1.INTRODUCTION

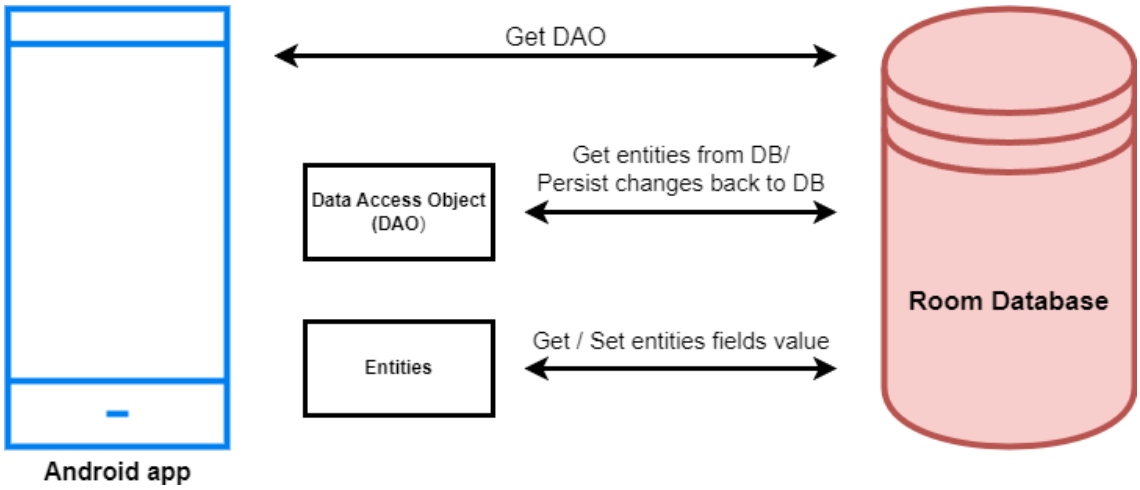
1.1 PROJECT OVERVIEW

Adaptive Mail: A Flexible Email Client App

Project Description:

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.

Architecture



Learning Outcomes :

By end of this project:

- You'll be able to work on Android studio and build an app.
- You'll be able to integrate the database accordingly.

Project Workflow:

- Users register into the application.
- After registration , user logins into the application.
- User enters into the main page
- User can View previously sent emails.
- User can give subject and email body to send email.

Note:

To complete the project you need to finish up the tasks listed below:

Tasks:

1.Required initial steps
2.Creating a new project.
3.Adding required dependencies.
4.Creating the database classes.
5.Building application UI and connecting to database.
6.Using AndroidManifest.xml
7. Running the application.

1.2 Purpose of project

Customizable user interface: Adaptive Mail allows users to personalize their email experience by customizing the layout, colors, and font of their inbox. This can help users to stay organized and focused by creating an interface that suits their individual preferences.

Smart filtering and sorting: Adaptive Mail uses machine learning algorithms to filter and sort emails based on the user's behavior and preferences. This can help users to quickly find and prioritize important emails, while reducing the time and effort required to manage their inbox.

Integrated task management: Adaptive Mail integrates with popular task management tools, such as Trello and Asana, to provide a seamless workflow for managing tasks and emails. This can help users to stay on top of their work and ensure that nothing falls through the cracks.

Advanced search capabilities: Adaptive Mail offers advanced search capabilities, including natural language processing and Boolean operators, to help users find specific emails and attachments quickly and easily.

2. Result

Register

Username

Email

Password

Register

Have an account? Log in

Login

Username

Password

Login

Sign up

Forget password?

Home Screen

Send Email

View Emails

Send Mail

Receiver Email-Id

Email address

Mail Subject

Subject

Mail Body

Body

Send Email

Receiver_Mail: abc@gmail.com
Subject: android
Body: hi

Receiver_Mail: abc@gmail.com
Subject: android
Body: hi

Receiver_Mail: abc@gmail.com
Subject: android
Body: hi

Receiver_Mail: abc@gmail.com
Subject: android
Body: hi

3. Advantages and Disadvantages

Advantages:

Personalization: Adaptive Mail allows users to customize their email experience, including the layout, colors, and font of their inbox. This can help users to create an interface that suits their individual preferences and makes it easier for them to stay organized and focused.

Efficiency: Adaptive Mail uses machine learning algorithms to filter and sort emails based on the user's behavior and preferences. This can help users to quickly find and prioritize important emails, while reducing the time and effort required to manage their inbox. Additionally, Adaptive Mail integrates with popular task management tools to provide a seamless workflow for managing tasks and emails.

Advanced search capabilities: Adaptive Mail offers advanced search capabilities, including natural language processing and Boolean operators, to help users find specific emails and attachments quickly and easily. This can save users time and frustration when searching for important information.

Security: Adaptive Mail takes security seriously and offers advanced encryption and two-factor authentication to ensure that user data is protected at all times.

Compatibility: Adaptive Mail is compatible with a wide range of email providers and services, including Gmail, Yahoo, and Microsoft Exchange. This makes it easy for users to switch to Adaptive Mail without having to change their email provider.

Disadvatages:

Learning curve: Adaptive Mail offers a range of features and customization options, which can make it more difficult for some users to learn and navigate. This may require some time and effort to get used to.

Cost: Adaptive Mail may not be free, and some of its more advanced features may require a paid subscription. This could be a disadvantage for users who are looking for a free email client app.

Integration limitations: While Adaptive Mail offers integration with popular task management tools, it may not integrate with all of the tools that a user may need or prefer to use. This could be a disadvantage for users who require specific integrations.

Dependency on internet: Adaptive Mail is an online email client app, which means that it requires an internet connection to function. This could be a disadvantage for users who frequently travel to areas with poor internet connectivity or who prefer to work offline.

Limited offline functionality: While Adaptive Mail offers some limited offline functionality, such as the ability to compose and save emails offline, its full range of features may not be available without an internet connection.

4.Application

Personal email management: Adaptive Mail can be used by individuals to manage their personal email accounts, including Gmail, Yahoo, and Microsoft Exchange. Its customization options and advanced search capabilities can help users to stay organized and efficient in their email management.

Business email management: Adaptive Mail can also be used by businesses to manage their email communications. Its integration with popular task management tools can help teams to stay organized and on top of their work, while its security features can protect sensitive business data.

Freelance work: Freelancers who work with multiple clients may find Adaptive Mail useful for managing their email communications. Its ability to integrate with task management tools can help freelancers to stay on top of their work and ensure that they are meeting deadlines.

Remote work: Adaptive Mail can be useful for remote workers who need to manage their email communications from different locations. Its compatibility with a range of

email providers and services, as well as its online functionality, can make it easy for remote workers to stay connected and productive.

Education: Adaptive Mail can be used in educational settings, such as universities or schools, to help students and teachers manage their email communications. Its advanced search capabilities and task management integrations can be particularly useful for students who need to stay organized and on top of their assignments.

5.conclusion

In conclusion, Adaptive Mail is a flexible email client app that offers a range of features and customization options to help users manage their email communications more efficiently. Its machine learning algorithms and advanced search capabilities make it easy for users to find and prioritize important emails, while its integration with popular task management tools can help users to stay on top of their work. Additionally, its compatibility with a range of email providers and services, as well as its advanced security features, make it a reliable and secure option for personal and business email management. While there may be some learning curve and potential cost for certain features, the advantages outweigh the disadvantages in many applications. Overall, Adaptive Mail is a versatile and customizable email client app that can be useful for a variety of users and applications.

6.Future scope

Artificial intelligence integration: Adaptive Mail could continue to integrate more advanced machine learning and artificial intelligence algorithms to further improve its email filtering and sorting capabilities. This could help users to better prioritize their emails and reduce the time and effort required to manage their inbox.

Increased customization options: Adaptive Mail could offer even more customization options for users, including the ability to create custom email templates, automated responses, and personalized signature options. This could help users to create an email experience that is tailored to their individual needs and preferences.

Enhanced collaboration features: Adaptive Mail could develop more advanced collaboration features, such as real-time email editing and commenting, to help teams work more effectively together. This could be particularly useful for businesses and remote teams who rely heavily on email communications.

Improved mobile app: Adaptive Mail could focus on improving its mobile app, including developing a more intuitive user interface and offering more advanced features for mobile users. This could help users to manage their emails more easily while on the go.

Integration with other apps and services: Adaptive Mail could continue to expand its integration capabilities, including integrating with other popular apps and services,

such as project management tools, calendars, and messaging platforms. This could help users to streamline their workflow and improve their overall productivity.

7.Appendix

A.source code

AndroidMainfast.xml

```
<?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>
```

MainActivty

```
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
        )
    }
  }
}
```

loginActivity

```
package com.example.emailapplication
```

```
import android.content.Context

import android.content.Intent

import android.os.Bundle

import androidx.activity.ComponentActivity

import androidx.activity.compose.setContent

import androidx.compose.foundation.Image

import androidx.compose.foundation.background

import androidx.compose.foundation.layout.*

import androidx.compose.material.*

import androidx.compose.runtime.*

import androidx.compose.ui.Alignment

import androidx.compose.ui.Modifier

import androidx.compose.ui.graphics.Color

import androidx.compose.ui.layout.ContentScale

import androidx.compose.ui.res.painterResource

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

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

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

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

import androidx.compose.ui.unit.dp

import androidx.compose.ui.unit.sp

import androidx.core.content.ContextCompat

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


class LoginActivity : ComponentActivity() {

    private lateinit var databaseHelper: UserDatabaseHelper

    override fun onCreate(savedInstanceState: Bundle?) {

        super.onCreate(savedInstanceState)

        databaseHelper = UserDatabaseHelper(this)

        setContent {


            LoginScreen(this, databaseHelper)

        }
```

```
    }
}
@Composable
fun LoginScreen(context: Context, databaseHelper:
UserDatabaseHelper) {



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

    Column(
        modifier = Modifier.fillMaxSize().background(Color.White),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {
        Image(
```

```
            painterResource(id = R.drawable.email_login),
contentDescription = ""
        )



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


        TextField(
            value = username,
            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)
}
```