

# Project Report Template

## Compose Input: A Demonstration of Text Input and Validation with Android Compose

### 1 INTRODUCTION

#### 1.1 Overview

User interact with Android apps in various ways. For example, tap on a button with their finger, navigate through a screen using their physical keyboard, enter their email address using the on-screen keyboard.

Compose has a lot of built-in support for these use cases, but in some scenarios you need to customize or extend the default behavior.

User interface components give feedback to the device user by the way they respond to user interactions. Every component has its own way of responding to interactions, which helps the user know what their interactions are doing. For example, if a user touches a button on a device's touch screen, the button is likely to change in some way, perhaps by adding a highlight color. This change lets the user know that they touched the button. If the user didn't want to do that, they'll know to drag their finger away from the button before releasing—otherwise, the button will activate.

The Compose Gestures documentation covers how Compose components handle low-level pointer event, such as pointer moves and clicks. Out of the box, Compose abstracts those low-level events into higher-level interactions—for example, a series of pointer events might add up to a button press-and-release. Understanding those higher-level abstractions can help you customize how your UI responds to the user. For example, you might want to customize how a component's appearance changes when the user interacts with it, or maybe you just want to maintain a log of those user actions. This document gives you the information you need to modify the standard UI elements, or design your own.

#### 1.2 Purpose

This application that enable people to take surveys on their smartphone or tablet, even when the device is not connected to the internet. They are used to collect feedback, design, send and analyse surveys.

Our application is used to gather information on the behavior of a specific group of people from a determined area. This kind of survey allows health care experts to understand better how a community acts towards health.

Health surveys are a necessary and helpful instrument for decision-making when crafting a health plan. Health surveys provide specific information about the epidemiological situation, health trends, life habits, and the use of health services from the patients' point of view.

This type of survey allows physicians to locate risk factors in the community around the hospital or health care centers, such as tobacco use, alcohol use, poor diet habits, and lack of physical exercise, which are common health behaviors.

Health survey software **helps you recognize weaknesses in patient experience like long check-in processes and wait times**. Once you identify these weaknesses, you can take the necessary steps.

The purpose of surveys is **to get answers to important questions**. For the most part, they're used to find out what people think about a subject and why they feel that way about it. Surveys can come in many forms, depending on what you're hoping to achieve by collecting data. address them accordingly. Keep the survey responses you get from your patients anonymous.

The Community Health Survey **helps the Health Department gain insight into factors that may impact the community's current health and how they may be improved**. The data collected will help us set goals and assist in implementing the Community Health Improvement Plan (CHIP).

The **National Family Health Survey (NFHS)** is an India-wide survey conducted by Ministry of Health and Family Welfare, Government of India, with the International Institute for Population Sciences serving as the nodal agency.

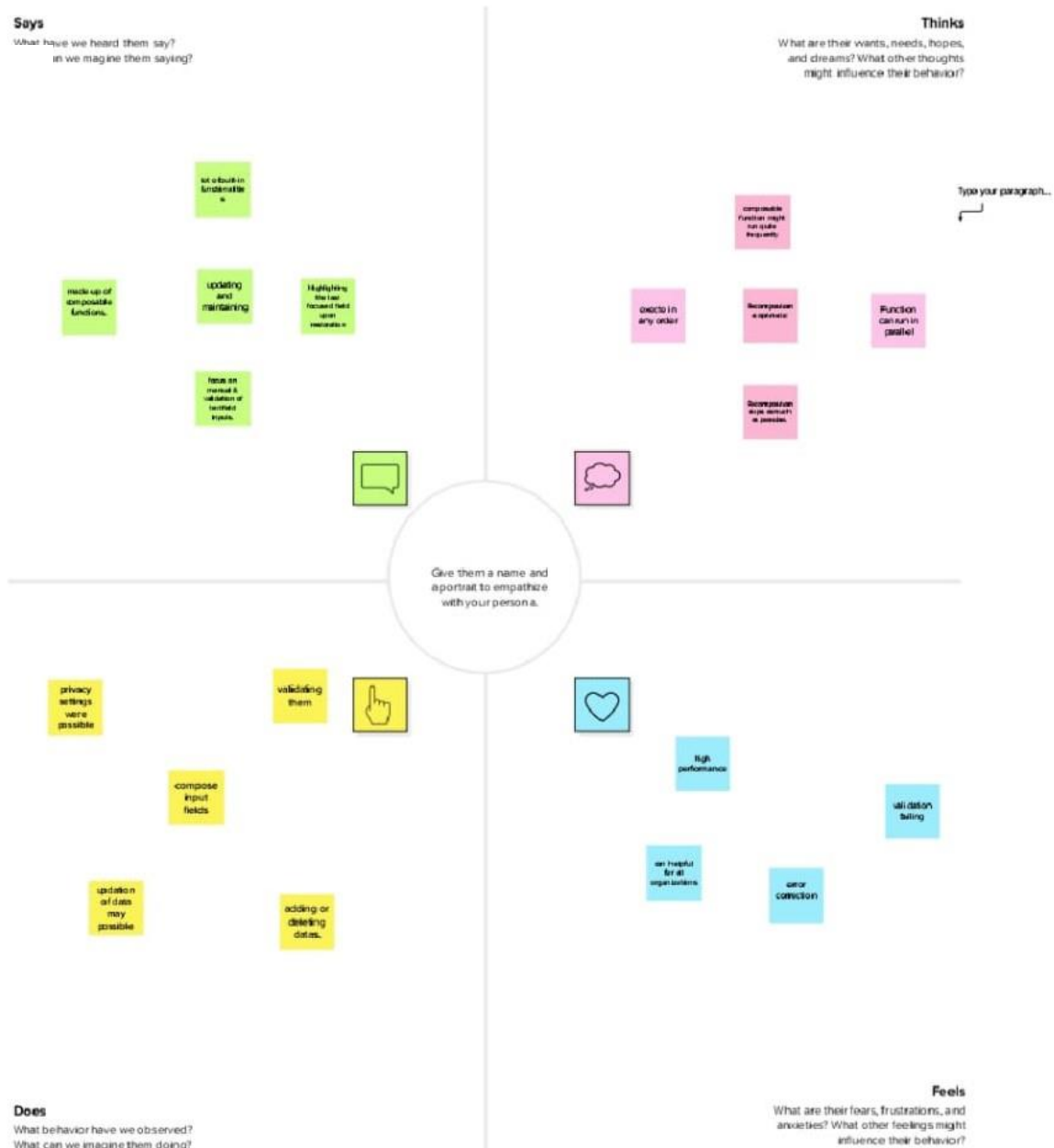
## **2 PROBLEM DEFINITION & DESIGN THINKING**

## 2.1 Empathy Map



### Build empathy

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



## 2.2 Ideation & Brainstorming Map



## Brainstorm & idea prioritization

Use this template in your own brainstorming sessions so your team can unleash their imagination and start shaping concepts even if you're not sitting in the same room.

- 0 minutes to prepare
- 10 minutes to collaborate
- 20 minutes to prioritize

Share template feedback

04

### Before you collaborate

A little bit of preparation goes a long way with this session. Here's what you need to do to get going.

0 minutes

#### 1. Brain gathering

Get everyone on the same page with a quick brainstorming session. Share your ideas and get feedback.

#### 2. Set the goal

Then, discuss the problem you're focusing on solving in the brainstorming session.

#### 3. Learn how to use the facilitation tools

Learn how to use the facilitation tools to make your session more effective and productive.

Open article

01

### Define your problem statement

What problem are you trying to solve? Frame your problem as a how might we statement. This will be the focus of your brainstorm.

0 minutes

How might we (your problem statement)?

02

### Key rules of brainstorming

To ensure a successful and productive session.

- Stay on topic.
- Define judgment.
- Go for volume.
- Encourage wild ideas.
- Let others build.
- One person, one idea.

03

### Brainstorm

Write down any ideas that come to mind that address your problem statement.

0 minutes

Tip: You can use this template to brainstorm ideas for your problem statement. Write down any ideas that come to mind that address your problem statement.

Person 1/2/3/4/5/6/7/8	Person 1	Person 2	Person 3	Person 4	Person 5	Person 6	Person 7	Person 8
Person 1								
Person 2								
Person 3								
Person 4								
Person 5								
Person 6								
Person 7								
Person 8								

3

### Group ideas

Take turns sharing your ideas while clustering similar or related notes as you go. Once all sticky notes have been grouped, give each cluster a sentence like "based on a cluster is bigger than its sticky notes, by and large you and team it up into smaller sub-groups."

20 minutes

**TIP**  
Use color-coded sticky notes to cluster ideas by topic, by team, by function, and by priority. This will help you and your team to group ideas.

Shared preferences  
Internal and external storage  
Focus on validation of textfield

4

### Prioritize

Your team should all be on the same page about what's important moving forward. Place your ideas on this grid to determine which ideas are important and which are feasible.

20 minutes

**Importance**  
How much do you think this idea is important? (1-5)

**Feasibility**  
How much do you think this idea is feasible? (1-5)

update and maintain were possible

A way of informing the user that their questions were successfully registered.

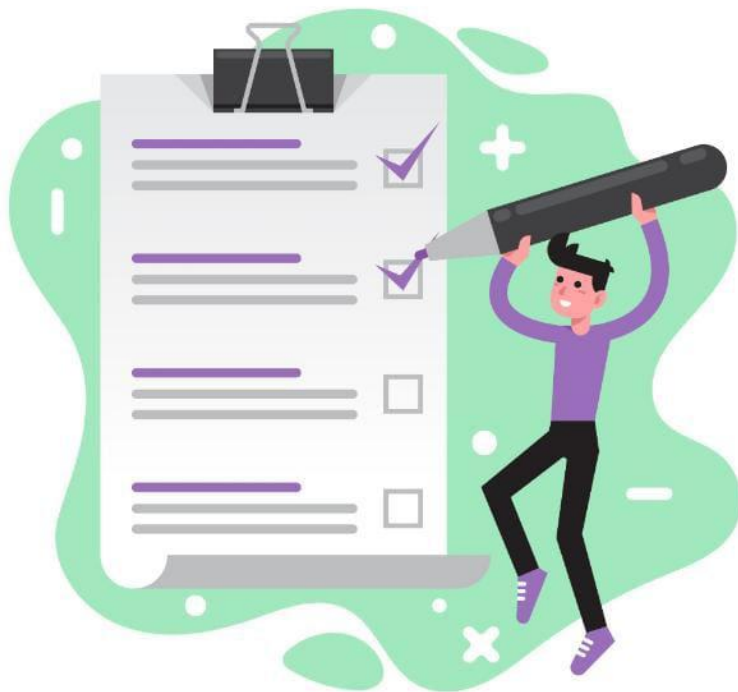
Specify a shared external storage in your app

**TIP**  
Rearrange sticky notes on the grid to clarify focus. The team can agree on the most important and feasible ideas.

## 3 RESULT

**Admin Module:**

**Register Page:**



## Register

Username

Abisha

Email

abi

Password

admin

User registered successfully

Register

## Login Page:



*Login*

Username  
Abisha

Password  
admin

Successfully log in

Login

[Register](#)

[Forget password?](#)

## Admin Page:

### Survey Details

Name: ghg  
Age: 18  
Mobile\_Number: 4567964323  
Gender: Female  
Diabetics: Diabetic

Name: ghg  
Age: 18  
Mobile\_Number: 4567964323  
Gender: Female  
Diabetics: Diabetic

Name: ghg  
Age: 18  
Mobile\_Number: 4567964323  
Gender: Female  
Diabetics: Diabetic

Name: dff  
Age: 34  
Mobile\_Number: 3467975765  
Gender: Female  
Diabetics: Diabetic

Name: ggg  
Age: ggg  
Mobile\_Number: 7398494948  
Gender: Male  
Diabetics: Diabetic



## User Module:

### Login Page:



*Login*

Username  
andrea

Password  
1234

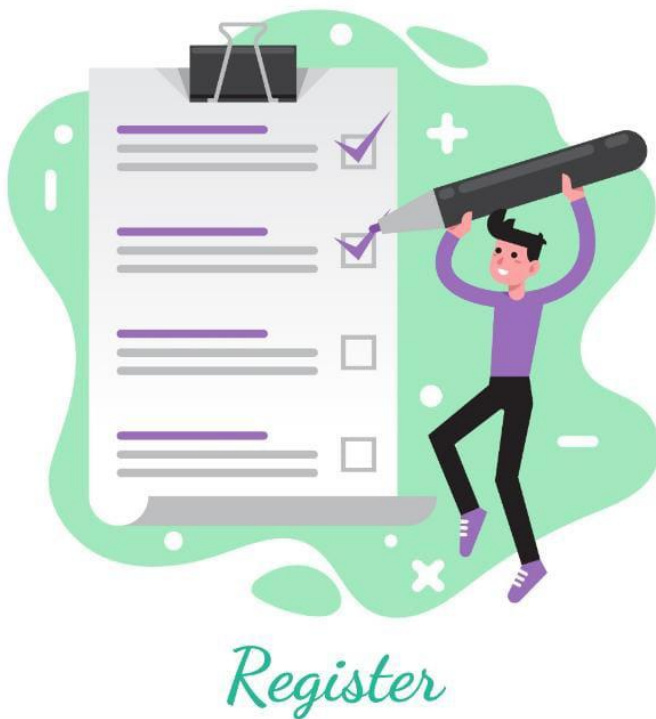
Invalid username or password

Login

[Register](#)

[Forget password?](#)

## Register Page:



*Register*

Username  
andrea

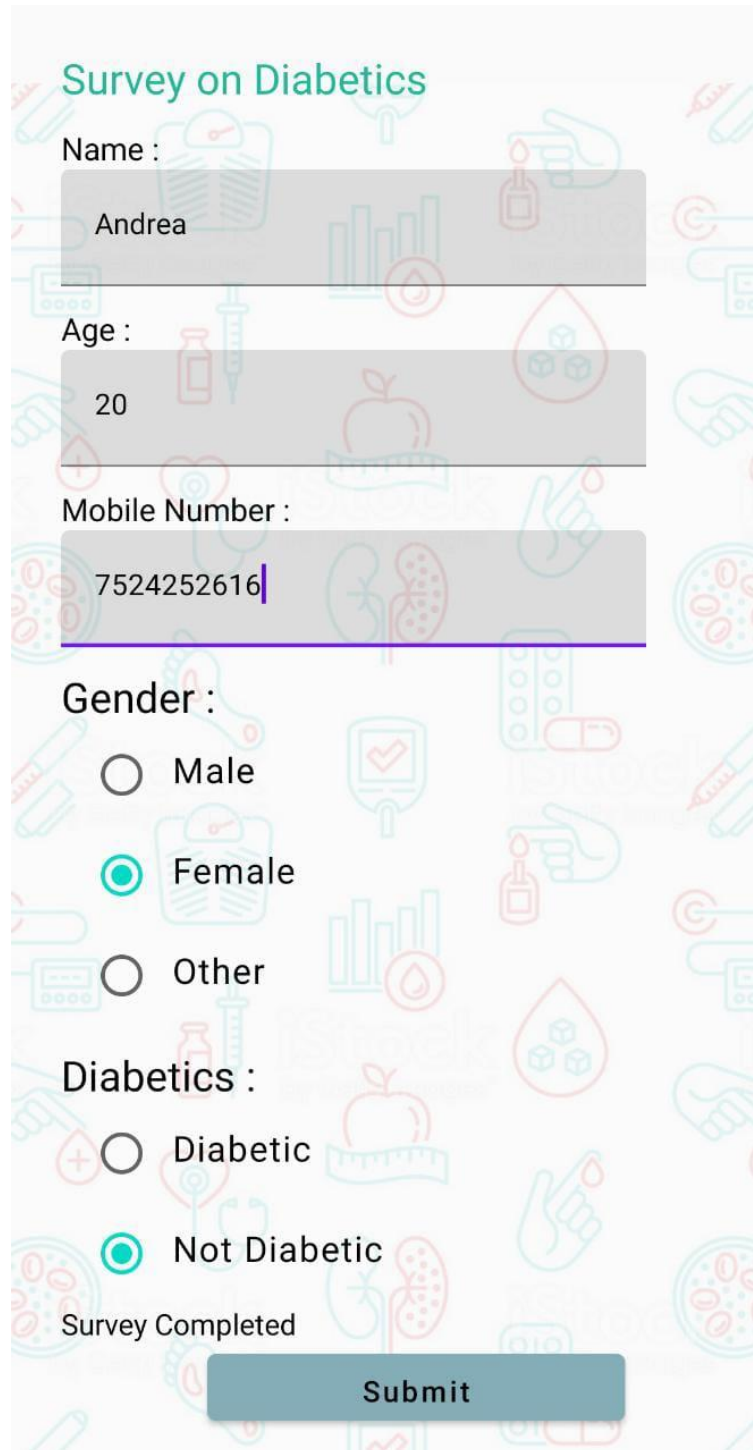
Email  
riya@gmail.com

Password  
1234

User registered successfully

Register

## Main Page:



**Survey on Diabetics**

Name :  
Andrea

Age :  
20

Mobile Number :  
7524252616

Gender :  
☐ Male  
☒ Female  
☐ Other

Diabetics :  
☐ Diabetic  
☒ Not Diabetic

Survey Completed

Submit

## 4 ADVANTAGES & DISADVANTAGES

### Advantages:

Often the only way to gather important qualitative (text) feedback from your market is by using a questionnaire. Using open ended questions you can gain deeper insights into what your customers think about each aspect of the business.

The problem is the technical skills and resources required to design, build and analyses these tests is formidable. Not only do you need to expend resources creating the different version of that you want to test, advanced statistical tools and analysis are needed to interpret the results accurately.

**It helps to know which areas present severe issues and make the necessary adjustments to correct them.** Hospitals can collect information on medical care times and improve metrics, for example, through health surveys.

A health survey will let a hospital or health care center know what its patients think about the service they provide. With our platform, they can receive patients' comments in real-time. It helps to know which areas present severe issues and make the necessary adjustments to correct them.

Hospitals can collect information on medical care times and improve metrics, for example, through health surveys. They will be able to know if a diagnosis is made correctly or if the medications that patients obtain are adequate for their treatment. By collecting patient feedback through health surveys, the hospital can avoid potential lawsuits for loss of life due to poor care.

### Disadvantages:

1. Respondents may not feel encouraged to provide accurate, honest answers.
2. Respondents may not feel comfortable providing answers that present themselves in a unfavorable manner.
3. Respondents may not be fully aware of their reasons for any given answer because of lack of memory on the subject, or even boredom.

The survey that was used by the researcher from the very beginning, as well as the method of administering it, cannot be changed all throughout the process of data gathering. Although this inflexibility can be viewed as a weakness of the survey method, this can also be a strength considering the fact that preciseness and fairness can both be exercised in the study.

## 5 APPLICATIONS

In-app surveys are **an amazing way to connect with your customers, and therefore a crucial tool for collecting user feedback.** These surveys are deployed into apps, enabling you to gather relevant feedback on how your customers view your app as a whole, or how they view the individual features.

Unlike inefficient traditional telephone access to doctors and healthcare organizations, mobile health apps **make it easy and secure for patients to send messages, schedule appointments & connect to care providers 24/7 for telemedicine visits.**

Mass availability and use of health apps raises the question as to how they might be integrated into healthcare systems towards improving prevention and therapy. This study has researched prevailing opinion on health apps amongst primary care physicians, potential application areas physicians have seen in their experience with these apps up to now, and situations suitable for using apps in patient care.

general practitioners are still reluctant to bring up or recommend health apps in their consultations. Many physicians do not feel capable of giving expert advice to patients on the apps available. Many general practitioners are aware of the potential that health apps may have in improving prevention and treatment. However, there are reservations and uncertainties regarding clarity, transparency, and privacy issues in these apps. More focus should be placed on these concerns to ensure ideal conditions for integrating health apps into primary care.

### **Summary:**

The apps may be of particular benefit in primary care considering the heavy pressure on time and resources as well as the wide range of symptoms, diseases and patient groups involved. The apps would seem ideal as support tools for physicians to use on specific patients in primary care, thus promoting health in the long term. Studies have also emphasised the benefits of apps as an aid in optimising differential diagnosis as well as patient and treatment compliance.

Only a few studies have examined the application potential for health apps specifically in primary care. Surveys amongst mixed specialist groups have shown that 25–45% of physicians occasionally discuss health apps with their patients. Apart from that, 36–42% see reinforced patient empowerment as a major benefit in these apps, followed by improved patient training. Surveys conducted in the US and Australia indicate relative reluctance amongst medical professionals in raising the topic of digital or mobile health monitoring solutions with their patients. Physicians may see opportunities for health apps in reinforcing patient motivation and empowerment, but express uncertainty as to whether the apps are ready for prime time in view of reliability, data privacy, technical maturity and applicability, and their everyday implementation in patient care systems.

## **6 CONCLUSION**

The present study reveals that many students were aware of the apps and using it regularly to track their physical activity and calorie intake. It also motivates them to maintain a healthy lifestyle. Users are taking these Apps quite seriously and update the credentials regularly so that output can be appropriately tracked. Accordingly, they are not only finding the tips and suggestions useful but also using these Apps in different ways like tracking calories, weight and monitoring sleep quality. Also, they are reviewing the data on monthly/ weekly basis to change the fitness activities or food intake.

Surveys are a commonly used tool in healthcare epidemiology and antimicrobial stewardship research. Surveys allow selection of a relatively large sample of people from a predetermined population, followed by collection of data from those individuals, and may be exploratory, descriptive, or explanatory.

This paper concentrates more on the security awareness and public personal information concern. Furthermore, this study only concentrates on the educated and professional samples, which make up a

population that is mostly influential in India's situations. They have the potential to give input to the government for the improvement of system even though their opinion may not be representative of the whole India's population. The present study was aimed to measure the level of information security awareness on using online system which shows that most of the respondents have high awareness result in from frequent use of the computer and internet in work and daily matters. In term of gender, male respondents have similar awareness level is compared to the male respondents. This is shown through their focus which was more and the easier security measures such as password changing, creating unique password and checking the status bar.

In addition, India's are still comfortable in sharing their personal information with their groups, namely people are hospital staffs who are closed to them or knowledgeable about them, relatives and family.

## 7 FUTURE SCOPE

The world is moving towards digitalization. A healthcare feature based on mobile apps is the ultimate solution for the future. People are leaning toward these technology-driven apps due to different reasons. Several healthcare apps are emerging with specific features and advanced technology.

Almost all the processes worldwide lead to an app-based solution that is implied in the healthcare sector. The world is asking for more healthcare mobile apps with innovative features and technology solutions over time.

The world is constantly growing, and each segment's growth is entirely towards digitalization and technology-driven features. This factor will help you to be a knowledgeable healthcare mobile app developer. While considering your Scope in this industry with futuristic projection., there are many other factors.

The world is getting to digitalized with each phase and getting too extensive access towards smartphones and devices. And this will give the mobile health app developers a considerable scope in the future. And this will give the mobile health app developers a considerable scope in the future.

People need an advanced technology stack while accessing their health-related issues for consultation. With time, the technologies need to be upgraded, and along with that, the acceptance of the apps will also increase. So, it would be best if you created apps with advanced backend and frontend accessibility with an elevated creativity ratio. Integrating new features which meet the user requirement will upgrade the approach for healthcare applications.

Accuracy is the most imperative aspect of a medical condition. The Scope of errors can be harmful. With the help of the [healthcare apps developed](#) by healthcare app developers, the health examination in a real-time situation is possible. A doctor can record a patient's real-time condition and proper process treatment.

That patient can be monitored ideally in specific cases such as allergies, chronic conditions, etc. So, people are looking forward to innovative healthcare apps, providing a sustainable scope for developers. According to a recent survey, 90 percent of physicians use healthcare apps with e-records, schedules, etc.

## 8 APPENDIX

### A. Source Code

#### 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 diabetesOptions = 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(background-color = 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_NUMBER)),
            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_NUMBER)),
            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_NUMBER)),
                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()
    }

    @SuppressWarnings("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
    }

    @SuppressWarnings("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
    }

    @SuppressWarnings("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
}
}

```

## Build.gradle

```

plugins {
    id 'com.android.application'
    id 'org.jetbrains.kotlin.android'
}

android {
    namespace 'com.example.surveyapplication'
    compileSdk 33

    defaultConfig {
        applicationId "com.example.surveyapplication"
        minSdk 21
        targetSdk 33
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
        vectorDrawables {
            useSupportLibrary true
        }
    }

    buildTypes {

```

```

        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-
rules.pro'
        }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
    kotlinOptions {
        jvmTarget = '1.8'
    }
    buildFeatures {
        compose true
    }
    composeOptions {
        kotlinCompilerExtensionVersion '1.2.0'
    }
    packagingOptions {
        resources {
            excludes += '/META-INF/{AL2.0,LGPL2.1}'
        }
    }
}

```

```

dependencies {
    implementation 'androidx.core:core-ktx:1.7.0'
    implementation 'androidx.lifecycle:lifecycle-runtime-ktx:2.3.1'
    implementation 'androidx.activity:activity-compose:1.3.1'
    implementation "androidx.compose.ui:ui:$compose_ui_version"
    implementation "androidx.compose.ui:ui-tooling-preview:$compose_ui_version"
    implementation 'androidx.compose.material:material:1.2.0'
    implementation 'androidx.room:room-common:2.5.0'
    implementation 'androidx.room:room-ktx:2.5.0'
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.5'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.5.1'
    androidTestImplementation "androidx.compose.ui:ui-test-junit4:$compose_ui_version"
    debugImplementation "androidx.compose.ui:ui-tooling:$compose_ui_version"
    debugImplementation "androidx.compose.ui:ui-test-manifest:$compose_ui_version"
    implementation 'androidx.room:room-common:2.5.0'
    implementation 'androidx.room:room-ktx:2.5.0'
}

```