Adaptive Flexible Email Client

Presented By

Team ID: NM2023TMID34998

Team Size: 4

Team Leader: Pradeep Raja V

Team member: NABISHA B

Team member: RATHISH R

Team member: VASANTH A

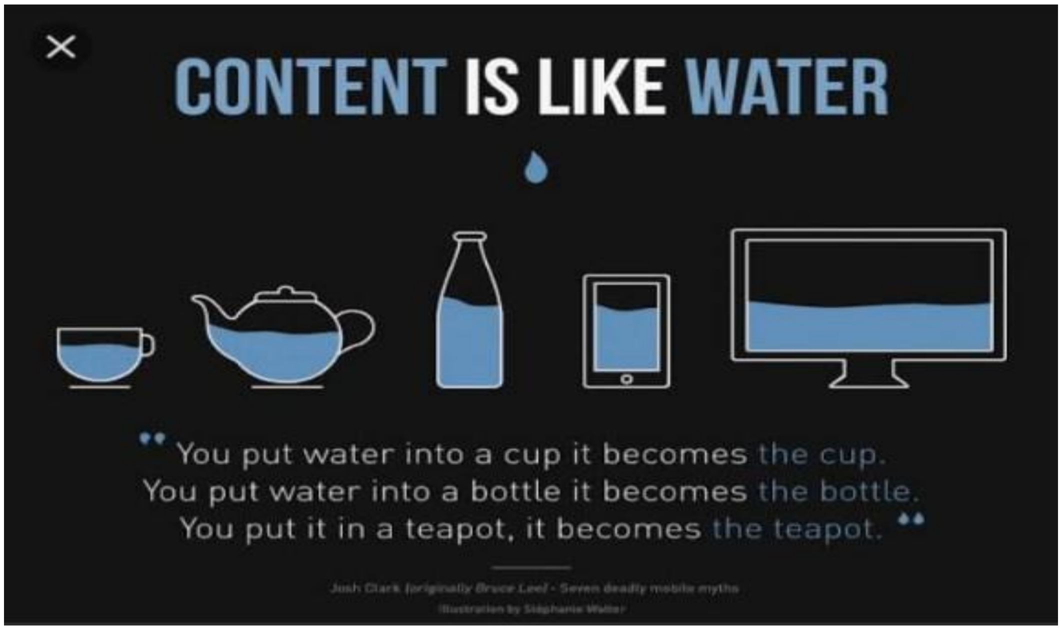
INTRODUCTION:

A flexible client app is a versatile software application designed to adapt to the changing needs of its users. Such an app is designed to be customizable and scalable, allowing users to tailor it to their specific requirements and preferences.

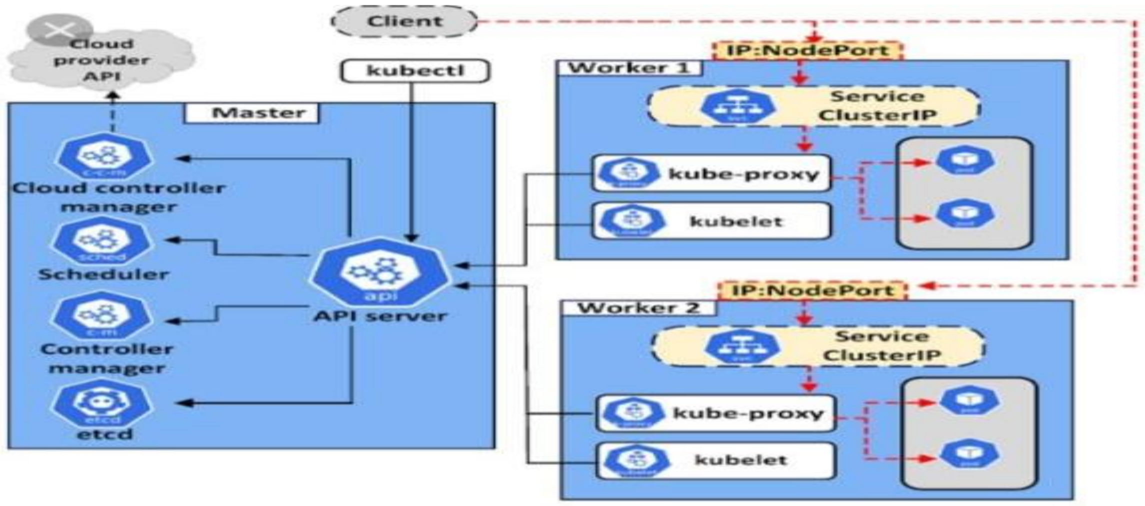
Whether it is a mobile app or a desktop application, a flexible client app is designed to provide a seamless user experience by offering a wide range of features, functionalities, and integration options. Its flexible architecture enables it to be easily modified or extended to suit the evolving needs of its users, making it an ideal choice for businesses and individuals alike.

In this era of rapidly changing technology and customer demands, a flexible client app can provide a competitive advantage by allowing organizations to stay ahead of the curve and meet the ever-changing needs of their customers.

ADAPTIVE AND RESPONSIVE DESIGN:


CONTENT IS LIKE WATER
"You put water into a cup it becomes the cup.
You put water into a bottle it becomes the bottle.
You put it in a teapot, it becomes the teapot."
Josh Clark (originally Bruce Lee) - Seven deadly mobile myths
Illustration by Stéphanie Walter

SENSORS:



Advantages:
Multi-Account Management: A flexible e-mail client app allows users to manage multiple e-mail accounts in a single interface, making it easy to stay on top of all their messages without having to switch between different applications.

Customization Options: With a flexible e-mail client app, users can customize the look and feel of their inbox, as well as the settings and preferences to suit their needs. This makes it easier to organize their e-mails and increase their productivity.

Integration with other tools: A flexible e-mail client app can integrate with other tools and services, such as calendars, task managers, and cloud storage platforms. This integration provides a seamless experience, allowing users to access all their essential tools from a single application.

Advanced Security Features: A flexible e-mail client app can provide advanced security features such as spam filters, phishing protection, and encryption options. This ensures that users’ e-mails are protected from potential threats.

Intelligent Automation: A flexible e-mail client app can offer intelligent automation features, such as auto-responses, smart sorting, and reminders. This saves users time and helps them manage their e-mails more efficiently.

Disadvantages:
Complexity: A flexible email client app can be more complex than a traditional email client. This can make it harder to use, especially for people who are not tech-savvy.

Customization: While customization is one of the key advantages of a flexible email client app, it can also be a disadvantage. Too much customization can be overwhelming and confusing for some users.

Compatibility: A flexible email client app may not be compatible with all email providers or services. This can limit its usefulness for some users.

Security: A flexible email client app may not have the same level of security as a traditional email client. This can be a concern for users who are worried about the security of their emails.

Technical Support: A flexible email client app may not have the same level of technical support as a traditional email client. This can be a problem if you run into technical issues or need help troubleshooting problems.

Future Scope:

Customizable user interface: Users can customize the layout and design of the email client app according to their preferences.

Multiple account support: The email client app can support multiple email accounts, allowing users to manage all their emails in one place.

Smart inbox: The email client app can have a smart inbox that automatically categorizes emails into different folders based on their content.

Advanced search: The email client app can offer advanced search features that allow users to find specific emails quickly.

Email scheduling: The email client app can allow users to schedule emails to be sent at a later time.

Reminders: The email client app can have a reminder feature that alerts users when they receive an important email.

Integration with other apps: The email client app can integrate with other apps like calendars, task managers, and notes, making it easier for users to manage their tasks.

Encryption and security: The email client app can provide encryption and security features to ensure the safety of the user’s data.

Automatic replies: The email client app can offer automatic reply features that allow users to set up pre-written responses to frequently asked questions.

Analytics and insights: The email client app can provide analytics and insights into email usage, helping users to manage their inbox more efficiently.

Overall, a flexible email client app that offers these features can significantly improve the productivity and efficiency of users

Appendix:

An appendix for a flexible email client app could include additional features or information that can be useful for users. Here are some examples:

Keyboard shortcuts: A list of keyboard shortcuts that can be used to quickly navigate the email client app.

Supported email protocols: A list of email protocols that the app supports, such as POP, IMAP, and Exchange.

Privacy policy: A detailed description of how the app handles user data and protects user privacy.

Frequently Asked Questions (FAQs): A list of frequently asked questions and their answers to help users troubleshoot common issues.

Release notes: A list of the latest updates and features added to the app, along with bug fixes and improvements.

User guide: A comprehensive guide on how to use the app, including step-by-step instructions and screenshots.

Customer support: Contact information for customer support, such as email, phone, or chat support.

User feedback: A section for users to provide feedback on the app, such as suggestions for improvements or bug reports.

XML CODES:

```
<?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>
```

UI THEME
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?,
)
```

Color.kt

```
Package com.example.emailapplication.ui.theme

Import androidx.compose.ui.graphics.Color

Val Purple200 = Color(0xFFBB86FC)
Val Purple500 = Color(0xFF6200EE)
Val Purple700 = Color(0xFF3700B3)
Val Teal200 = Color(0xFF03DAC5)
```

Shape.kt

```
Package com.example.emailapplication.ui.theme

Import androidx.compose.foundation.shape.RoundedCornerShape
Import androidx.compose.material.Shapes
Import androidx.compose.ui.unit.dp

Val Shapes = Shapes(
   Small = RoundedCornerShape(4.dp),
   Medium = RoundedCornerShape(4.dp),
   Large = RoundedCornerShape(0.dp)
)
```

Theme.kt

```
Package com.example.emailapplication.ui.theme

Import androidx.compose.foundation.isSystemInDarkTheme
Import androidx.compose.material.MaterialTheme
Import androidx.compose.material.darkColors
Import androidx.compose.material.lightColors
Import androidx.compose.runtime.Composable

Private val DarkColorPalette = darkColors(
   Primary = Purple200,
   primaryVariant = Purple700,
   secondary = Teal200
)

Private val LightColorPalette = lightColors(
   Primary = Purple500,
   primaryVariant = Purple700,
   secondary = Teal200

   /* Other default colors to override
   Background = Color.White,
   Surface = Color.White,
   onPrimary = Color.White,
   onSecondary = Color.Black,
   onBackground = Color.Black,
   onSurface = Color.Black,
   */
)

@Composable
Fun EmailApplicationTheme(
   darkTheme: Boolean = isSystemInDarkTheme(),
   content: @Composable () -> Unit
) {
   Val colors = if (darkTheme) {
      DarkColorPalette
   } else {
      LightColorPalette
   }

   MaterialTheme(
      Colors = colors,
      Typography = Typography,
      Shapes = Shapes,
      Content = content
   )
}
Type.kt
Package com.example.emailapplication.ui.theme

Import androidx.compose.material.Typography
Import androidx.compose.ui.text.TextStyle
Import androidx.compose.ui.text.font.FontFamily
Import androidx.compose.ui.text.font.FontWeight
Import androidx.compose.ui.unit.sp

// Set of Material typography styles to start with
Val Typography = Typography(
    Body1 = TextStyle(
        fontFamily = FontFamily.Default,
        fontWeight = FontWeight.Normal,
        fontSize = 16.sp
    )
    /* Other default text styles to override
    Button = TextStyle(
        fontFamily = FontFamily.Default,
        fontWeight = FontWeight.W500,
        fontSize = 14.sp
    ),
    Caption = TextStyle(
        fontFamily = FontFamily.Default,
        fontWeight = FontWeight.Normal,
        fontSize = 12.sp
    )
    */
)
```

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?,
)
```

EmailDao.kt

```
Package com.example.emailapplication
```

```
Import androidx.room.*

@Dao
Interface EmailDao {

   @Query("SELECT * FROM email_table WHERE  subject= :subject")
   Suspend fun getOrderBySubject(subject: String): Email?

   @Insert(onConflict = OnConflictStrategy.REPLACE)
   Suspend fun insertEmail(email: Email)

   @Update
   Suspend fun updateEmail(email: Email)

   @Delete
   Suspend fun deleteEmail(email: Email)
}
```

EmailDatabase.kt

```
Package com.example.emailapplication

Import android.content.Context
Import androidx.room.Database
Import androidx.room.Room
Import androidx.room.RoomDatabase

@Database(entities = [Email::class], version = 1)
Abstract class EmailDatabase : RoomDatabase() {

   Abstract fun emailDao(): EmailDao

   Companion object {

      @Volatile
      Private var instance: EmailDatabase? = null

      Fun getDatabase(context: Context): EmailDatabase {
            Return instance ?: synchronized(this) {
                Val newInstance = Room.databaseBuilder(
                    Context.applicationContext,
                    EmailDatabase::class.java,
                    “email_database”
                ).build()
                Instance = newInstance
                newInstance
            }
        }
    }
}

EmailDatabaseHelper.kt
Package com.example.emailapplication

Import android.annotation.SuppressLint
Import android.content.ContentValues
Import android.content.Context
Import android.database.Cursor
Import android.database.sqlite.SQLiteDatabase
Import android.database.sqlite.SQLiteOpenHelper

Class 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
    }

}
```

LoginActivity.kt

```
Package com.example.emailapplication

Import android.content.Context
Import android.content.Intent
Import android.os.Bundle
Import androidx.activity.ComponentActivity
Import androidx.activity.compose.setContent
Import androidx.compose.foundation.Image
Import androidx.compose.foundation.background
Import androidx.compose.foundation.layout.*
Import androidx.compose.material.*
Import androidx.compose.runtime.*
Import androidx.compose.ui.Alignment
Import androidx.compose.ui.Modifier
Import androidx.compose.ui.graphics.Color
Import androidx.compose.ui.layout.ContentScale
Import androidx.compose.ui.res.painterResource
Import androidx.compose.ui.text.font.FontFamily
Import androidx.compose.ui.text.font.FontWeight
Import androidx.compose.ui.text.input.PasswordVisualTransformation
Import androidx.compose.ui.tooling.preview.Preview
Import androidx.compose.ui.unit.dp
Import androidx.compose.ui.unit.sp
Import androidx.core.content.ContextCompat
Import com.example.emailapplication.ui.theme.EmailApplicationTheme

Class LoginActivity : ComponentActivity() {
   Private lateinit var databaseHelper: UserDatabaseHelper
   Override fun onCreate(savedInstanceState: Bundle?) {
      Super.onCreate(savedInstanceState)
      databaseHelper = UserDatabaseHelper(this)
      setContent {

         LoginScreen(this, databaseHelper)
      }
   }
}
@Composable
Fun LoginScreen(context: Context, databaseHelper: UserDatabaseHelper) {


   Var username by remember { mutableStateOf("") }
   Var password by remember { mutableStateOf("") }
   Var error by remember { mutableStateOf("") }

   Column(
      Modifier = Modifier.fillMaxSize().background(Color.White),
      horizontalAlignment = Alignment.CenterHorizontally,
      verticalArrangement = Arrangement.Center
   ) {
      Image(
         painterResource(id = R.drawable.email_login), contentDescription = ""
      )
```

```
Text(
   fontSize = 36.sp,
   fontWeight = FontWeight.ExtraBold,
   fontFamily = FontFamily.Cursive,
   text = “Login”
)
Spacer(modifier = Modifier.height(10.dp))

TextField(
   Value = username,
   onValueChange = { username = it },
   label = { Text(“Username”) },
   modifier = Modifier.padding(10.dp)
      .width(280.dp)
)

TextField(
   Value = password,
   onValueChange = { password = it },
   label = { Text(“Password”) },
   visualTransformation = PasswordVisualTransformation(),
   modifier = Modifier.padding(10.dp)
      .width(280.dp)
)

If (error.isNotEmpty()) {
   Text(
      Text = error,
      Color = MaterialTheme.colors.error,
      Modifier = Modifier.padding(vertical = 16.dp)
   )
}

Button(
   onClick = {
      if (username.isNotEmpty() && password.isNotEmpty()) {
```

```
                val user = databaseHelper.getUserByUsername(username)
                if (user != null && user.password == password) {
                    error = “Successfully log in”
                    context.startActivity(
                        Intent(
                            Context,
                            MainActivity::class.java
                        )
                    )
                    //onLoginSuccess()
                }

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

        {
            Spacer(modifier = Modifier.width(60.dp))
            Text(color = Color(0xFF31539a),text = “Forget password?”)
            }
        }
    }
}
Private fun startMainPage(context: Context) {
    Val intent = Intent(context, MainActivity::class.java)
    ContextCompat.startActivity(context, intent, null)
}
MainActivity.kt
Package com.example.emailapplication

Import android.content.Context
Import android.content.Intent
Import android.os.Bundle
Import androidx.activity.ComponentActivity
Import androidx.activity.compose.setContent
Import androidx.compose.foundation.Image
Import androidx.compose.foundation.background
Import androidx.compose.foundation.layout.*
Import androidx.compose.material.*
Import androidx.compose.runtime.Composable
Import androidx.compose.ui.Alignment
Import androidx.compose.ui.Modifier
Import androidx.compose.ui.graphics.Color
Import androidx.compose.ui.layout.ContentScale
Import androidx.compose.ui.res.painterResource
Import androidx.compose.ui.text.font.FontWeight
Import androidx.compose.ui.tooling.preview.Preview
Import androidx.compose.ui.unit.dp
Import androidx.compose.ui.unit.sp
Import androidx.core.content.ContextCompat
Import androidx.core.content.ContextCompat.startActivity
Import com.example.emailapplication.ui.theme.EmailApplicationTheme

Class MainActivity : ComponentActivity() {
    Override fun onCreate(savedInstanceState: Bundle?) {
        Super.onCreate(savedInstanceState)
        setContent {
            // A surface container using the ‘background’ color from the theme
            Surface(
              Modifier = Modifier.fillMaxSize().background(Color.White),
            ) {
              Email(this)
            }

        }
    }
}

@Composable
Fun Email(context: Context) {
    Text(
        Text = “Home Screen”,
        Modifier = Modifier.padding(top = 74.dp, start = 100.dp, bottom = 24.dp),
        Color = Color.Black,
        fontWeight = FontWeight.Bold,
        fontSize = 32.sp
    )

    Column(
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {
        Image(
            painterResource(id = R.drawable.home_screen), contentDescription = “”
        )


        Button(onClick = {
            Context.startActivity(
                Intent(
                    Context,
                    SendMailActivity::class.java
                )
```

```
          )
       },
          Colors = ButtonDefaults.buttonColors(backgroundColor =
Color(0xFFadbef4))
       ) {
          Text(
             Text = “Send Email”,
             Modifier = Modifier.padding(10.dp),
             Color = Color.Black,
             fontSize = 15.sp
          )
       }

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

       Button(onClick = {
          Context.startActivity(
             Intent(
                Context,
                ViewMailActivity::class.java
             )
          )
       },
          Colors = ButtonDefaults.buttonColors(backgroundColor =
Color(0xFFadbef4))
       ) {
          Text(
             Text = “View Emails”,
             Modifier = Modifier.padding(10.dp),
             Color = Color.Black,
             fontSize = 15.sp
          )
       }


    }
}
```

```
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
           )
        }
     }
}
```

User.kt

```
Package com.example.emailapplication

Import androidx.room.ColumnInfo
Import androidx.room.Entity
Import androidx.room.PrimaryKey

@Entity(tableName = “user_table”)
Data class User(
   @PrimaryKey(autoGenerate = true) val id: Int?,
   @ColumnInfo(name = “first_name”) val firstName: String?,
   @ColumnInfo(name = “last_name”) val lastName: String?,
   @ColumnInfo(name = “email”) val email: String?,
   @ColumnInfo(name = “password”) val password: String?,

   )
```

UserDao.kt

```
Package com.example.emailapplication

Import androidx.room.*

@Dao
Interface UserDao {

   @Query(“SELECT * FROM user_table WHERE email = :email”)
```

```
    Suspend fun getUserByEmail(email: String): User?

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    Suspend fun insertUser(user: User)

    @Update
    Suspend fun updateUser(user: User)

    @Delete
    Suspend fun deleteUser(user: User)
}

UserDatabase.kt
Package com.example.emailapplication

Import android.content.Context
Import androidx.room.Database
Import androidx.room.Room
Import androidx.room.RoomDatabase

@Database(entities = [User::class], version = 1)
Abstract class UserDatabase : RoomDatabase() {

    Abstract fun userDao(): UserDao

    Companion object {

        @Volatile
        Private var instance: UserDatabase? = null

        Fun getDatabase(context: Context): UserDatabase {
            Return instance ?: synchronized(this) {
                Val newInstance = Room.databaseBuilder(
                    Context.applicationContext,
                    UserDatabase::class.java,
                    "user_database"
                ).build()
            Instance = newInstance
            newInstance
         }
      }
   }
}
UserDatabaseHelper.kt
Package com.example.emailapplication

Import android.annotation.SuppressLint
Import android.content.ContentValues
Import android.content.Context
Import android.database.Cursor
Import android.database.sqlite.SQLiteDatabase
Import android.database.sqlite.SQLiteOpenHelper

Class UserDatabaseHelper(context: Context) :
   SQLiteOpenHelper(context, DATABASE_NAME, null, DATABASE_VERSION) {

   Companion object {
      Private const val DATABASE_VERSION = 1
      Private const val DATABASE_NAME = "UserDatabase.db"

      Private const val TABLE_NAME = "user_table"
      Private const val COLUMN_ID = "id"
      Private const val COLUMN_FIRST_NAME = "first_name"
      Private const val COLUMN_LAST_NAME = "last_name"
      Private const val COLUMN_EMAIL = "email"
      Private const val COLUMN_PASSWORD = "password"
   }

   Override fun onCreate(db: SQLiteDatabase?) {
      Val createTable = "CREATE TABLE $TABLE_NAME (" +
            "$COLUMN_ID INTEGER PRIMARY KEY AUTOINCREMENT, " +
            "$COLUMN_FIRST_NAME TEXT, " +
            "$COLUMN_LAST_NAME TEXT, " +
            "$COLUMN_EMAIL TEXT, " +
            "$COLUMN_PASSWORD TEXT" +
            ")"

        Db?.execSQL(createTable)
    }

    Override fun onUpgrade(db: SQLiteDatabase?, oldVersion: Int, newVersion: Int)
{
        Db?.execSQL("DROP TABLE IF EXISTS $TABLE_NAME")
        onCreate(db)
    }

    Fun insertUser(user: User) {
        Val db = writableDatabase
        Val values = ContentValues()
        Values.put(COLUMN_FIRST_NAME, user.firstName)
        Values.put(COLUMN_LAST_NAME, user.lastName)
        Values.put(COLUMN_EMAIL, user.email)
        Values.put(COLUMN_PASSWORD, user.password)
        Db.insert(TABLE_NAME, null, values)
        Db.close()
    }

    @SuppressLint("Range")
    Fun getUserByUsername(username: String): User? {
        Val db = readableDatabase
        Val cursor: Cursor = db.rawQuery("SELECT * FROM $TABLE_NAME WHERE
$COLUMN_FIRST_NAME = ?", arrayOf(username))
        Var user: User? = null
        If (cursor.moveToFirst()) {
            User = User(
                Id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),
                firstName =
cursor.getString(cursor.getColumnIndex(COLUMN_FIRST_NAME)),
                lastName =
cursor.getString(cursor.getColumnIndex(COLUMN_LAST_NAME)),
                email = cursor.getString(cursor.getColumnIndex(COLUMN_EMAIL)),
            password =
cursor.getString(cursor.getColumnIndex(COLUMN_PASSWORD)),
         )
      }
      Cursor.close()
      Db.close()
      Return user
   }
   @SuppressLint("Range")
   Fun getUserById(id: Int): User? {
      Val db = readableDatabase
      Val cursor: Cursor = db.rawQuery("SELECT * FROM $TABLE_NAME WHERE
$COLUMN_ID = ?", arrayOf(id.toString()))
      Var user: User? = null
      If (cursor.moveToFirst()) {
         User = User(
            Id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),
            firstName =
cursor.getString(cursor.getColumnIndex(COLUMN_FIRST_NAME)),
            lastName =
cursor.getString(cursor.getColumnIndex(COLUMN_LAST_NAME)),
            email = cursor.getString(cursor.getColumnIndex(COLUMN_EMAIL)),
            password =
cursor.getString(cursor.getColumnIndex(COLUMN_PASSWORD)),
         )
      }
      Cursor.close()
      Db.close()
      Return user
   }

   @SuppressLint("Range")
   Fun getAllUsers(): List<User> {
      Val users = mutableListOf<User>()
      Val db = readableDatabase
      Val cursor: Cursor = db.rawQuery("SELECT * FROM $TABLE_NAME", null)
      If (cursor.moveToFirst()) {
         Do {
            Val user = User(
               Id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),
               firstName =
cursor.getString(cursor.getColumnIndex(COLUMN_FIRST_NAME)),
               lastName =
cursor.getString(cursor.getColumnIndex(COLUMN_LAST_NAME)),
               email = cursor.getString(cursor.getColumnIndex(COLUMN_EMAIL)),
               password =
cursor.getString(cursor.getColumnIndex(COLUMN_PASSWORD)),
            )
            Users.add(user)
         } while (cursor.moveToNext())
      }
      Cursor.close()
      Db.close()
      Return users
   }

}
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}”)
                }
             }
          }
       }

   }
}
ExampleInstrumentedTest.kt
Package com.example.emailapplication

Import androidx.test.platform.app.InstrumentationRegistry
Import androidx.test.ext.junit.runners.AndroidJUnit4

Import org.junit.Test
Import org.junit.runner.RunWith

Import org.junit.Assert.*

/**
 * Instrumented test, which will execute on an Android device.
 *
 * See [testing documentation](http://d.android.com/tools/testing).
 */
@RunWith(AndroidJUnit4::class)
Class ExampleInstrumentedTest {
   @Test
   Fun useAppContext() {
      // Context of the app under test.
      Val appContext = 
InstrumentationRegistry.getInstrumentation().targetContext
      assertEquals(“com.example.emailapplication”, appContext.packageName)
   }
```

```
}
ExampleUnitTest.kt
Package com.example.emailapplication

Import org.junit.Test

Import org.junit.Assert.*

/**
 * Example local unit test, which will execute on the development machine (host).
 *
 * See [testing documentation](http://d.android.com/tools/testing).
 */
Class ExampleUnitTest {
   @Test
   Fun addition_isCorrect() {
      assertEquals(4, 2 + 2)
   }
}
```