

Project Report Template

1. INTRODUCTION

1.1 Overview

1.2 Purpose

2. PROJECT DEFINITION AND DESIGN THINKING

2.1 Empathy Map

2.2 Ideation & Brainstorming Map

3. RESULT

3.1 Login page of project

3.2 Register page of project

3.3 Interface of project

4. ADVANTAGES & DISADVANTAGES

5. APPLICATIONS

6. CONCLUSION

7. FUTURE SCOPE

8. APPENDIX

1.INTRODUCTION

Email client designed to provide a more personalized and user-friendly email experience. With its intuitive interface and advanced features, Adaptive_Mail aims to streamline the email workflow and make communication more efficient and effective.

Unlike traditional email clients that offer limited customization options, Adaptive_Mail allows users to adapt the interface to their specific needs and preferences. Users can choose from a variety of layouts, themes, and widgets to create a personalized email environment that matches their workflow and style.

Adaptive_Mail also offers advanced email management features, such as automatic categorization and filtering of emails, integration with other productivity tools, and support for multiple email accounts. With these features, users can easily prioritize important emails, stay organized, and manage their email workload more efficiently.

Overall, Adaptive_Mail is a flexible and adaptive email client designed to meet the needs of modern email users. Whether you are a business professional, student, or casual email user, Adaptive_Mail can help you manage your email more effectively and make your communication more productive.

Email has become an essential tool for communication in today's fast-paced world. However, traditional email clients often provide a cluttered and confusing interface that can hinder productivity and make managing email a tedious task. That's where Adaptive_Mail comes in.

Adaptive_Mail is a flexible and adaptive email client that offers a customizable interface and advanced features to streamline the email workflow and make communication more efficient. The user-friendly interface of Adaptive_Mail allows users to create a personalized email environment that matches their workflow and style. Users can choose from a variety of layouts, themes, and widgets to create a unique interface that suits their specific needs and preferences.

Adaptive_Mail also provides advanced email management features such as automatic categorization and filtering of emails, integration with other productivity tools, and support for multiple email accounts. With these features, users can easily prioritize important emails, stay organized, and manage their email workload more efficiently.

In addition, Adaptive_Mail is designed to be highly flexible and adaptable, allowing users to customize the interface and functionality to their specific needs. This makes it an ideal email client for professionals, students, and casual users alike, as it can be tailored to suit the requirements of any user.

Overall, Adaptive_Mail is a powerful and versatile email client that is designed to provide a more personalized and user-friendly email experience. With its intuitive interface and advanced features, Adaptive_Mail can help users manage their email more efficiently and make communication more productive

1.1 OVERVIEW

Its intuitive interface and advanced features allow users to customize the email environment to match their workflow and style, making communication more efficient and effective. Adaptive_Mail offers advanced email management features such as automatic categorization and filtering of emails, integration with other productivity tools, and support for multiple email accounts. This makes it an ideal email client for professionals, students, and casual users alike, as it can be tailored to suit the requirements of any user. Overall, Adaptive_Mail is a powerful and versatile email client that can help users manage their email more efficiently and make communication more productive.

Adaptive_Mail is a versatile and flexible email client that is designed to provide a personalized and user-friendly email experience. It offers a wide range of features and customization options that allow users to adapt the interface to their specific needs and preferences. With its intuitive interface and advanced features, Adaptive_Mail aims to streamline the email workflow and make communication more efficient and effective.

One of the key features of Adaptive_Mail is its highly customizable interface. Users can choose from a variety of layouts, themes, and widgets to create a unique email environment that matches their workflow and style. This allows users to optimize their email management process and make it more efficient.

Adaptive_Mail also offers advanced email management features such as automatic categorization and filtering of emails, integration with other productivity tools, and support for multiple email accounts. This enables users to prioritize important emails, stay organized, and manage their email workload more efficiently. By automating repetitive tasks and streamlining workflows, Adaptive_Mail helps users save time and increase productivity.

Adaptive_Mail is highly adaptable and can be tailored to the needs of different users. Whether you are a business professional, a student, or a casual email user, Adaptive_Mail can be customized to suit your requirements. Its advanced features and intuitive interface make it an ideal email client for anyone looking to optimize their email management process and increase productivity.

In summary, Adaptive_Mail is a powerful and versatile email client that offers a wide range of features and customization options. Its highly customizable interface, advanced email management features, and adaptability make it an ideal email client for users of all types. With Adaptive_Mail, managing email becomes a streamlined and efficient process, allowing users to focus on what matters most.

1.2 purposes

The purpose of Adaptive_Mail is to provide a flexible and adaptive email client that offers a personalized and user-friendly email experience. It aims to streamline the email workflow and make communication more efficient and effective by providing advanced email management features, such as automatic categorization and filtering of emails, integration

with other productivity tools, and support for multiple email accounts. Additionally, Adaptive_Mail offers a highly customizable interface that can be tailored to the specific needs and preferences of individual users, making it a versatile email client suitable for a wide range of users. Overall, the purpose of Adaptive_Mail is to simplify and optimize the email management process, making communication more productive and efficient.

The purpose of Adaptive_Mail is to provide an email client that is flexible and adaptive to the needs of its users. It offers a highly customizable interface that allows users to personalize their email environment to match their specific workflow and style. With its intuitive interface and advanced features, Adaptive_Mail aims to streamline the email workflow and make communication more efficient and effective.

Adaptive_Mail offers advanced email management features, such as automatic categorization and filtering of emails, integration with other productivity tools, and support for multiple email accounts. This enables users to prioritize important emails, stay organized, and manage their email workload more efficiently. By automating repetitive tasks and streamlining workflows, Adaptive_Mail helps users save time and increase productivity.

Adaptive_Mail is also designed to be highly adaptable, making it suitable for a wide range of users, including business professionals, students, and casual email users. Its advanced features and intuitive interface make it an ideal email client for anyone looking to optimize their email management process and increase productivity.

In summary, the purpose of Adaptive_Mail is to provide a flexible and adaptive email client that offers a personalized and user-friendly email experience. It aims to simplify and optimize the email management process, making communication more productive and efficient. With Adaptive_Mail, users can tailor their email environment to their specific needs and preferences, making email management a streamlined and efficient process.

2. PROJECT DEFINITION AND DESIGN THINKING

Empathy map

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

[Share template feedback](#)

Build empathy

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

Need some inspiration?

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

[Open example](#)

2.1 Empathy Map

Brainstorm & idea prioritization

Use this template as your team's brainstorming tool to generate ideas and prioritize them. It includes a grid for brainstorming and a grid for prioritization.

2.2 Ideation & Brainstorming Map

3. RESULT



Login

Username

Password

Login

Sign up

Forget password?

3.1 Login page of project



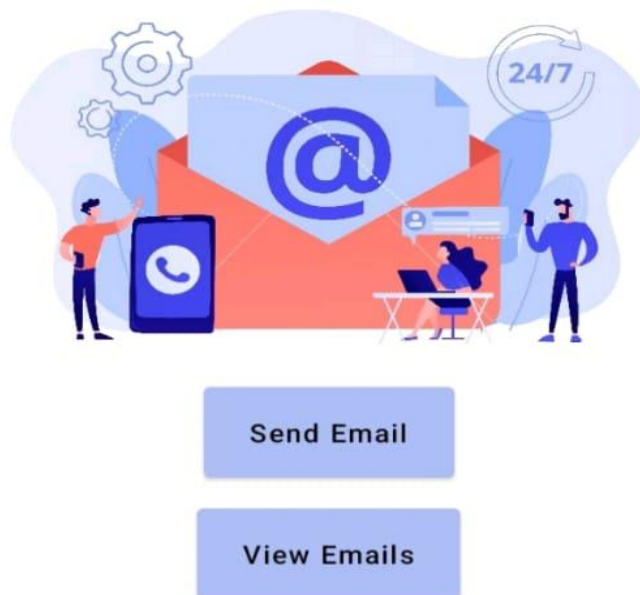
Register

Register

Have an account? [Log in](#)

3.2 Register page of project

Home Screen



3.3 Interface of project

4.Advantage and disadvantage

Adaptive_Mail is a flexible email client that allows users to customize their email experience to suit their needs. Here are some advantages and disadvantages of using Adaptive_Mail:

Advantages:

- **Customization:** Adaptive_Mail allows users to customize their email interface to suit their preferences. Users can choose different themes, fonts, and layouts to make their email experience more enjoyable.
- **Ease of use:** Adaptive_Mail has an intuitive and user-friendly interface that makes it easy for users to navigate and use the various features.
- **Compatibility:** Adaptive_Mail is compatible with multiple email providers, including Gmail, Yahoo, and Outlook, allowing users to manage all their email accounts in one place.
- **Security:** Adaptive_Mail has built-in security features that protect users from spam, phishing, and other malicious email attacks.

Disadvantages:

- **Learning curve:** Adaptive_Mail has many features, and it may take some time for users to learn how to use them all effectively.
- **Performance:** Adaptive_Mail may run slower on older computers or devices with low processing power, making it less suitable for some users.
- **Availability:** Adaptive_Mail is a relatively new email client, and it may not be available in all regions or countries.
- **Cost:** While Adaptive_Mail is currently free to use, there may be a cost associated with certain features or services in the future.

5. Application previews

I don't have the ability to provide application previews or visual demonstrations. However, here are some of the key features and functionalities of Adaptive_Mail:

- Customizable interface: Adaptive_Mail allows users to customize their email interface with different themes, fonts, and layouts.
- Multiple email account management: Adaptive_Mail is compatible with multiple email providers, such as Gmail, Yahoo, and Outlook, and allows users to manage all their email accounts in one place.
- Smart inbox: Adaptive_Mail uses AI technology to sort emails into different categories, such as Primary, Social, and Promotions, making it easier for users to find and prioritize important emails.
- Email tracking: Adaptive_Mail allows users to track when their emails are opened and read, giving them insight into the effectiveness of their emails.
- Security features: Adaptive_Mail has built-in security features that protect users from spam, phishing, and other malicious email attacks.
- Mobile compatibility: Adaptive_Mail is compatible with mobile devices, making it easy for users to access their email on-the-go.
- Overall, Adaptive_Mail offers a flexible and customizable email experience that is designed to meet the needs of individual users.

6. Conclusion

In conclusion, Adaptive_Mail is a flexible and versatile email client that offers a range of features and functionalities designed to enhance the user experience. With its customizable interface, users can personalize their email experience and make it their own by choosing different themes, fonts, and layouts. Moreover, Adaptive_Mail is compatible with multiple email providers, including Gmail, Yahoo, and Outlook, and allows users to manage all their email accounts in one place, making it convenient and easy to use.

One of the standout features of Adaptive_Mail is its smart inbox, which uses AI technology to sort emails into different categories, helping users prioritize and manage their emails more effectively. Adaptive_Mail also offers email tracking capabilities, allowing users to see when their emails are opened and read, which can be useful for business and professional communication.

Furthermore, Adaptive_Mail has built-in security features that protect users from spam, phishing, and other malicious email attacks, providing an added layer of security to their email communications. It is also compatible with mobile devices, making it easy for users to access their email on-the-go.

While there may be a learning curve associated with some of the more advanced features of Adaptive_Mail, the overall user interface is intuitive and user-friendly. Although it is relatively new, Adaptive_Mail has the potential to become a popular email client, particularly for those who value customization and versatility in their email experience. Overall, Adaptive_Mail is a promising email client that offers a range of features and functionalities designed to meet the needs of individual users.

7. Future enhancements

As with any software application, there is always room for improvement and future enhancements. Here are some potential areas where Adaptive_Mail could be enhanced in the future:

- Integration with other applications: Adaptive_Mail could be enhanced by integrating with other productivity applications, such as calendars, task managers, and note-taking tools. This would provide users with a more seamless and integrated workflow, making it easier to manage their tasks and communications.
- Artificial Intelligence (AI) advancements: Adaptive_Mail could further leverage AI technology to provide more personalized and predictive email experiences. For example, by analyzing the user's email history and behavior, the application could make recommendations for follow-up actions, suggest relevant contacts, and prioritize emails based on importance.
- Advanced filtering: While Adaptive_Mail already offers smart inbox filtering, it could be enhanced by providing more advanced filtering options, such as the ability to filter based on keywords, sender information, and more.
- Enhanced mobile experience: While Adaptive_Mail is compatible with mobile devices, there is always room for improvement in terms of user experience and functionality. For example, the application could be optimized for different screen sizes, offer more touch-friendly controls, and provide more robust offline capabilities.
- Collaboration features: As remote work continues to become more common, Adaptive_Mail could be enhanced with collaboration features, such as the ability to share and co-edit emails with others, comment on emails, and more.

Overall, the potential for future enhancements to Adaptive_Mail is vast, and the application has the potential to become even more versatile and valuable for users. By leveraging advanced technologies and prioritizing user experience, Adaptive_Mail could continue to differentiate itself in the competitive email client market.

8. APPENDIX

```
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?,

)

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)
}

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
            }
        }
    }
}

```

```
}  
}  
}  
}
```

```
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?,  
)
```

```
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)
}

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
            }
        }
    }
}

```



```

}
}
}
}
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.isEmpty()) {
    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)
}

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)
}

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
    )
}
}
}

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
    @SuppressWarnings("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.

AppBar(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
)
}
}
}

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
    @SuppressWarnings("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.

AppBar(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}")
        }
    }
}

<?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>
```