParameter")

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

databaseHelper = EmailDatabaseHelper(this)

setContent {

Scaffold(

// in scaffold we are specifying top bar.

topBar = {

// inside top bar we are specifying

// background color.

TopAppBar(backgroundColor = Color(0xFFadbef4), modifier = Modifier.height(80.dp),

// along with that we are specifying

// title for our top bar.

title = {

// in the top bar we are specifying

// title as a text

Text(

// on below line we are specifying

// text to display in top app bar.

text = "Send Mail",

fontSize = 32.sp,

color = Color.Black,

// on below line we are specifying

// modifier to fill max width.

modifier = Modifier.fillMaxWidth(),

// on below line we are

// specifying text alignment.

textAlign = TextAlign.Center,

)

}

)

}

) {

// on below line we are

// calling method to display UI.

openEmailer(this,databaseHelper)

}

}

}

}

@Composable

fun openEmailer(context: Context, databaseHelper: EmailDatabaseHelper) {

// in the below line, we are

// creating variables for URL

var recevierMail by remember {mutableStateOf("") }

var subject by remember {mutableStateOf("") }

var body by remember {mutableStateOf("") }

var error by remember { mutableStateOf("") }

// on below line we are creating

// a variable for a context

val ctx = LocalContext.current

// on below line we are creating a column

Column(

// on below line we are specifying modifier

// and setting max height and max width

// for our column

modifier = Modifier

.fillMaxSize()

.padding(top = 55.dp, bottom = 25.dp, start = 25.dp, end = 25.dp),

horizontalAlignment = Alignment.Start

) {

// on the below line, we are

// creating a text field.

Text(text = "Receiver Email-Id",

fontWeight = FontWeight.Bold,

fontSize = 16.sp)

TextField(

// on below line we are specifying

// value for our text field.

value = recevierMail,

// on below line we are adding on value

// change for text field.

onValueChange = { recevierMail = it },

// on below line we are adding place holder as text

label = { Text(text = "Email address") },

placeholder = { Text(text = "abc@gmail.com") },

// on below line we are adding modifier to it

// and adding padding to it and filling max width

modifier = Modifier

.padding(16.dp)

.fillMaxWidth(),

// on below line we are adding text style

// specifying color and font size to it.

textStyle = TextStyle(color = Color.Black, fontSize = 15.sp),

// on below line we are

// adding single line to it.

singleLine = true,

)

// on below line adding a spacer.

Spacer(modifier = Modifier.height(10.dp))

Text(text = "Mail Subject",

fontWeight = FontWeight.Bold,

fontSize = 16.sp)

// on the below line, we are creating a text field.

TextField(

// on below line we are specifying

// value for our text field.

value = subject,

// on below line we are adding on value change

// for text field.

onValueChange = { subject = it },

// on below line we are adding place holder as text

placeholder = { Text(text = "Subject") },

// on below line we are adding modifier to it

// and adding padding to it and filling max width

modifier = Modifier

.padding(16.dp)

.fillMaxWidth(),

// on below line we are adding text style

// specifying color and font size to it.

textStyle = TextStyle(color = Color.Black, fontSize = 15.sp),

// on below line we are

// adding single line to it.

singleLine = true,

)

// on below line adding a spacer.

Spacer(modifier = Modifier.height(10.dp))

Text(text = "Mail Body",

fontWeight = FontWeight.Bold,

fontSize = 16.sp)

// on the below line, we are creating a text field.

TextField(

// on below line we are specifying

// value for our text field.

value = body,

// on below line we are adding on value

// change for text field.

onValueChange = { body = it },

// on below line we are adding place holder as text

placeholder = { Text(text = "Body") },

// on below line we are adding modifier to it

// and adding padding to it and filling max width

modifier = Modifier

.padding(16.dp)

.fillMaxWidth(),

// on below line we are adding text style

// specifying color and font size to it.

textStyle = TextStyle(color = Color.Black, fontSize = 15.sp),

// on below line we are

// adding single line to it.

singleLine = true,

)

// on below line adding a spacer.

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

// on below line adding a

// button to send an email

Button(onClick = {

if( recevierMail.isNotEmpty() && subject.isNotEmpty() && body.isNotEmpty()) {

val email = Email(

id = null,

recevierMail = recevierMail,

subject = subject,

body = body

)

databaseHelper.insertEmail(email)

error = "Mail Saved"

} else {

error = "Please fill all fields"

}

// on below line we are creating

// an intent to send an email

val i = Intent(Intent.ACTION\_SEND)

// on below line we are passing email address,

// email subject and email body

val emailAddress = arrayOf(recevierMail)

i.putExtra(Intent.EXTRA\_EMAIL,emailAddress)

i.putExtra(Intent.EXTRA\_SUBJECT,subject)

i.putExtra(Intent.EXTRA\_TEXT,body)

// on below line we are

// setting type of intent

i.setType("message/rfc822")

// on the below line we are starting our activity to open email application.

ctx.startActivity(Intent.createChooser(i,"Choose an Email client : "))

},

colors = ButtonDefaults.buttonColors(backgroundColor = Color(0xFFd3e5ef))

) {

// on the below line creating a text for our button.

Text(

// on below line adding a text ,

// padding, color and font size.

text = "Send Email",

modifier = Modifier.padding(10.dp),

color = Color.Black,

fontSize = 15.sp

)

}

}

}

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)

}

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?,

)

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

}

}

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}")

}

}

}

}

}

}

Build.gradile

plugins {

id 'com.android.application'

id 'org.jetbrains.kotlin.android'

}

android {

namespace 'com.example.emailapplication'

compileSdk 33

defaultConfig {

applicationId "com.example.emailapplication"

minSdk 21

targetSdk 33

versionCode 1

versionName "1.0"

testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"

vectorDrawables {

useSupportLibrary true

}

}

buildTypes {

release {

minifyEnabled false

proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'

}

}

compileOptions {

sourceCompatibility JavaVersion.VERSION\_1\_8

targetCompatibility JavaVersion.VERSION\_1\_8

}

kotlinOptions {

jvmTarget = '1.8'

}

buildFeatures {

compose true

}

composeOptions {

kotlinCompilerExtensionVersion '1.2.0'

}

packagingOptions {

resources {

excludes += '/META-INF/{AL2.0,LGPL2.1}'

}

}

}

dependencies {

implementation 'androidx.core:core-ktx:1.7.0'

implementation 'androidx.lifecycle:lifecycle-runtime-ktx:2.3.1'

implementation 'androidx.activity:activity-compose:1.3.1'

implementation "androidx.compose.ui:ui:$compose\_ui\_version"

implementation "androidx.compose.ui:ui-tooling-preview:$compose\_ui\_version"

implementation 'androidx.compose.material:material:1.2.0'

implementation 'androidx.room:room-common:2.5.0'

implementation 'androidx.room:room-ktx:2.5.0'

testImplementation 'junit:junit:4.13.2'

androidTestImplementation 'androidx.test.ext:junit:1.1.5'

androidTestImplementation 'androidx.test.espresso:espresso-core:3.5.1'

androidTestImplementation "androidx.compose.ui:ui-test-junit4:$compose\_ui\_version"

debugImplementation "androidx.compose.ui:ui-tooling:$compose\_ui\_version"

debugImplementation "androidx.compose.ui:ui-test-manifest:$compose\_ui\_version"

}