A Demonstration of Text Input and Validation with Android Compose

INTRODUCTION

1.1 Overview:

In this demonstration, we will explore how to implement text input and validation using Android Compose, the modern UI toolkit introduced by Google for Android app

development. Text input and validation are essential components of many apps, and Compose provides powerful tools and APIs to create dynamic and interactive user

interfaces with a declarative approach.

We will cover the following topics in this demonstration:

Text Input: We will start by creating a text input field using Compose's TextField component. We will learn how to customize its appearance, behavior,

and input types. We will also explore how to handle input events, set up hints, and interact with the keyboard.

Input Validation: Next, we will delve into input validation, where we will demonstrate how to validate user input to ensure it meets certain criteria.

We will implement validation logic using Compose's built-in functions and extensions, and show how to provide feedback to users on input validation errors.

Error Handling: We will discuss error handling strategies for text input and validation, including how to display error messages, highlight invalid input,

and reset input fields when errors occur. We will also cover handling different types of errors, such as runtime exceptions, and recovering gracefully from them.

Testing: We will touch upon the importance of testing text input and validation in your app. We will demonstrate how to write unit tests for text input fields and

input validation logic using Compose's testing utilities, and ensure reliable input handling in your app.

Throughout this demonstration, we will provide practical examples and code snippets to illustrate the concepts and techniques discussed.

By the end of this demonstration, you will have a solid understanding of how to implement text input and validation with Android Compose,

and how to create a seamless user experience in your apps. Let's dive in and explore the world of text input and validation with Compose!

1.2 Purpose:

The purpose of the demonstration on text input and validation with Android Compose is to provide developers with a practical guide on how to implement text input and

validation in their Android apps using Compose. The demonstration aims to cover various aspects of text input and validation, including creating text input fields,

customizing their appearance and behavior, validating user input, handling errors, and

testing input handling logic.

The overall purpose of the demonstration is to help developers understand how to:

Create text input fields using Compose's TextField component.

Customize the appearance, behavior, and input types of text input fields.

Implement input validation logic using Compose's built-in functions and extensions.

Provide feedback to users on input validation errors.

Handle errors gracefully and reset input fields when errors occur.

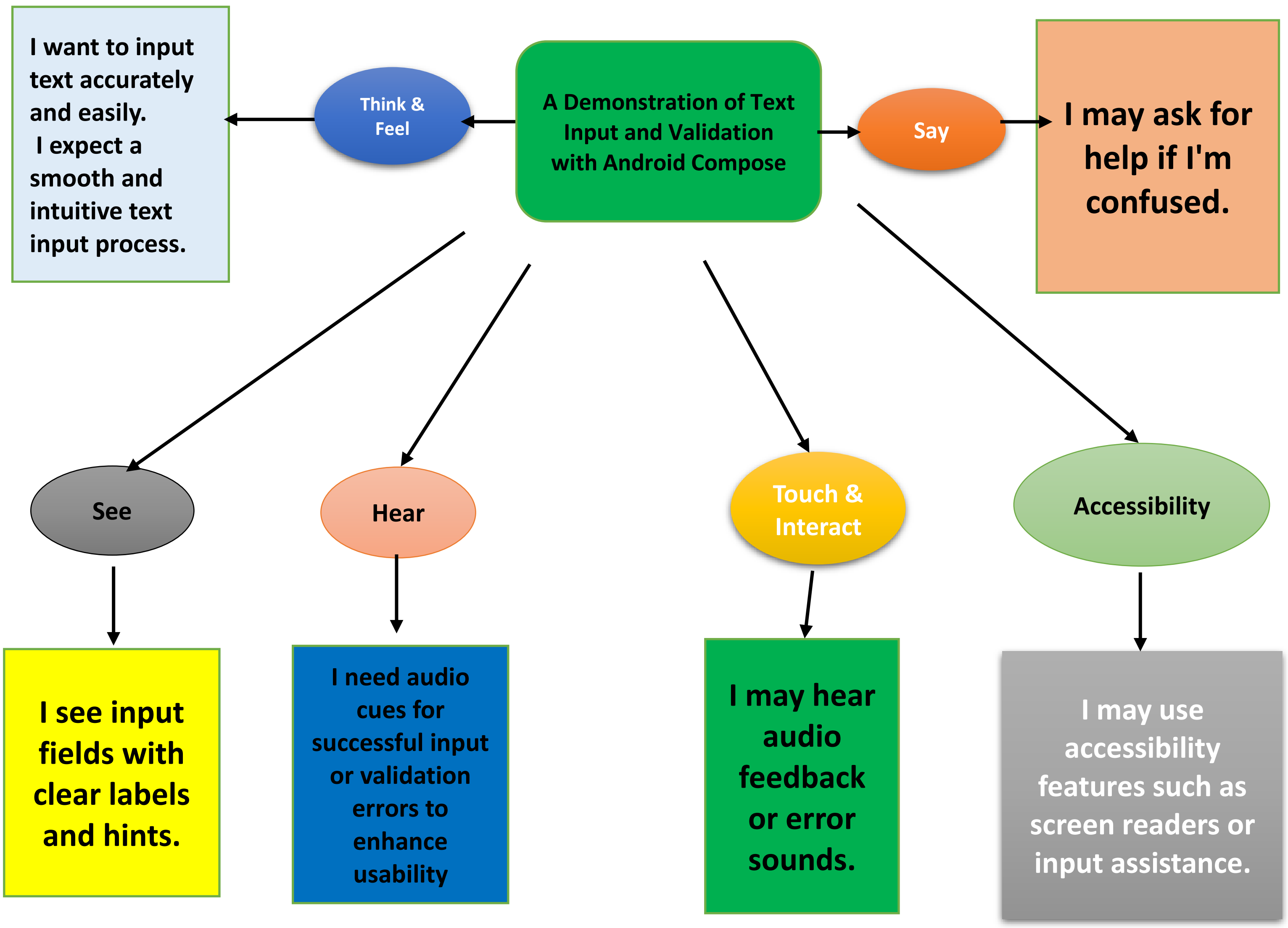
Write unit tests for text input fields and input validation logic using Compose's testing utilities.

The demonstration will provide practical examples, code snippets, and guidance on best practices to help developers implement text input and validation effectively in their Android apps. By the end of the demonstration, developers should have a solid understanding of how to use Android Compose to create robust

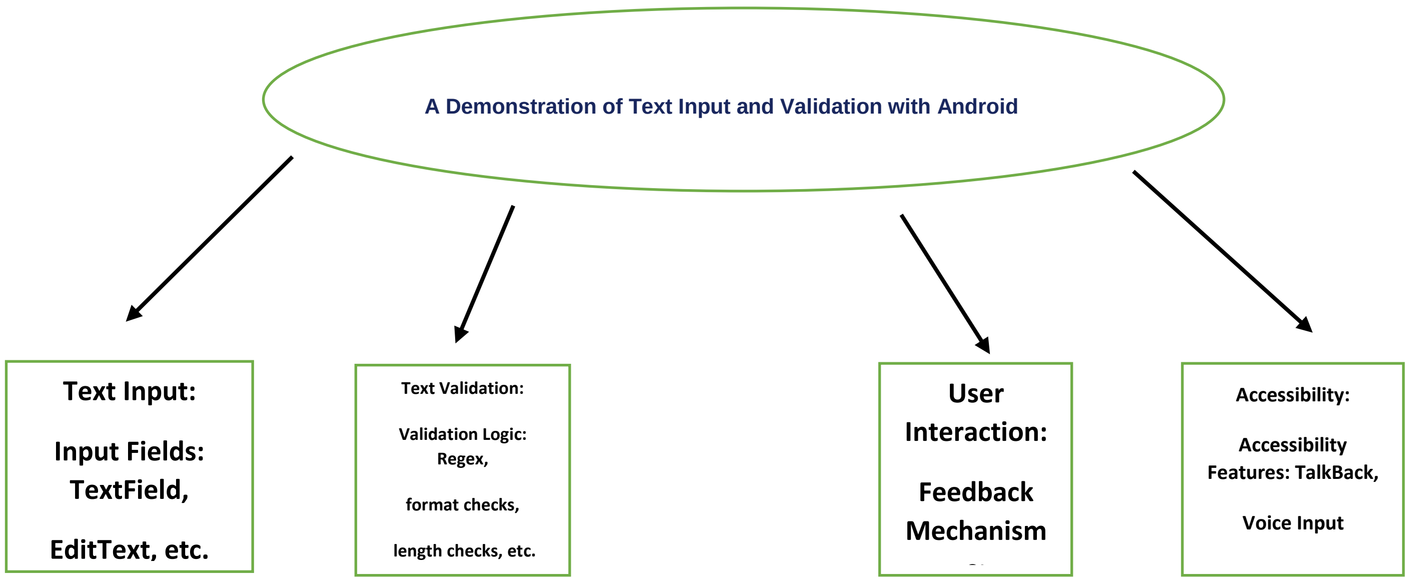
and user-friendly text input and validation features in their apps.

2.0 Problem Definition & Design Thinking

2.1 Empathy Map

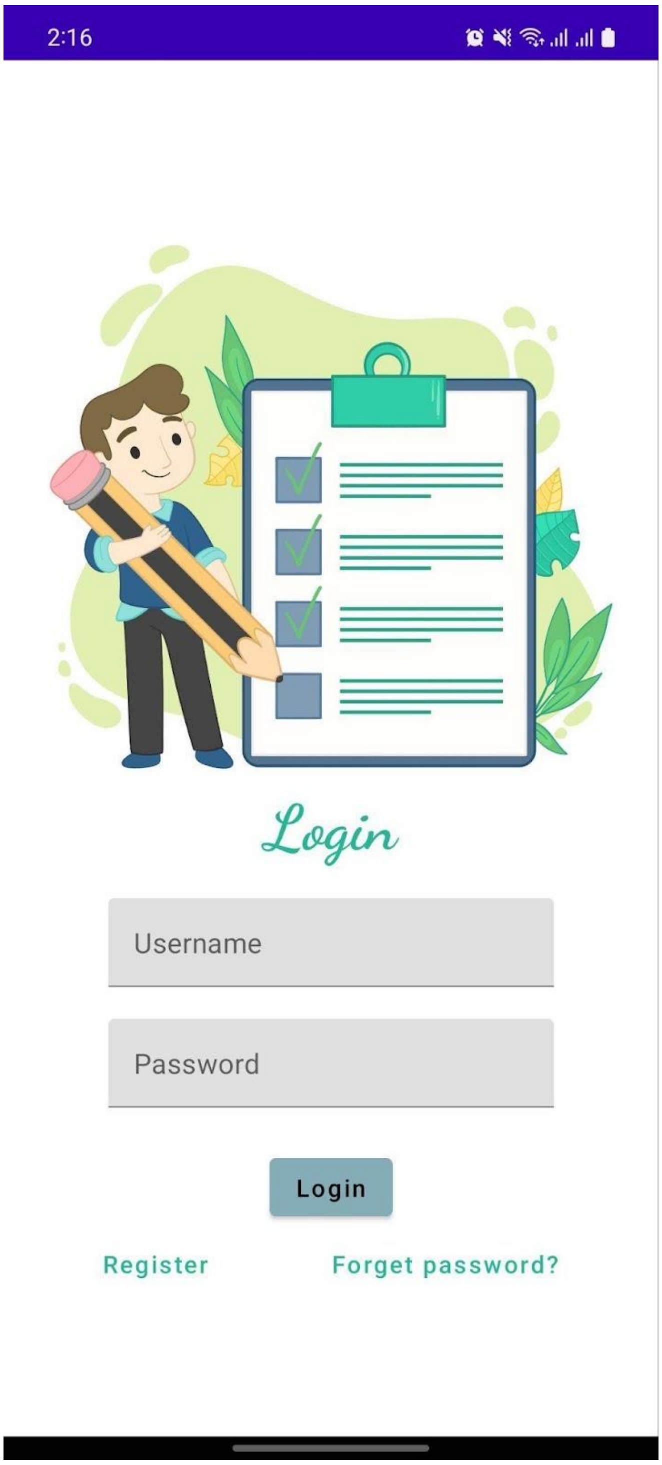

A Demonstration of Text Input and Validation with Android Compose
Think & Feel
I want to input text accurately and easily. I expect a smooth and intuitive text input process.
Say
I may ask for help if I'm confused.
See
I see input fields with clear labels and hints.
Hear
I need audio cues for successful input or validation errors to enhance usability
Touch & Interact
I may hear audio feedback or error sounds.
Accessibility
I may use accessibility features such as screen readers or input assistance.

2.2 IDEATION AND BRAIN STORMING MAP:

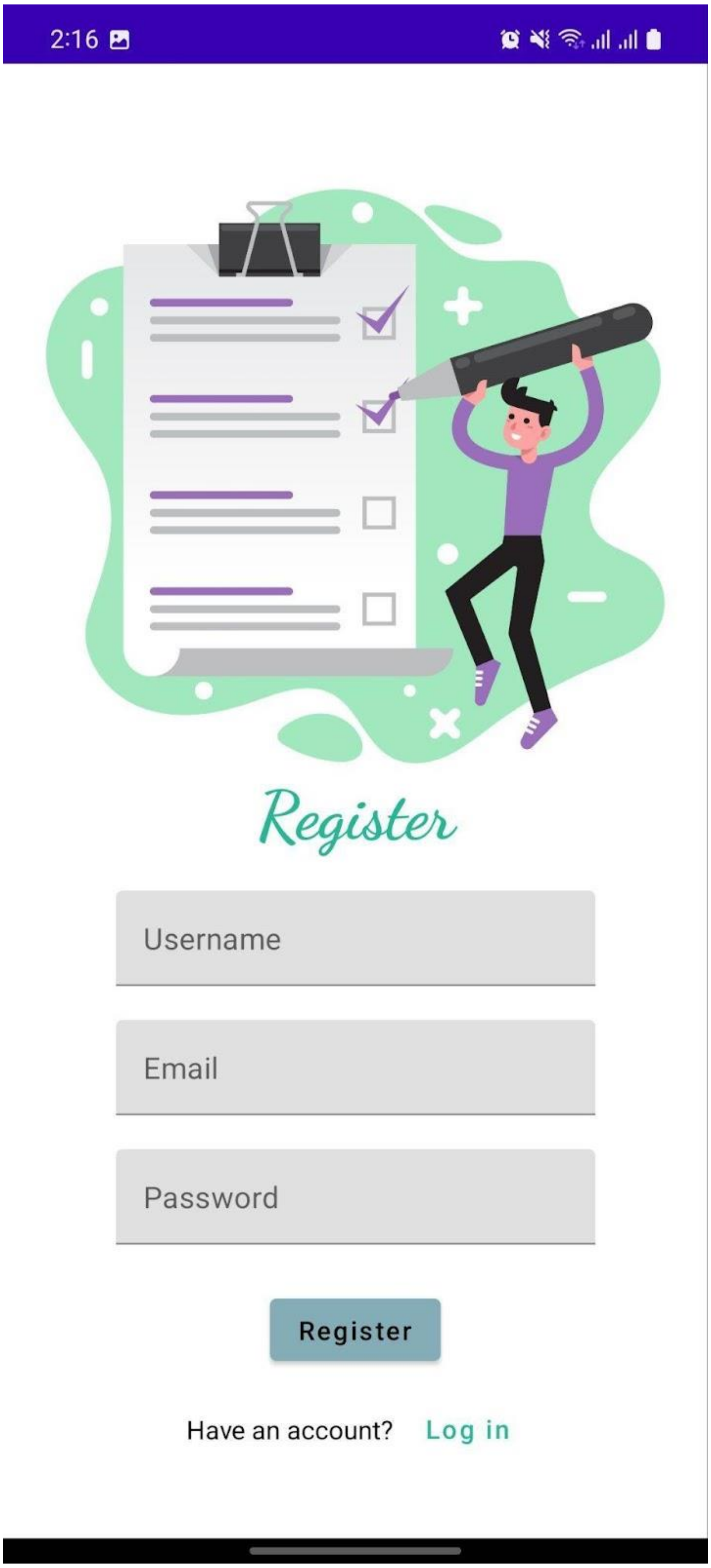

A Demonstration of Text Input and Validation with Android
Text Input: Input Fields: TextField, EditText, etc.
Text Validation: Validation Logic: Regex, format checks, length checks, etc.
User Interaction: Feedback Mechanism
Accessibility: Accessibility Features: TalkBack, Voice Input

3.0 Result:

Login Page :

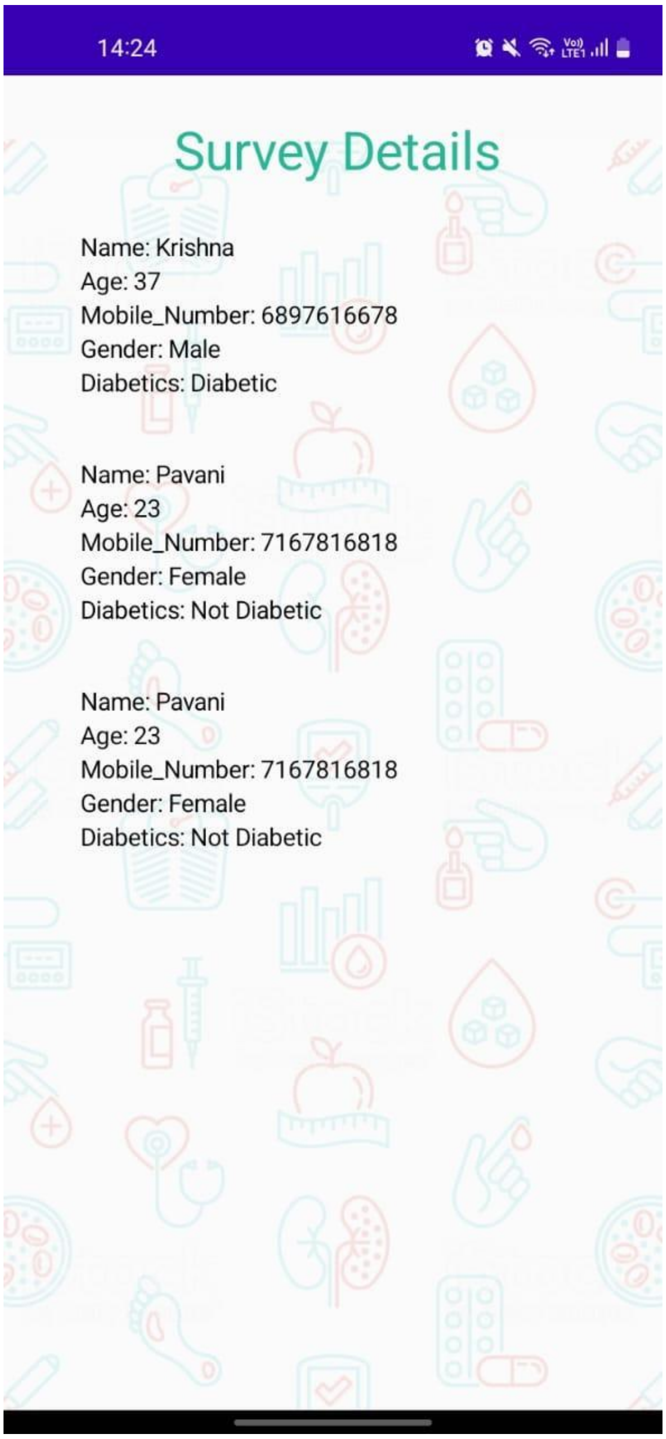


Register Page :


2:16
Register
Username
Email
Password
Register
Have an account?
Log in

After logging in with Admin Credentials which are hard coded. Password must be "admin" .

Admin page:


14:24

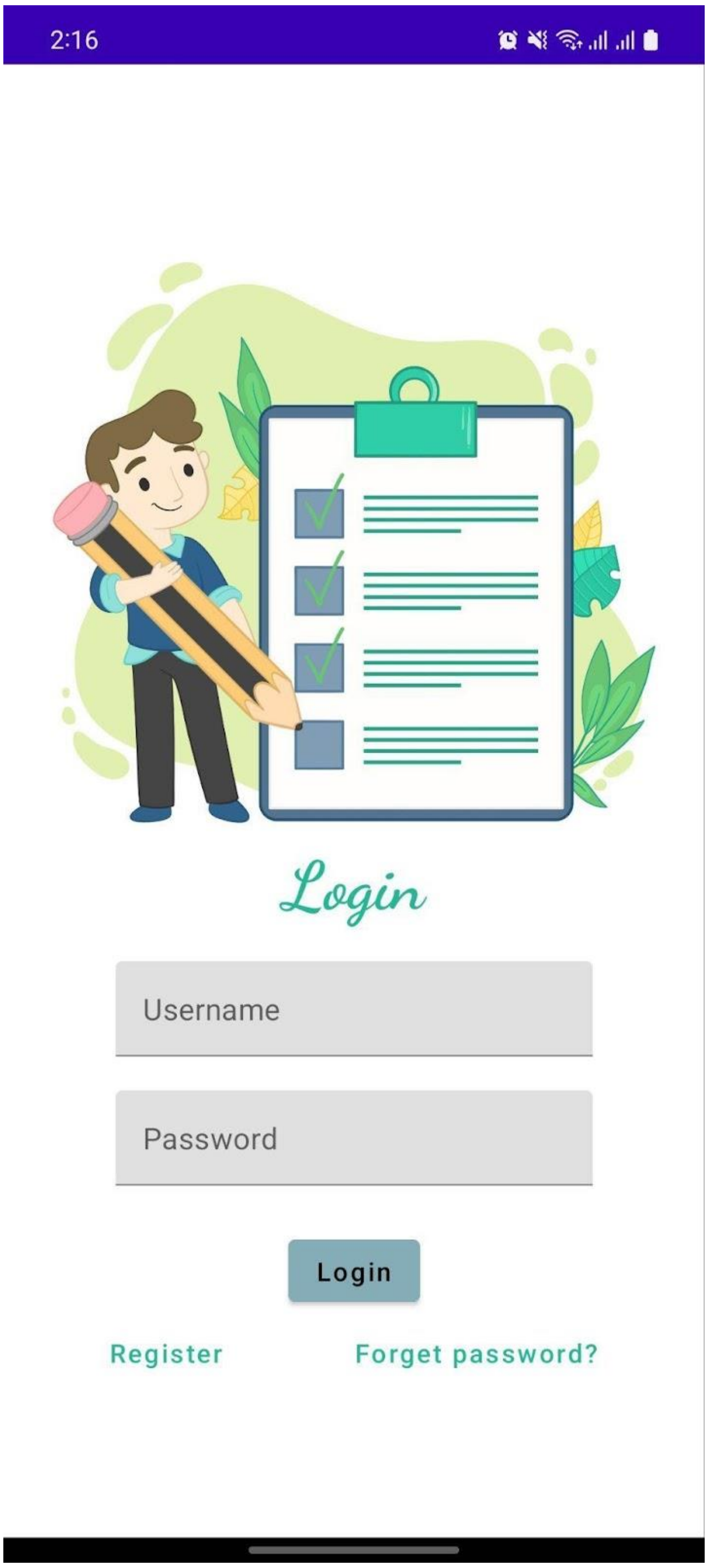
Survey Details

Name: Krishna
Age: 37
Mobile_Number: 6897616678
Gender: Male
Diabetics: Diabetic

Name: Pavani
Age: 23
Mobile_Number: 7167816818
Gender: Female
Diabetics: Not Diabetic

Name: Pavani
Age: 23
Mobile_Number: 7167816818
Gender: Female
Diabetics: Not Diabetic

User Module:
Login Page :


2:16

Login

Username

Password

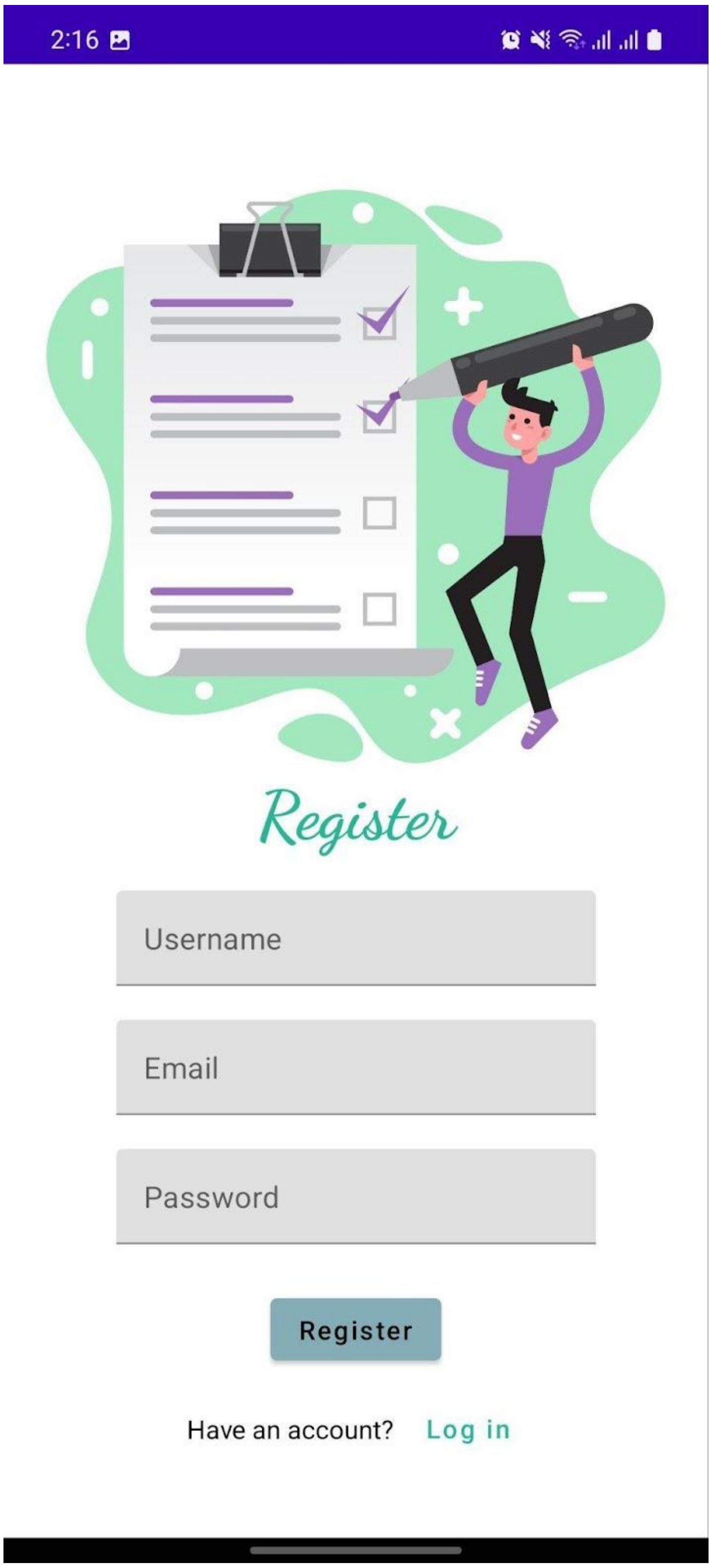
Login

Register

Forget password?

Register Page :

Main Page :


2:16

Username

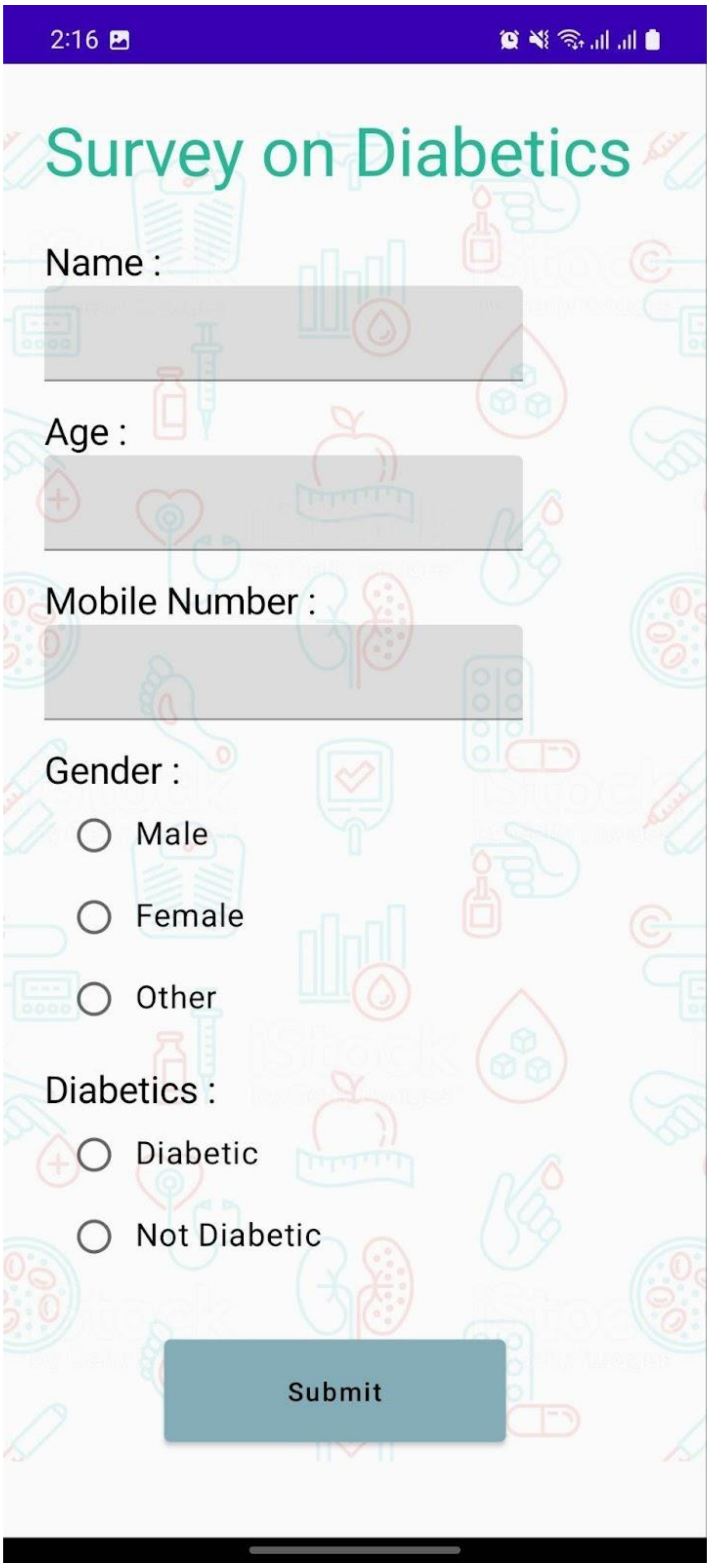
Email

Password

Register

Have an account? Log in

Main Page :



2:16

Survey on Diabetics

Name :

Age :

Mobile Number :

Gender :

○ Male

○ Female

○ Other

Diabetics :

○ Diabetic

○ Not Diabetic

4.0 Advantages & Disadvantage

Advantage:

Declarative UI: Android Compose follows a declarative UI approach, allowing developers to define the user interface using a more concise and intuitive syntax. This makes it easier to create text input fields with the desired UI elements, such as labels, hints, and error messages, in a more organized and efficient way.

Reactive UI updates: Android Compose supports reactive UI updates, which means that any changes in the input data or validation state can trigger automatic updates in the UI. This enables real-time validation and feedback to the user, helping them quickly identify and correct input errors.

Easy handling of input events: Android Compose provides built-in support for handling input events, such as keyboard input, touch events, and gestures, making it straightforward to capture and process user input in text fields. This allows for seamless and intuitive user interactions with the text input fields.

Form validation made simple: Android Compose allows developers to implement form validation logic in a more concise and readable way, making it easier to validate input data based on custom business rules or pre-defined validation patterns. This helps ensure that only valid data is accepted and processed by the app, reducing the risk of errors and improving data integrity.

Flexibility in input types: Android Compose offers flexibility in handling different types of input, such as text, numbers, dates, and special characters. It provides built-in support for input masks, input transformations, and input filters, making it easier to handle diverse input requirements, such as formatting phone numbers, dates, or credit card numbers.

Modular and reusable components: Android Compose allows developers to create reusable UI components, including text input fields, which can be easily shared across different parts of the app. This promotes code reusability and maintainability, making it easier to update or modify text input and validation behavior in one place and see the changes reflected throughout the app.

Rich theming and styling options: Android Compose provides rich theming and styling options, allowing developers to customize the appearance of text input fields to match the app's visual design. This enables a consistent and visually appealing user experience across the app, enhancing the overall app's look and feel.

Overall, Android Compose offers several advantages for text input and validation in mobile apps, including a declarative UI approach, reactive UI updates, easy handling of input events, simplified form validation, flexibility in input types, modular and reusable components, and rich theming options. These advantages can help improve the user experience and streamline the development process when implementing text input and validation in Android Compose apps.

Disadvantage:

Now, one potential disadvantage of using Android Compose for text input and validation is that the validation logic can become scattered and difficult to manage as the form complexity increases. Here are some reasons why this can be a disadvantage:

Code duplication: In Android Compose, you may need to define the same validation logic in multiple composable functions if you have similar input fields with similar validation requirements. This can result in code duplication, making it harder to maintain and update the validation logic in case of changes or bug fixes.

Lack of centralized validation: Unlike some other UI frameworks, Android Compose does not provide a built-in, centralized mechanism for managing validation logic. This means that you may need to manually implement and manage validation logic in each individual composable function, which can lead to scattered and hard-to-maintain validation code.

Increased complexity: As the form complexity increases, the validation logic can become more complex and intertwined with the UI logic, making it harder to understand and debug. This can result in longer development time, increased chances of introducing bugs, and reduced code readability.

Limited reusability: In some cases, you may want to reuse the same validation logic across different parts of your app or even across different apps. However, with Android Compose, the validation

logic is typically tightly coupled with the UI logic, which can make it challenging to extract and reuse the validation logic in other parts of your app or in other apps.

Maintenance overhead: With Android Compose being a relatively new technology, it may require frequent updates and changes as the framework evolves. This can result in additional maintenance overhead, especially if the validation logic is scattered across multiple composable functions and needs to be updated in multiple places.

It's worth mentioning that these disadvantages can be mitigated with careful design and architecture choices, such as encapsulating validation logic in separate classes or functions, using a central validation framework, or leveraging existing libraries or third-party solutions. However, it's important to be aware of the potential challenges and consider them when deciding to use Android Compose for text input and validation in a mobile app.

5.0 Applications:

simple demonstration of how you can implement text input and validation using Android Compose in an Android app.

Let's assume we want to create a registration form with three input fields: name, email, and password. We will implement basic validation for each input field, including checking for empty fields and validating the email format and password strength.

Certainly! Here's a theoretical demonstration of text input and validation in an Android Compose application.

In an Android Compose application, text input can be achieved using the TextField component.

The TextField component allows users to enter text and displays the entered text.

You can specify the initial value of the text input by setting the value parameter of the TextField component.

Users can modify the text input by typing on the keyboard or using an input method, and the changes are reflected in the value parameter.

You can use the onValueChange callback of the TextField component to handle the changes in the text input and update your application state accordingly.

Text Validation:

Text validation is the process of verifying if the entered text meets certain criteria or constraints.

In an Android Compose application, text validation can be implemented by adding validation logic to the onValueChange callback of the TextField component.

You can check the entered text against the desired criteria, such as checking for empty input, checking for a minimum/maximum length, or validating against a regular expression.

If the entered text does not meet the validation criteria, you can update your application state to reflect the validation error, such as displaying an error message to the user.

You can use state variables, such as mutable state or view model, to keep track of the current value of the text input and any validation errors.

Once the input is validated, you can take appropriate action, such as submitting the form, storing the input in a database, or performing any other desired functionality.

Overall, text input and validation in an Android Compose application involve using the TextField component to capture user input, implementing validation logic in the onValueChange callback, and updating the application state based on the validation result.

In the above example, we use the TextField composable from Android Compose to create input fields for name, email, and password. We use the value and onValueChange parameters to store and update the local state for each input field. We also use the Button composable to create a submit button, which triggers the validation logic when clicked.

The validation logic checks for empty fields using the isNotEmpty() function, and then calls the isValidEmail() and isValidPassword() helper functions to perform email format and password strength validation, respectively. You can implement your own validation logic inside these helper functions as per your app's requirements.

Note: This is a simplified example and may not cover all the edge cases or error handling mechanisms. In a real-world application, you may need to consider additional validation scenarios and implement appropriate error handling mechanisms based on your specific use case.

I hope this demonstrates how you can implement text input and validation using Android Compose in an Android app

6.0 CONCLUSION:

In conclusion, Android Compose provides a powerful and flexible way to implement text input and validation in your applications. It offers a declarative approach to building user interfaces, making it easy to create dynamic and interactive forms with input fields and validation logic.

With Android Compose, you can easily create text input fields using TextField composable, and apply various validation techniques such as input length, format, and content validation using state and modifier functions. You can also display error messages or feedback to users based on the validation results.

The advantages of using Android Compose for text input and validation include its simplicity, reusability, and flexibility. Compose allows you to define UI components as composable functions, making them highly reusable across your application. It also provides a more concise and expressive way to manage the state and behavior of UI components, making it easier to implement complex validation logic.

However, it's worth noting that Android Compose is still a relatively new technology and may have some limitations, such as limited documentation and community support compared to the older

XML-based layout system. Additionally, as with any UI implementation, it's important to carefully consider user experience (UX) principles and accessibility guidelines when implementing text input and validation in your application.

Overall, Android Compose provides a modern and efficient way to implement text input and validation in your Android applications, and it's worth exploring and experimenting with in your projects.

7.0 FUTURLE SCOPE:

The future scope of text input and validation with Android Compose is quite promising. As a relatively new technology, Android Compose is continuously evolving and improving with regular updates from Google. Here are some potential areas of future development and advancements in text input and validation with Android Compose:

Enhanced Input Field Customization: Android Compose may offer more customization options for input fields, such as adding custom themes, styles, and animations, to provide a consistent and personalized user experience.

Advanced Validation Techniques: Compose may introduce more advanced validation techniques, such as real-time validation, regex-based validation, and validation based on external data sources, to enable more complex and dynamic validation scenarios.

Improved Accessibility Support: Compose may further enhance its accessibility features, such as providing built-in support for accessibility labels, hints, and error messages for input fields, to ensure that apps built with Compose are accessible to all users, including those with disabilities.

Integration with ML/AI: Compose may integrate with machine learning and artificial intelligence technologies to enable intelligent text input and validation, such as auto-suggesting input based on user behavior or predicting input based on historical data.

Error Handling and Error Recovery: Compose may provide improved error handling and error recovery mechanisms for text input and validation, such as handling network errors, server-side validation errors, and providing options for users to recover from input errors.

Cross-platform Support: Compose may expand its support to other platforms, such as desktop and web, allowing developers to use the same codebase and UI components for text input and validation across multiple platforms.

Community Contributions: As the community around Android Compose continues to grow, more contributions from developers and designers are expected, leading to the development of new libraries, tools, and best practices for text input and validation in Compose-based applications.

In conclusion, the future scope of text input and validation with Android Compose is bright, with potential advancements in

customization, validation techniques, accessibility support, ML/AI integration, error handling, cross-platform support, and community contributions. It's an exciting time for Android developers to explore and leverage the capabilities of Android Compose for creating modern and user-friendly text input and validation experiences in their applications.

8.APPENDIX:

AndroidManifest.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.SurveyApplication"
    tools:targetApi="31">
```

```
<activity
   android:name=".RegisterActivity"
   android:exported="false"
   android:label="@string/title_activity_register"
   android:theme="@style/Theme.SurveyApplication" />
<activity
   android:name=".MainActivity"
   android:exported="false"
   android:label="MainActivity"
   android:theme="@style/Theme.SurveyApplication" />
<activity
   android:name=".AdminActivity"
   android:exported="false"
   android:label="@string/title_activity_admin"
   android:theme="@style/Theme.SurveyApplication" />
<activity
   android:name=".LoginActivity"
   android:exported="true"
   android:label="@string/app_name"
   android:theme="@style/Theme.SurveyApplication">
   <intent-filter>
      <action android:name="android.intent.action.MAIN" />
```

```
                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

AdminActivity.kt:

```
         package com.example.surveyapplication

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.lazy.LazyColumn
import androidx.compose.foundation.lazy.LazyRow
import androidx.compose.foundation.lazy.items
import androidx.compose.material.MaterialTheme
import androidx.compose.material.Surface
import androidx.compose.material.Text
```

```
}

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.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import
com.example.surveyapplication.ui.theme.SurveyApplicationTheme


class AdminActivity : ComponentActivity() {
    private lateinit var databaseHelper: SurveyDatabaseHelper
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        databaseHelper = SurveyDatabaseHelper(this)
        setContent {
            val data = databaseHelper.getAllSurveys();
            Log.d("swathi", data.toString())
            val survey = databaseHelper.getAllSurveys()
            ListListScopeSample(survey)
        }
    }
}
```

```
@Composable
fun ListListScopeSample(survey: List<Survey>) {

   Image(
      painterResource(id = R.drawable.background),
contentDescription = "",
      alpha =0.1F,
      contentScale = ContentScale.FillHeight,
      modifier = Modifier.padding(top = 40.dp)
   )

   Text(
      text = "Survey Details",
      modifier = Modifier.padding(top = 24.dp, start = 106.dp, bottom
= 24.dp),
      fontSize = 30.sp,
      color = Color(0xFF25b897)
   )
   Spacer(modifier = Modifier.height(30.dp))
   LazyRow(
      modifier = Modifier
         .fillMaxSize()
         .padding(top = 80.dp),
        horizontalArrangement = Arrangement.SpaceBetween
    ) {
        item {

            LazyColumn {
                items(survey) { survey ->
                    Column(
                        modifier = Modifier.padding(
                            top = 16.dp,
                            start = 48.dp,
                            bottom = 20.dp
                        )
                    ) {
                        Text("Name: ${survey.name}")
                        Text("Age: ${survey.age}")
                        Text("Mobile_Number: ${survey.mobileNumber}")
                        Text("Gender: ${survey.gender}")
                        Text("Diabetics: ${survey.diabetics}")
                    }
                }
            }
        }
    }
}
```

LoginActivity.kt:

```
        package com.example.surveyapplication


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.tooling.preview.Preview

import androidx.compose.ui.unit.dp

import androidx.compose.ui.unit.sp

import androidx.core.content.ContextCompat
```

```
import
com.example.surveyapplication.ui.theme.SurveyApplicationTheme

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.survey_login),
contentDescription = "")

        Text(
            fontSize = 36.sp,
            fontWeight = FontWeight.ExtraBold,
            fontFamily = FontFamily.Cursive,
            color = Color(0xFF25b897),
            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()
                  }
                  if (user != null && user.password == "admin") {
                    error = "Successfully log in"
                    context.startActivity(
                      Intent(
                        context,
                        AdminActivity::class.java
                      )
                    )
                  }
                  else {
                    error =  "Invalid username or password"
                  }
```

```
            } else {
                error = "Please fill all fields"
            }
        },
        colors = ButtonDefaults.buttonColors(backgroundColor =
Color(0xFF84adb8)),
        modifier = Modifier.padding(top = 16.dp)
    ) {
        Text(text = "Login")
    }
    Row {
        TextButton(onClick = {context.startActivity(
            Intent(
                context,
                RegisterActivity::class.java
            )
        )}
        )
        { Text(color = Color(0xFF25b897),text = "Register") }
        TextButton(onClick = {
        })

        {
                Spacer(modifier = Modifier.width(60.dp))
                Text(color = Color(0xFF25b897),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.surveyapplication

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.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.style.TextAlign
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import
com.example.surveyapplication.ui.theme.SurveyApplicationTheme

class MainActivity : ComponentActivity() {
    private lateinit var databaseHelper: SurveyDatabaseHelper
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        databaseHelper = SurveyDatabaseHelper(this)
        setContent {
            FormScreen(this, databaseHelper)
        }
    }
}

@Composable
```

```
fun FormScreen(context: Context, databaseHelper:
SurveyDatabaseHelper) {

    Image(
        painterResource(id = R.drawable.background),
contentDescription = "",
        alpha =0.1F,
        contentScale = ContentScale.FillHeight,
        modifier = Modifier.padding(top = 40.dp)
        )




    // Define state for form fields
    var name by remember { mutableStateOf("") }
    var age by remember { mutableStateOf("") }
    var mobileNumber by remember { mutableStateOf("") }
    var genderOptions = listOf("Male", "Female", "Other")
    var selectedGender by remember { mutableStateOf("") }
    var error by remember { mutableStateOf("") }
    var diabeticsOptions = listOf("Diabetic", "Not Diabetic")
    var selectedDiabetics by remember { mutableStateOf("") }
```

```
Column(
    modifier = Modifier.padding(24.dp),
    horizontalAlignment = Alignment.Start,
    verticalArrangement = Arrangement.SpaceEvenly
) {

    Text(
        fontSize = 36.sp,
        textAlign = TextAlign.Center,
        text = "Survey on Diabetics",
        color = Color(0xFF25b897)
    )

    Spacer(modifier = Modifier.height(24.dp))

    Text(text = "Name :", fontSize = 20.sp)
    TextField(
        value = name,
        onValueChange = { name = it },
    )

    Spacer(modifier = Modifier.height(14.dp))

    Text(text = "Age :", fontSize = 20.sp)
```

```
TextField(
    value = age,
    onValueChange = { age = it },
)

Spacer(modifier = Modifier.height(14.dp))

Text(text = "Mobile Number :", fontSize = 20.sp)
TextField(
    value = mobileNumber,
    onValueChange = { mobileNumber = it },
)

Spacer(modifier = Modifier.height(14.dp))

Text(text = "Gender :", fontSize = 20.sp)
RadioGroup(
    options = genderOptions,
    selectedOption = selectedGender,
    onSelectedChange = { selectedGender = it }
)

Spacer(modifier = Modifier.height(14.dp))
```

```
        Text(text = "Diabetics :", fontSize = 20.sp)
        RadioGroup(
            options = diabeticsOptions,
            selectedOption = selectedDiabetics,
            onSelectedChange = { selectedDiabetics = it }
        )

        Text(
            text = error,
            textAlign = TextAlign.Center,
            modifier = Modifier.padding(bottom = 16.dp)
        )
        // Display Submit button
        Button(
            onClick = {  if (name.isNotEmpty() && age.isNotEmpty() &&
mobileNumber.isNotEmpty() && genderOptions.isNotEmpty() &&
diabeticsOptions.isNotEmpty()) {
                val survey = Survey(
                    id = null,
                    name = name,
                    age = age,
                    mobileNumber = mobileNumber,
                    gender = selectedGender,
                    diabetics = selectedDiabetics
            )
            databaseHelper.insertSurvey(survey)
            error = "Survey Completed"


         } else {
            error = "Please fill all fields"
         }
         },
         colors = ButtonDefaults.buttonColors(backgroundColor =
Color(0xFF84adb8)),
         modifier = Modifier.padding(start = 70.dp).size(height = 60.dp,
width = 200.dp)
      ) {
         Text(text = "Submit")
      }
   }
}
@Composable
fun RadioGroup(
   options: List<String>,
   selectedOption: String?,
   onSelectedChange: (String) -> Unit
) {
   Column {
        options.forEach { option ->
            Row(
                Modifier
                    .fillMaxWidth()
                    .padding(horizontal = 5.dp)
            ) {
                RadioButton(
                    selected = option == selectedOption,
                    onClick = { onSelectedChange(option) }
                )
                Text(
                    text = option,
                    style = MaterialTheme.typography.body1.merge(),
                    modifier = Modifier.padding(top = 10.dp),
                    fontSize = 17.sp
                )
            }
        }
    }
}
```

RegisterActivity.kt:

```
        package com.example.surveyapplication
```

```
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.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.core.content.ContextCompat
import
com.example.surveyapplication.ui.theme.SurveyApplicationTheme
```

```
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.survey_signup),
contentDescription = "")

        Text(
            fontSize = 36.sp,
            fontWeight = FontWeight.ExtraBold,
            fontFamily = FontFamily.Cursive,
            color = Color(0xFF25b897),
            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(0xFF84adb8)),
            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(0xFF25b897),text = "Log in")
            }
        }
    }
}
private fun startLoginActivity(context: Context) {
    val intent = Intent(context, LoginActivity::class.java)
    ContextCompat.startActivity(context, intent, null)
}
```

Survey.kt:

```
    package com.example.surveyapplication

import androidx.room.ColumnInfo
import androidx.room.Entity
import androidx.room.PrimaryKey

@Entity(tableName = "survey_table")
data class Survey(
    @PrimaryKey(autoGenerate = true) val id: Int?,
    @ColumnInfo(name = "name") val name: String?,
    @ColumnInfo(name = "age") val age: String?,
    @ColumnInfo(name = "mobile_number") val mobileNumber: 
String?,
    @ColumnInfo(name = "gender") val gender: String?,
    @ColumnInfo(name = "diabetics") val diabetics: String?,

    )
```

SurveyDao.kt:

```
    package com.example.surveyapplication

import androidx.room.*

@Dao
interface SurveyDao {

    @Query("SELECT * FROM survey_table WHERE age = :age")
    suspend fun getUserByAge(age: String): Survey?

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertSurvey(survey: Survey)

    @Update
    suspend fun updateSurvey(survey: Survey)

    @Delete
    suspend fun deleteSurvey(survey: Survey)
}
```

SurveyDatabase.kt:

```
                    package com.example.surveyapplication

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

@Database(entities = [Survey::class], version = 1)
abstract class SurveyDatabase : RoomDatabase() {
    abstract fun surveyDao(): SurveyDao

    companion object {

        @Volatile
        private var instance: SurveyDatabase? = null

        fun getDatabase(context: Context): SurveyDatabase {
            return instance ?: synchronized(this) {
                val newInstance = Room.databaseBuilder(
                    context.applicationContext,
                    SurveyDatabase::class.java,
                    "user_database"
                ).build()
                instance = newInstance
                newInstance
            }
        }
    }
}
```

SurveyDatabaseHelper.kt:

```
  package com.example.surveyapplication
```

```
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 SurveyDatabaseHelper(context: Context) :
    SQLiteOpenHelper(context, DATABASE_NAME, null,
DATABASE_VERSION) {

    companion object {
        private const val DATABASE_VERSION = 1
        private const val DATABASE_NAME = "SurveyDatabase.db"


        private const val TABLE_NAME = "survey_table"
        private const val COLUMN_ID = "id"
        private const val COLUMN_NAME = "name"
        private const val COLUMN_AGE = "age"
        private const val COLUMN_MOBILE_NUMBER=
"mobile_number"
        private const val COLUMN_GENDER = "gender"
        private const val COLUMN_DIABETICS = "diabetics"
    }
    override fun onCreate(db: SQLiteDatabase?) {
        val createTable = "CREATE TABLE $TABLE_NAME (" +
                "$COLUMN_ID INTEGER PRIMARY KEY AUTOINCREMENT, "
+
                "$COLUMN_NAME TEXT, " +
                "$COLUMN_AGE TEXT, " +
                "$COLUMN_MOBILE_NUMBER TEXT, " +
                "$COLUMN_GENDER TEXT," +
                "$COLUMN_DIABETICS TEXT" +
                ")"

        db?.execSQL(createTable)
    }

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

    fun insertSurvey(survey: Survey) {
        val db = writableDatabase
        val values = ContentValues()
        values.put(COLUMN_NAME, survey.name)
        values.put(COLUMN_AGE, survey.age)
        values.put(COLUMN_MOBILE_NUMBER, survey.mobileNumber)
        values.put(COLUMN_GENDER, survey.gender)
        values.put(COLUMN_DIABETICS, survey.diabetics)
        db.insert(TABLE_NAME, null, values)
        db.close()
    }

    @SuppressLint("Range")
    fun getSurveyByAge(age: String): Survey? {
        val db = readableDatabase
        val cursor: Cursor = db.rawQuery("SELECT * FROM
$TABLE_NAME WHERE $COLUMN_AGE = ?", arrayOf(age))
        var survey: Survey? = null
        if (cursor.moveToFirst()) {
            survey = Survey(
                id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),
                name =
cursor.getString(cursor.getColumnIndex(COLUMN_NAME)),
                age =
cursor.getString(cursor.getColumnIndex(COLUMN_AGE)),
                mobileNumber =
cursor.getString(cursor.getColumnIndex(COLUMN_MOBILE_NUMBE
R)),
            gender =
cursor.getString(cursor.getColumnIndex(COLUMN_GENDER)),
            diabetics =
cursor.getString(cursor.getColumnIndex(COLUMN_DIABETICS)),
         )
      }
      cursor.close()
      db.close()
      return survey
   }
   @SuppressLint("Range")
   fun getSurveyById(id: Int): Survey? {
      val db = readableDatabase
      val cursor: Cursor = db.rawQuery("SELECT * FROM
$TABLE_NAME WHERE $COLUMN_ID = ?", arrayOf(id.toString()))
      var survey: Survey? = null
      if (cursor.moveToFirst()) {
         survey = Survey(
            id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),
            name =
cursor.getString(cursor.getColumnIndex(COLUMN_NAME)),
            age =
cursor.getString(cursor.getColumnIndex(COLUMN_AGE)),
            mobileNumber =
cursor.getString(cursor.getColumnIndex(COLUMN_MOBILE_NUMBE
R)),
            gender =
cursor.getString(cursor.getColumnIndex(COLUMN_GENDER)),
            diabetics =
cursor.getString(cursor.getColumnIndex(COLUMN_DIABETICS)),
         )
      }
      cursor.close()
      db.close()
      return survey
   }

   @SuppressLint("Range")
   fun getAllSurveys(): List<Survey> {
      val surveys = mutableListOf<Survey>()
      val db = readableDatabase
      val cursor: Cursor = db.rawQuery("SELECT * FROM
$TABLE_NAME", null)
      if (cursor.moveToFirst()) {
         do {
            val survey = Survey(
               cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),

cursor.getString(cursor.getColumnIndex(COLUMN_NAME)),
               cursor.getString(cursor.getColumnIndex(COLUMN_AGE)),
cursor.getString(cursor.getColumnIndex(COLUMN_MOBILE_NUMBE
R)),

cursor.getString(cursor.getColumnIndex(COLUMN_GENDER)),

cursor.getString(cursor.getColumnIndex(COLUMN_DIABETICS))
                )
                surveys.add(survey)
            } while (cursor.moveToNext())
        }
        cursor.close()
        db.close()
        return surveys
    }

}
```

User.kt:

```
package com.example.surveyapplication


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.surveyapplication

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.surveyapplication

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.surveyapplication


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
    }

}
```