Adaptive Mail: A Flexible Email Client App

1. Abstract:

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

Key functionalities include:

- ✓ **Messaging Interface:**
  - The app simulates a typical messaging interface, allowing users to send and receive messages in real-time.
  - Users can view a history of previous messages, providing a seamless and continuous conversation experience**.**
- ✓ **Compose UI Toolkit:**
  - Utilizes Jetpack Compose, Android's modern toolkit for building native UI.
  - Demonstrates how to create flexible and responsive UIs with less code compared to traditional XML layouts.
- ✓ **Data Management:**
  - Showcases efficient data handling techniques, including managing message states and user inputs.
  - Integrates with backend services to fetch and store messages, ensuring data persistence and synchronization.
- ✓ **User Interactions:**
  - Implements intuitive and engaging user interactions.
  - Supports various input methods, including text and possibly multimedia messages, enhancing the conversational experience.

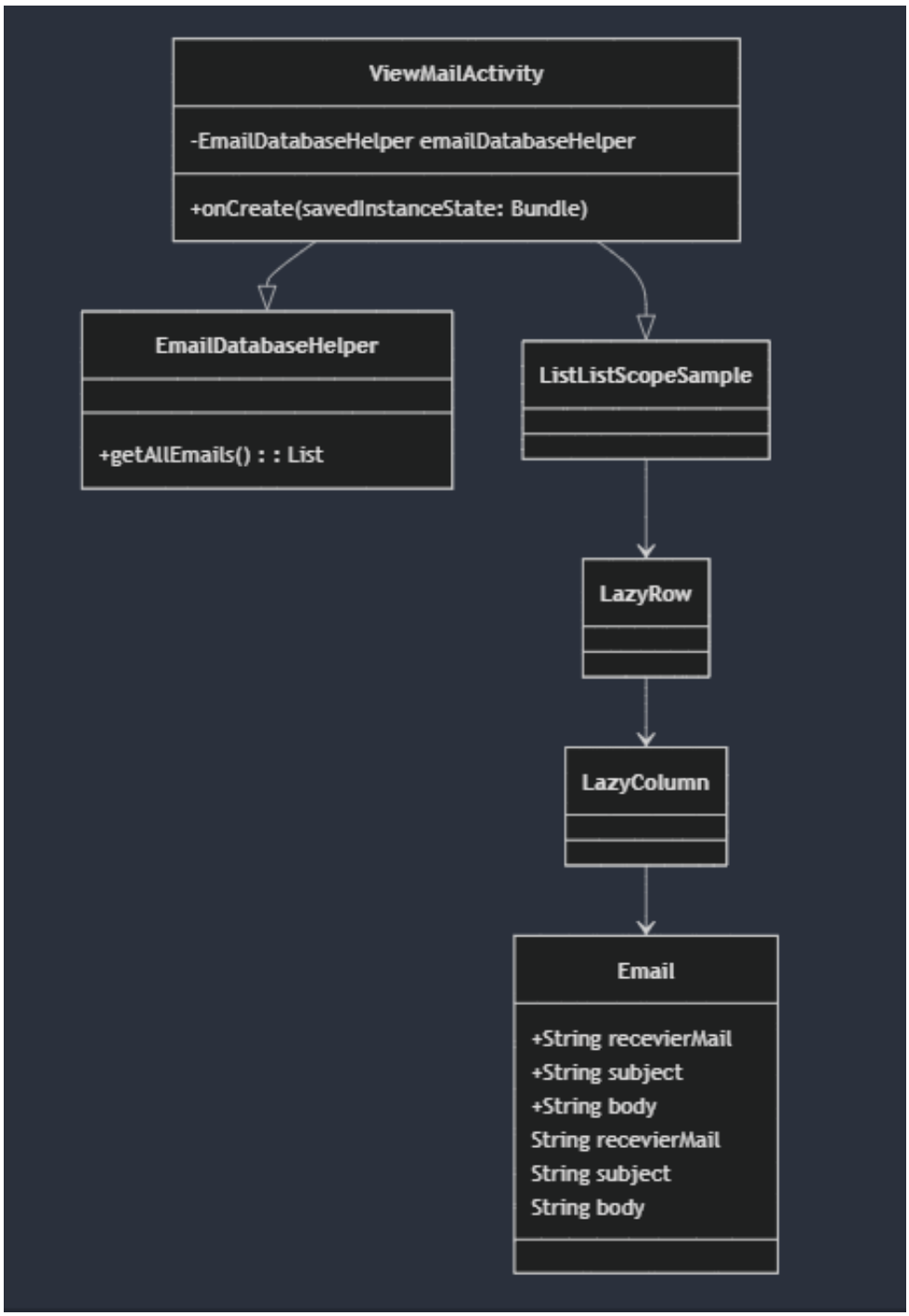
Benefits of Using Adaptive Mail App

- **Learning Tool**: Serves as an educational resource for developers new to Jetpack Compose and modern Android development practices.
- **Best Practices**: Demonstrates best practices in building scalable and maintainable UIs.
- **Reference Implementation**: Provides a reference implementation that developers can extend and customize for their own messaging applications.

2. Introduction:

The Adaptive Mail app is a sample project designed to demonstrate the power and versatility of the Android Compose UI toolkit. This project offers a practical example for developers who are keen to explore and harness the capabilities of Jetpack Compose, Android's modern toolkit for building native UI.

Data flow diagram:



3. Software Requirement:

Development Tools

1. **Android Studio**: The official IDE for Android development. Ensure you have the latest version installed.
2. **Kotlin**: The programming language used for the development of the app. Android Studio comes with built-in support for Kotlin.

SDKs and Libraries

1. **Android SDK**: Includes essential tools and APIs for developing Android apps.
2. **Jetpack Compose**: The toolkit for building native UIs in a declarative way. Make sure to add the Compose dependencies in your build.gradle file.

Version Control

1. **Git:** For version control and collaboration. You can use services like GitHub, GitLab, or Bitbucket to host your repository.

4. Sample Program Code :

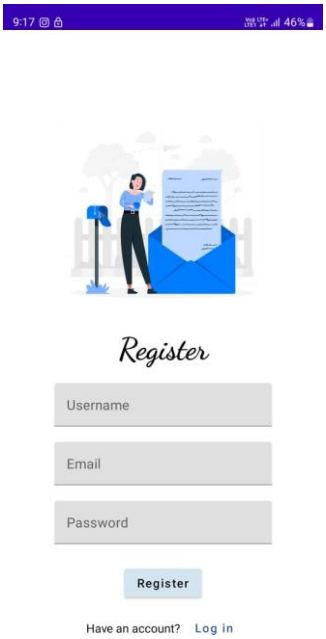
```
package com.example.emailapplication

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
```

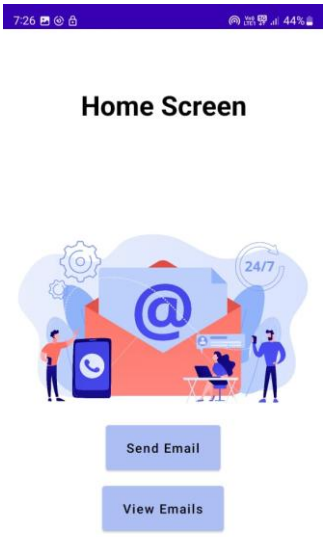
```
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import com.example.emailapplication.ui.theme.EmailApplicationTheme

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>) {
```

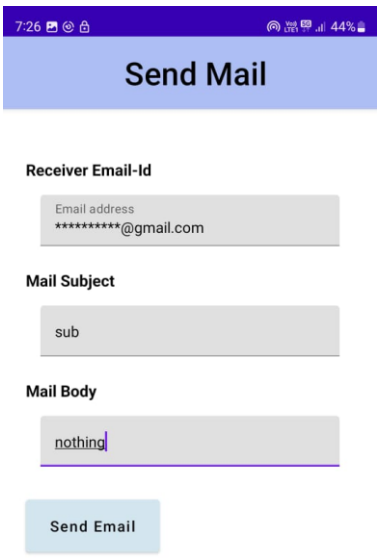
```
    LazyRow(
        modifier = Modifier
            .fillMaxSize(),
        horizontalArrangement = Arrangement.SpaceBetween
    ) {
        item {

            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}")
                    }
                }
            }
        }

    }
}
```

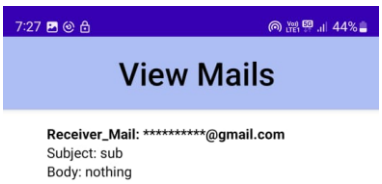
5. Output


7:26
44%
Login
Username
Ighlm
Password
Successfully log in
Login
Sign up
Forget password?


9:17
46%
Register
Username
Email
Password
Register
Have an account?
Log in


7:26
44%
Home Screen
24/7
Send Email
View Emails


7:26
44%
Send Mail
Receiver Email-Id
Email address
**********@gmail.com
Mail Subject
sub
Mail Body
nothing
Send Email


7:27
44%
View Mails
Receiver_Mail: **********@gmail.com
Subject: sub
Body: nothing

6. Conclusion:

The Adaptive Mail app is an excellent showcase of how powerful and efficient the Android Compose UI toolkit can be for developing modern, conversational UIs. Whether you're looking to understand the basics of Compose or seeking inspiration for your next project, this sample app is a valuable resource.

7. Future Enhancement:

To continually improve and adapt to user needs, several enhancements are planned for the future development of Adaptive Mail:

- **Rich Media Support**: Allow users to send and receive images, videos, and audio messages.
- **Emojis and Stickers**: Integrate a library of emojis and stickers to make conversations more expressive.
- **Message Reactions**: Enable users to react to messages with predefined or custom reactions.
- **Dark Mode**: Implement a dark mode to improve usability in low-light conditions and enhance visual appeal.
- **Customizable Themes**: Allow users to personalize the app's look and feel with different themes and color schemes.
- **Improved Notifications**: Enhance notifications with quick reply options and more detailed previews.
- • **Optimized Load Times**: Improve the speed and responsiveness of the app, especially when loading large message histories.
- • **Cloud Synchronization**: Enable synchronization of messages across multiple devices using cloud storage solutions.
- • **Offline Mode**: Implement offline capabilities so users can access and compose messages without an internet connection.
- • **End-to-End Encryption**: Ensure that all messages are securely encrypted to protect user privacy.
- • **Advanced Authentication**: Implement biometric authentication (e.g., fingerprint or facial recognition) for enhanced security.
- • **Data Anonymization**: Add features to anonymize user data and messages for privacy-conscious users.