Smart Internz

PROJECT REPORT TEMPLATE

1.INTRODUCTION

1.1 Overview

A Brief Description about the project

Android software development is the process by which applications are created for devices running the Android operating system. Google states that[3] "Android apps can be written using Kotlin, Java, and C++ languages" using the Android software development kit (SDK), while using other languages is also possible. All non-Java virtual machine (JVM) languages, such as Go, JavaScript, C, C++ or assembly, need the help of JVM language code, that may be supplied by tools, likely with restricted API support. Some programming languages and tools allow cross-platform app support (i.e. for both Android and iOS). Third party tools, development environments, and language support have also continued to evolve and expand since the initial SDK was released in 2008. The official Android app distribution mechanism to end users is Google Play; it also allows staged gradual app release, as well as distribution of pre-release app versions to testers

1.2 Purpose

The Use of this project

- A Project that demonstrates the uses of android Jetpack composed to build a UI for Survey App. Survey App project Built Using in the Android Jetpack Compose UI toolkit.The App Allows the User to

Answer a Series of Questions.It Showcases some of the key Features of the Compose UI Toolkit,Data Management and User

- You 'll be able to Work on Android Studio and build an app.

2.PROBLEM DEFINITION & DESIGN THINKING

2.1 Empathy Map

Paste the Empathy Map Screenshots

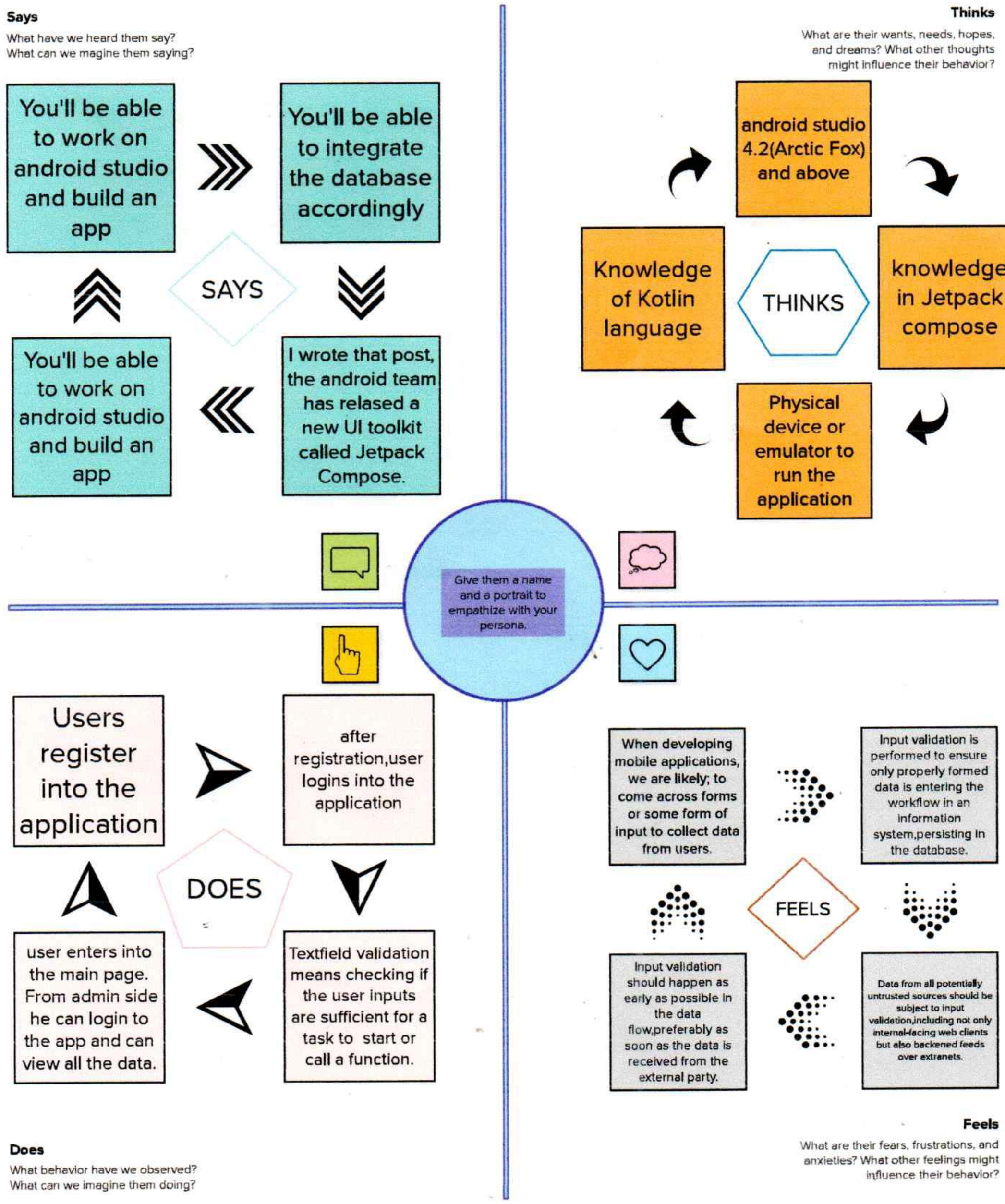
Empathy Map

- An Empathy map is a simple ,easy to digest visual that captures knowledgement about a users behaviours and attitudes

- It is useful tool to helps teams better Understand their users
- creating an effective solution requires understanding the true problem and the person who is experiencing it.
- The Exercise of creating the map helps participants consider things from the users persceptive along with his / her goals and challenges

Build empathy

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



Says

What have we heard them say?
What can we magine them saying?

You'll be able to work on android studio and build an app

You'll be able to integrate the database accordingly

I wrote that post, the android team has relased a new UI toolkit called Jetpack Compose.

You'll be able to work on android studio and build an app

Thinks

What are their wants, needs, hopes, and dreams? What other thoughts might influence their behavior?

android studio 4.2(Arctic Fox) and above

knowledge in Jetpack compose

Physical device or emulator to run the application

Knowledge of Kotlin language

Give them a name and a portrait to empathize with your persona.

Users register into the application

after registration,user logins into the application

Textfield validation means checking if the user inputs are sufficient for a task to start or call a function.

user enters into the main page. From admin side he can login to the app and can view all the data.

When developing mobile applications, we are likely; to come across forms or some form of input to collect data from users.

Input validation is performed to ensure only properly formed data is entering the workflow in an information system,persisting in the database.

Data from all potentially untrusted sources should be subject to input validation,including not only internal-facing web clients but also backened feeds over extranets.

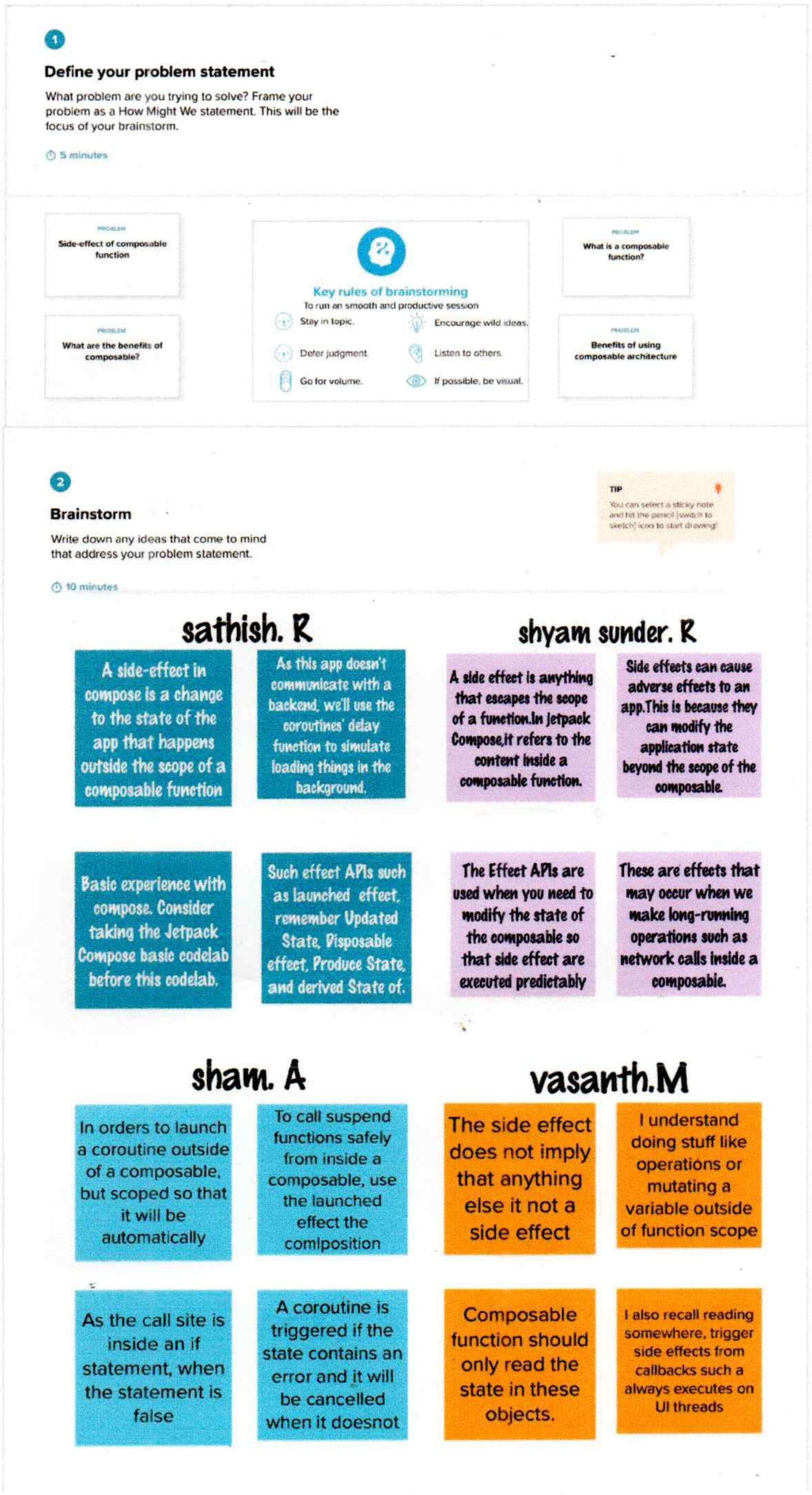
Input validation should happen as early as possible in the data flow,preferably as soon as the data is received from the external party.

Feels

What are their fears, frustrations, and anxieties? What other feelings might influence their behavior?

Does

What behavior have we observed?
What can we imagine them doing?



1

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.

5 minutes

PROBLEM
Side-effect of composable function

PROBLEM
What are the benefits of composable?

Key rules of brainstorming
To run an smooth and productive session

- Stay in topic.
- Encourage wild ideas.
- Defer judgment.
- Listen to others.
- Go for volume.
- If possible, be visual.

PROBLEM
What is a composable function?

PROBLEM
Benefits of using composable architecture

2

Brainstorm

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

10 minutes

TIP
You can select a sticky note and hit the pencil [switch to sketch] icon to start drawing!

sathish. R

A side-effect in compose is a change to the state of the app that happens outside the scope of a composable function

As this app doesn't communicate with a backend, we'll use the coroutines' delay function to simulate loading things in the background.

Basic experience with compose. Consider taking the Jetpack Compose basic codelab before this codelab.

Such effect APIs such as launched effect, remember Updated State, Disposable effect, Produce State, and derived State of.

shyam sunder. R

A side effect is anything that escapes the scope of a function.In Jetpack Compose,it refers to the content inside a composable function.

Side effects can cause adverse effects to an app.This is because they can modify the application state beyond the scope of the composable.

The Effect APIs are used when you need to modify the state of the composable so that side effect are executed predictably

These are effects that may occur when we make long-running operations such as network calls inside a composable.

sham. A

In orders to launch a coroutine outside of a composable, but scoped so that it will be automatically

To call suspend functions safely from inside a composable, use the launched effect the comlposition

As the call site is inside an if statement, when the statement is false

A coroutine is triggered if the state contains an error and it will be cancelled when it doesnot

vasanth.M

The side effect does not imply that anything else it not a side effect

I understand doing stuff like operations or mutating a variable outside of function scope

Composable function should only read the state in these objects.

I also recall reading somewhere, trigger side effects from callbacks such a always executes on UI threads

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 label. If a cluster is bigger than six sticky notes, try and see if you and break it up into smaller sub-groups.

20 minutes

Benefits of Using Composable Architecture :

Improved Flexibility. By focusing on individual components of composable architecture, developers can create more flexible systems and easily adapt them to new requirements and technologies.....
Better Scalability......
Increased Efficiency.....
Organizational Benefits.......

What are the benefits of composable?

With composable infrastructure, it's possible to allocate the exact compute, storage, or memoryresources needed in any given situation. This approach allows for the composing and recomposing of infrastructure to meet the plrecise demands of various workloads

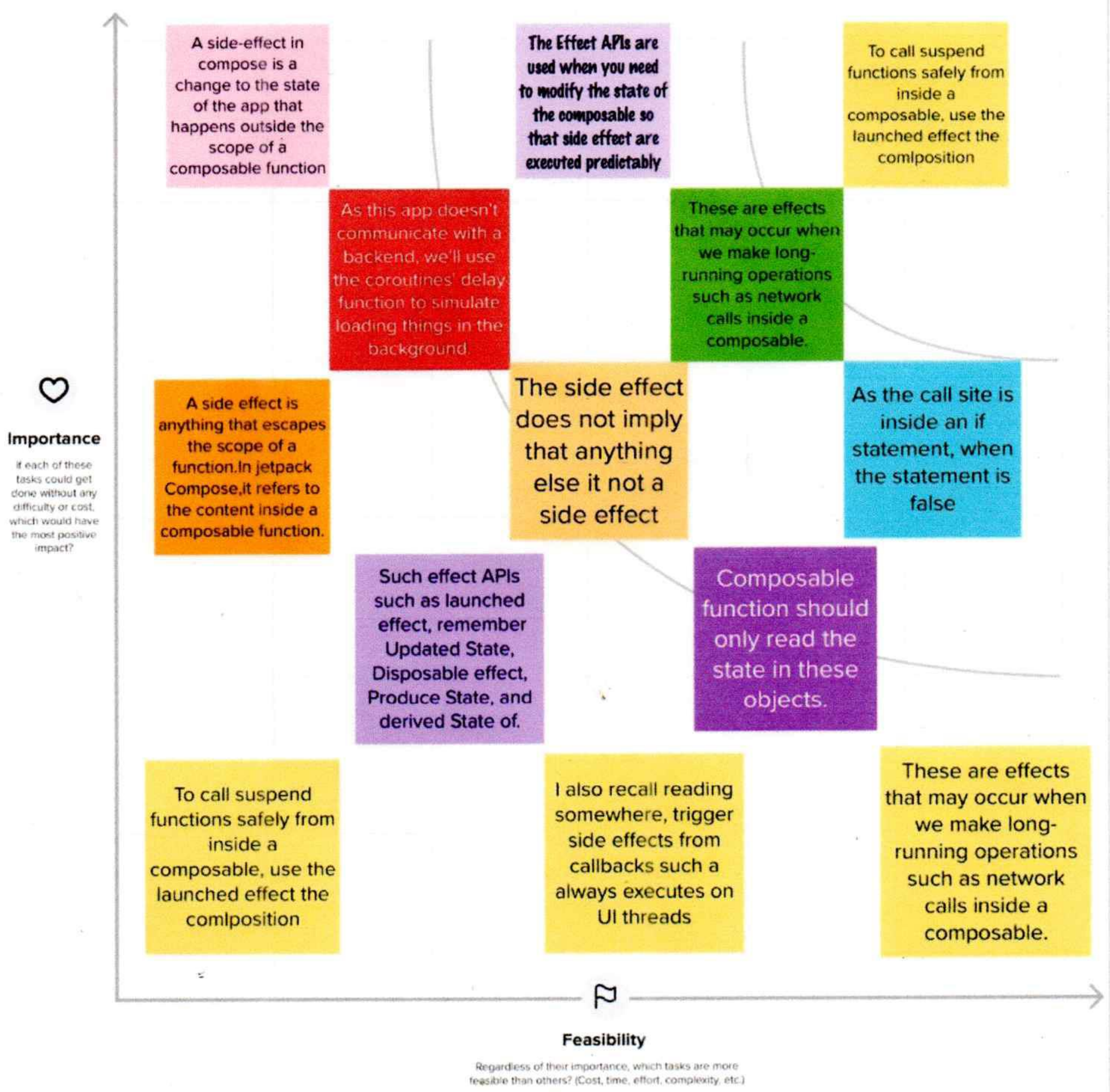
What is a composable function?

Composable fuctions can accept parameters, which allow the app logic to describe the UI. In this case, our widget accepts a String so it can greet the user by name. The function desplays text in the UI. It does so by calling the Text() composable function, which actually creates the text UI element.

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



A side-effect in compose is a change to the state of the app that happens outside the scope of a composable function

The Effect APIs are used when you need to modify the state of the composable so that side effect are executed predictably

To call suspend functions safely from inside a composable, use the launched effect the comlposition

As this app doesn't communicate with a backend, we'll use the coroutines' delay function to simulate loading things in the background.

These are effects that may occur when we make long-running operations such as network calls inside a composable.

A side effect is anything that escapes the scope of a function.In jetpack Compose,it refers to the content inside a composable function.

The side effect does not imply that anything else it not a side effect

As the call site is inside an if statement, when the statement is false

Such effect APIs such as launched effect, remember Updated State, Disposable effect, Produce State, and derived State of.

Composable function should only read the state in these objects.

To call suspend functions safely from inside a composable, use the launched effect the comlposition

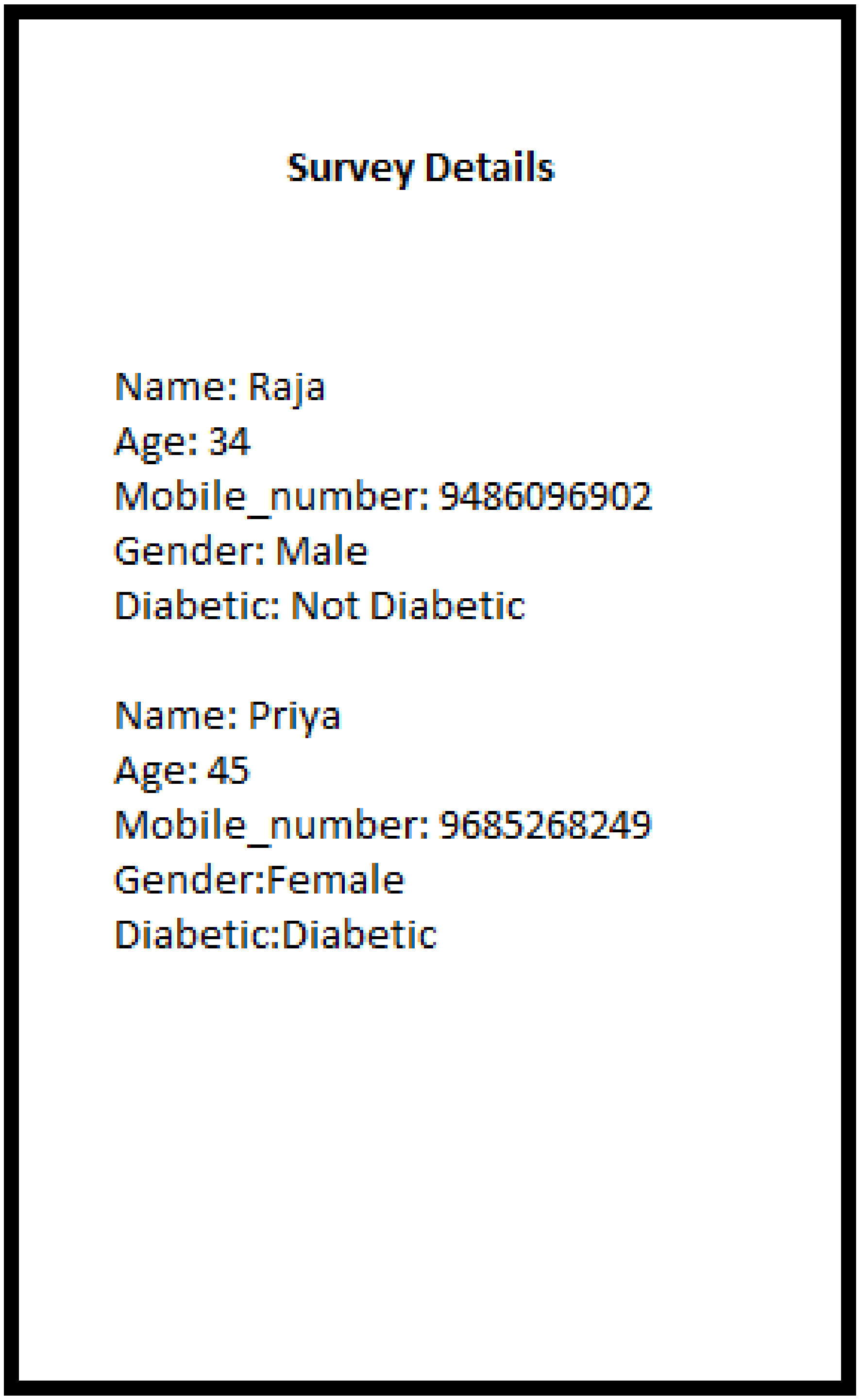
I also recall reading somewhere, trigger side effects from callbacks such a always executes on UI threads

These are effects that may occur when we make long-running operations such as network calls inside a composable.

Importance
If each of these tasks could get done without any difficulty or cost, which would have the most positive impact?
Feasibility
Regardless of their importance, which tasks are more feasible than others? (Cost, time, effort, complexity, etc.)

RESULT:

SAMPLE OUTPUT:



```
                Survey Details


Name: Raja
Age: 34
Mobile_number: 9486096902
Gender: Male
Diabetic: Not Diabetic

Name: Priya
Age: 45
Mobile_number: 9685268249
Gender:Female
Diabetic:Diabetic
```

4.ADVANTAGES & DISADVANTAGES

❖ ADVANTAGES

- ***focusManager*** is used to clear current focus and to move it in certain direction. In our case it's *down*.
- ***keyboardController*** is used to hide/show keyboard.
- ***creditCardNumberFocusRequester*** & ***nameFocusRequested*** are *FocusRequesters*. They allow us to request focus for composables on demand(eg. from events
  - ***Modifier.focusRequester.onFocusChanged*** — We've added focus requesters and *onFocusChanged* listener to our composable modifiers.
  - ***fieldValue*** — is a class holding information about the editing state.The input service updates text selection, cursor, text and text composition. This class represents those values and allows to observe changes to those values in the text editing composables. We need it to place the input indicator to the end of the entered text upon requesting focus after process death/if input is not empty.

Auto-validation UX can be improved by adding debouncing to the validation events. Debounce sample is inside the repository

❖ DISADVANTAGES

1) Keyboard isn't opened upon entering the screen.

2) No *TextField* is focused upon entering the screen.

3) There is no way to tell which *TextField* was focused last, after process death occurred.

4) No *ImeAction* handling for the *name TextField.*

5) Keyboard isn't dismissed upon successful button click

5.APPLICATION

The Areas Where this Solution can be applied

This Application can be used for

- Surveying the person's having diabetics or not

6.CONCLUSION

- A Conclusion is an important part of the paper : It provides closure for the reader while reminding the reader of the contents and importance of the paper.

7.FUTURE SCOPE

Android compose is the clearly future of Android .It Requires less code and it's easier to understand and maintain.Compose allows you to build higher quality screens more quickly

8.APPENDIX

A.Source code

Attach the code for the solution built

AndroidManifest.xml

```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>
```

Color.kt

```kotlin
package com.example.surveyapplication.ui.theme

import androidx.compose.ui.graphics.Color

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

Shape.kt

```kotlin
package com.example.surveyapplication.ui.theme

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

val Shapes = Shapes(
    small = RoundedCornerShape(4.dp),
    medium = RoundedCornerShape(4.dp),
    large = RoundedCornerShape(0.dp)
)
```

Theme.kt

```kotlin
package com.example.surveyapplication.ui.theme

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

private val DarkColorPalette = darkColors(
    primary = Purple200,
    primaryVariant = Purple700,
    secondary = Teal200
)

private val LightColorPalette = lightColors(
    primary = Purple500,
    primaryVariant = Purple700,
    secondary = Teal200

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

@Composable
fun SurveyApplicationTheme(
    darkTheme: Boolean = isSystemInDarkTheme(),
    content: @Composable () -> Unit
) {
    val colors = if (darkTheme) {
        DarkColorPalette
    } else {
        LightColorPalette
    }

    MaterialTheme(
        colors = colors,
        typography = Typography,
        shapes = Shapes,
        content = content
    )
}
```

Type.kt

```
package com.example.surveyapplication.ui.theme

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

// Set of Material typography styles to start with
val Typography = Typography(
    body1 = TextStyle(
        fontFamily = FontFamily.Default,
        fontWeight = FontWeight.Normal,
        fontSize = 16.sp
    )
    /* Other default text styles to override
    button = TextStyle(
        fontFamily = FontFamily.Default,
        fontWeight = FontWeight.W500,
        fontSize = 14.sp
    ),
    caption = TextStyle(
        fontFamily = FontFamily.Default,
        fontWeight = FontWeight.Normal,
        fontSize = 12.sp
    )
    */
)
```

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_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()
    }

    @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
    }

}
```

ExampleInstrumentedTest.kt

```
package com.example.surveyapplication

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

import org.junit.Test
import org.junit.runner.RunWith

import org.junit.Assert.*

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

ExampleUnitTest.kt

```
package com.example.surveyapplication

import org.junit.Test

import org.junit.Assert.*

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