



Adaptive mail: A Flexible Email client App

1 INTRODUCTION :

1.1 Overview

Email is a service which allows us to send the message in electronic mode over the internet. It offers an efficient, inexpensive and real time mean of distributing information among people.

Each user of email is assigned a unique name for his email account. This name is known as E-mail address. Different users can send and receive messages according to the e-mail address.

E-mail is generally of the form username domainname. For example, webmaster tutorialspoint.com is an e-mail address where webmaster is username and tutorialspoint.com is domain name.

Although a significant amount of social science research is now addressing questions about the scope, the nature, and the consequences of electronic mail, it does not appear that many definitive and enduring answers will be forthcoming. Electronic mail is the most universal and the oldest form of computer-mediated communication, but the technology is still evolving in form and functionalities and it is still diffusing among global populations of users who are experimenting with how to integrate its capabilities into the course of their day to day lives. This does not mean that we should not collect data and assess the evolution of this genre but it does mean that we need to appreciate that we are examining a dynamic phenomenon. The answers we find today may not be valid tomorrow. However it is clear that electronic mail is here to stay and that it has stimulated whole-scale changes in the way we work and play with each other in an increasingly global village.

Electronic mail was one of the first services to emerge when developers began to assemble networks of computers. Since nearly every computer user learns to use e-mail, it became one of the first vehicles for group communication. On a discussion list, messages arrive

in one's electronic mailbox, either individually or in a periodic digest. A user subscribes to a discussion list by sending a request to join the list. In the early days, a human kept track of the membership and maintained the list. Today there are robot list managers that automate subscription requests. Some discussion lists are moderated, meaning that a human screens all traffic and exercises editorial control over which messages are passed along.

There is increasing overlap between e-mail discussion lists and bulletin board forums. Some systems allow users to participate in the same discussion by either means. Many e-mail discussion lists maintain archives which can be accessed via the Web.

1.2 Purpose

Despite all the methods of communication email is still popular and has become as necessary as a phone number or mailing address. Similar to calling somebody, email has become a standard mode of communication with the expectation being that everybody should have an email address. Email started out as a simple communication tool but is now used for much more than that.

In many ways, email has made my life easier. It allows me to:

- Contact people all over the world for free (or inexpensively)
- Communicate with more than one person at a time
- Document interactions
- Leave messages any time of day without bothering people

It was only when people started using it in alternative ways that things started to get messy, really messy. Instead of looking for a different model email kept evolving to meet new demands and expectations such as:

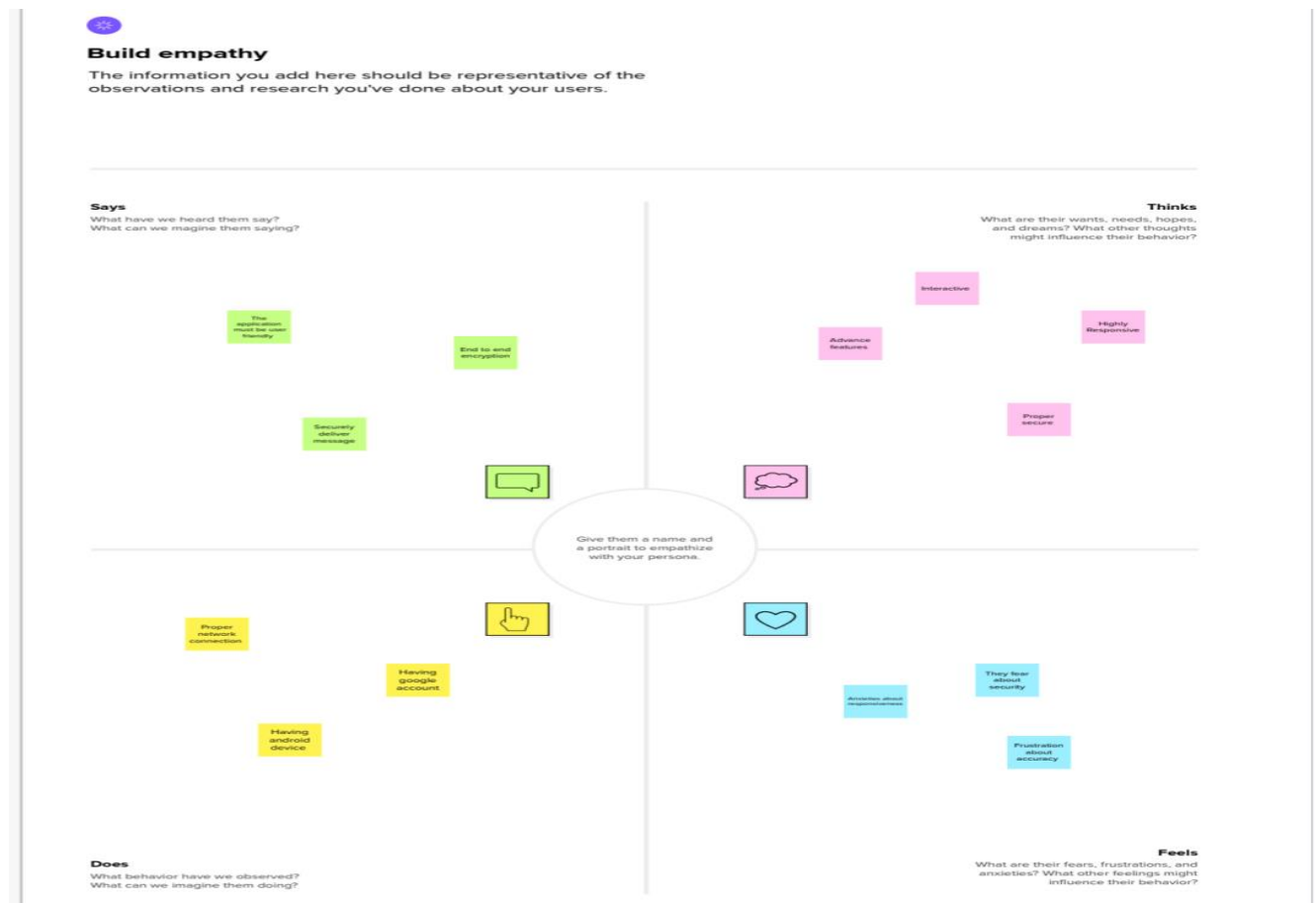
- Working collaboratively
- Sending attachments
- Keeping a conversation together for multiple people

- Searching capabilities
- Automating actions with rules
- Integrating calendars and appointments, etc.

It was almost possible to live in your email. Some of these new demands were a natural fit for this mode of communication, while others stretched the limitations and made it seem really clunky. For example sharing digital photos through email was never a good solution. The attachments are large to send and can quickly clog up an inbox making it problematic for both the sender and the recipients.


2 PROBLEM DEFINITION & DESIGN THINKING :

2.1 Empathy Map



2.2 Ideation & Brainstorming Map

template



Brainstorm & idea prioritization

Use this template in your own brainstorming sessions so your team can unleash their imagination and start shaping concepts even if you're not sitting in the same room.

10 minutes to prepare
1 hour to collaborate
2-6 people recommended

2

Before you collaborate

A little bit of preparation goes a long way with this session. Here's what you need to do to get going.

10 minutes

1

Define your problem statement

What problem are you trying to solve? Frame your problem as a How Might We statement. This will be the focus of your brainstorm.

5 minutes

3

Brainstorm

Write down any ideas that come to mind that address your problem statement.

10 minutes

Abessah M A

Belbin Joavil B S

Amin Sagar J

Belgin Paul P

Ashli J

Person 6

Person 7

Person 8

How might we [your problem statement]?

Key rules of brainstorming

To run an smooth and productive session

Stay in topic.

Encourage wild ideas.

Defer judgement.

Listen to others.

Go for volume.

If possible, be visual.

1

Group ideas

Take turns sharing your ideas while clustering similar or related notes as you go. Once all sticky notes have been grouped, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you can break it up into smaller sub-groups.

20 minutes

4

Prioritize

Your team should all be on the same page about what's important moving forward. Place your ideas on this grid to determine which ideas are important and which are feasible.

20 minutes

5

After you collaborate

You can export the mural as an image or pdf to share with members of your company who might find it helpful.

Quick add-ons

Share the mural

Export the mural

Keep moving forward

Strategy blueprint

Customer experience journey map

Strengths, weaknesses, opportunities & threats

Share template feedback

3

Group ideas

Take turns sharing your ideas while clustering similar or related notes as you go. Once all sticky notes have been grouped, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you can break it up into smaller sub-groups.

20 minutes

4

Prioritize

Your team should all be on the same page about what's important moving forward. Place your ideas on this grid to determine which ideas are important and which are feasible.

20 minutes

5

After you collaborate

You can export the mural as an image or pdf to share with members of your company who might find it helpful.

Quick add-ons

Share the mural

Export the mural

Keep moving forward

Strategy blueprint

Customer experience journey map

Strengths, weaknesses, opportunities & threats

Share template feedback

1

Group ideas

Take turns sharing your ideas while clustering similar or related notes as you go. Once all sticky notes have been grouped, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you can break it up into smaller sub-groups.

20 minutes

2

Prioritize

Your team should all be on the same page about what's important moving forward. Place your ideas on this grid to determine which ideas are important and which are feasible.

20 minutes

3

After you collaborate

You can export the mural as an image or pdf to share with members of your company who might find it helpful.

Quick add-ons

Share the mural

Export the mural

Keep moving forward

Strategy blueprint

Customer experience journey map

Strengths, weaknesses, opportunities & threats

Share template feedback

3 RESULT :

Final Output of the Application:

Login Page:



Login

Username
Abeesh

Password
.....

Login

[Sign up](#)

[Forget password?](#)



Register Page :



Register

Username

Abeesh

Email

abeeshanish@gmail.com

Password

.....

Register

Have an account? [Log in](#)

Main Page :



Home Screen



Send Email

View Emails



View Mail Page:



Receiver_Mail: abeeshanish@gmail.com

Subject: my project

Body: Email application

Receiver_Mail: vichucruzzzz@gmail.com

Subject: android

Body: hi hellovishunu

Receiver_Mail: sujithmymon@gmail.com

Subject: android

Body: he is my mon

4 ADVANTAGES & DISADVANTAGES :

Electronic Mail e-mail is one of most widely used services of [internet](#). This service allows an Internet user to send a message in formatted manner mail to other Internet user in any part of the world. Message in mail not only contain text, but it also contains images, audio and videos data. The person who is sending mail is called sender and person who receives mail is called recipient. It is just like postal mail service.

Advantages of E-mail :

1. 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.
2. Various folders and sub-folders can be created within inbox of mail, so it provide management of messages.
3. It is effective and cheap means of communication because single message can be send to multiple people at same time.
4. E-mails are very easy to filter. User according to his/her priority can prioritize e-mail by specifying subject of e-mail.
5. E-mail is not just only for textual message. One can send any kind of multimedia within mail.
6. E-mail can be send at any hour of day, thus ensures timeliness of message.
7. It is secure and reliable method to deliver our message.
8. It also provide facility for edition and formatting of textual messages.
9. There is also facility of auto-responders in e-mail i.e. to send automated e-mails with certain text.
10. To write an e-mail there is no need of any kind of paper, thus it is environment friendly.

Disadvantages of E-mail :

1. 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.
2. It can be source of various spams. These spam mails can fill up inbox and to deletion of these mail consumes lot of time.
3. It is informal method of communication. The documents those require signatures are not managed by e-mail.
4. 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.
5. Sometimes, e-mails becomes misunderstood as it is not capable of expressing emotions.
6. To be updated, user have to check inbox from time-to-time.

5 APPLICATIONS :

It is essential that users know the items they need to work on or the activities that have happened since they last logged in.

Show users a list of items they can act on or need to review. For example, for email applications, show users a list of emails; for invoicing applications, show users a list of invoices and so forth.

In this chapter we moved from discussing the web browser to email applications. Web browsers had a lot of potential vulnerabilities that could be exploited by an attacker, email clients are no different and they too have a lot of potential dangers both from the environment they are thrust into with all its dangers and the popularity of the very email technology itself. However it can be said that web browsers are in close competition with another popular application this being the ubiquitous email application. They share many of the same features and in some ways, both perform the same functions therefore can be exploited identically. Both pose real danger to the end user and

both are constantly victimized by client-side attacks. There are many ways to secure them and how to secure your mail client was covered in this chapter. We also covered some of the most common client-side attacks one could fall prey to.

In today's world email is inarguably one of the "must have" components of modern business and life which also means that the email application or client is also a common or essential component and one that the security professional must learn to deal with. Email clients, much like web browsers, are a component that is present on the majority of systems and platforms; in fact just about every operating system comes with some sort of email client already installed on the system and just waiting to be configured. Since the email client and its usage is so common it has also become a very attractive and popular target for attacks of all types. The attacker will exploit what they can on the target system and if they can't get you via the web browser, they sure will via email.

The ability to create, manipulate, and share digital documents has created a host of new applications (email, word processing, ecommerce Web sites), but it also created a new set of problems—namely how to protect the privacy and integrity of digital documents when they're stored and transmitted. The invention of public key cryptography in the 1970s¹—most important, the ability to encrypt data without a shared key and the ability to "sign" data, ensuring its origin and integrity—pointed the way to a solution to those problems. Though these operations are quite conceptually simple, they both rely on the ability to bind a public key reliably with an identity sensible to the application using the operation (for example, a globally unique name, a legal identifier, or an email address). Public Key Infrastructure (PKI) is the umbrella term used to refer to the protocols and machinery used to perform this binding.

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. Ready to get started with your own email marketing program Create engaging email marketing campaigns in just a few clicks with Weebly Marketing. Try a beautiful, professionally designed template for free today.

Still hungry for more Stay current on the latest marketing tips and tricks on the Inspiration Center.

7 FUTURE SCOPE :

Those of us who are sitting in our daily hybrid or remote work setup are likely investing several hours each day into our inbox for work and additional time for personal activities. Despite the myriad communications tools that have emerged in the last decade to progress team collaboration from Zoom and Microsoft Teams to Slack and Signal recent research proves that email remains the most used utility for nearly every one of us. But the interesting thing about email is that even though it is such a critical utility, there is very little consumer choice of email services available. In fact, only three email services that are more than 20 years old own 91% of the U.S. email market (and likely the bulk worldwide): Gmail, Yahoo Mail and Microsoft Outlook. In my previous post. I discussed how difficult it can be for new businesses to enter a market already dominated by big tech, and this is also true for new players in the email service market. Of course, these businesses are on the forefront of technology and will be around for decades to come, so why is it important for the world to have another long-term contender email service? The main reason is that these custodians of email have already demonstrated they cannot solve the

biggest problems that exist today. Spam messages accounted for 45.1% of email traffic in 2021. Anyone can jam hundreds of messages into your inbox and can do it freely. Your inbox is critical for workflow, but control of it has been taken from you. You spend hours every week deleting, unsubscribing and even creating entirely new email addresses to reduce the noise.

8 APPENDIX :

A.Source Code:

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 receiverMail: String?,
    @ColumnInfo(name = "subject") val subject: String?,
    @ColumnInfo(name = "body") val body: String?,
)
```

EmailDao:

```
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:

```

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
            }
        }
    }
}

```

EmailDataBase Helper:

```

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.receiverMail)
        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
    }
    @SuppressWarnings("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
    }
    @SuppressWarnings("Range")
    fun getAllEmails(): List<Email> {
        val emails = mutableListOf<Email>()
        val db = readableDatabase
        val cursor: Cursor = db.rawQuery("SELECT * FROM $TABLE_NAME",
null)
        if (cursor.moveToFirst()) {
            do {
                val email = Email(
                    id =
cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),

```



```

                recevierMail =
cursor.getString(cursor.getColumnIndex(COLUMN_RECEIVER_MAIL)),
                subject =
cursor.getString(cursor.getColumnIndex(COLUMN_SUBJECT)),
                body =
cursor.getString(cursor.getColumnIndex(COLUMN_BODY)),
            )
            emails.add(email)
        } while (cursor.moveToNext())
    }
    cursor.close()
    db.close()
    return emails
}
}

```

LoginActivity:

```

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,
            onChange = { username = it },
            label = { Text("Username") },
            modifier = Modifier.padding(10.dp)
                .width(280.dp)
        )

        TextField(
            value = password,
            onChange = { password = it },
            label = { Text("Password") },
            visualTransformation = PasswordVisualTransformation(),
            modifier = Modifier.padding(10.dp)
                .width(280.dp)
        )
    }
}

```

```

        if (error.isNotEmpty()) {
            Text(
                text = error,
                color = MaterialTheme.colors.error,
                modifier = Modifier.padding(vertical = 16.dp)
            )
        }

        Button(
            onClick = {
                if (username.isNotEmpty() && password.isNotEmpty()) {
                    val user =
databaseHelper.getUserByUsername(username)
                    if (user != null && user.password == password) {
                        error = "Successfully log in"
                        context.startActivity(
                            Intent(
                                context,
                                MainActivity::class.java
                            )
                        )
                        //onLoginSuccess()
                    }

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

        {
            Spacer(modifier = Modifier.width(60.dp))
            Text(color = Color(0xFF31539a),text = "Forget
password?")
        }
    }

```

```

        }
    }
}
private fun startMainPage(context: Context) {
    val intent = Intent(context, MainActivity::class.java)
    ContextCompat.startActivity(context, intent, null)
}

```

MainActivity:

```

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(0xFFadbf4))
    ) {
        Text(
            text = "View Emails",
            modifier = Modifier.padding(10.dp),
            color = Color.Black,
            fontSize = 15.sp
        )
    }
}
}

```

RegisterActivity:

```

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,
        onChange = { 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:

```

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(background-color = Color(0xFFadbfef4),
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",
                            font-size = 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.
                            text-align = 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 receiverMail 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 = receiverMail,

            // on below line we are adding on value
            // change for text field.
            onChange = { receiverMail = 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.
        onChange = { 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.
    onChange = { 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(receiverMail)
        i.putExtra(Intent.EXTRA_EMAIL, emailAddress)
        i.putExtra(Intent.EXTRA_SUBJECT, subject)
        i.putExtra(Intent.EXTRA_TEXT, body)

        // on below line we are
        // setting type of intent
        i.setType("message/rfc822")

        // on the below line we are starting our activity to open
        email application.
        ctx.startActivity(Intent.createChooser(i, "Choose an Email
        client : "))

    },
    colors = ButtonDefaults.buttonColors(backgroundColor =
    Color(0xFFd3e5ef))
) {
    // on the below line creating a text for our button.
    Text(
        // on below line adding a text ,
        // padding, color and font size.
        text = "Send Email",
        modifier = Modifier.padding(10.dp),
        color = Color.Black,
        fontSize = 15.sp
    )
}
}
}

```

User:

```

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:

```

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:

```

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:

```

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()
    }

    @SuppressWarnings("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
    }

    @SuppressWarnings("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
    }

    @SuppressWarnings("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:

[illegible]

```

        // 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}")
                    }
                }
            }
        }
    }
}

```

} }