Compose input: A demonstration of text input and validation with android compose

APROJECT

Submitted by

JENIFER SELVA RANI J (20201231506216)

MAHARASI G (20201231506222)

SAROJA S (20201231506232)

VENNILA G (20201231506247)

SATHYA G (20201231506234)

MENTOR

Dr.T.ARULRAJ M.Sc.,M.Phil.,Ph.D

inpartialfulfillmentoftherequirementsfortheawardofdegreeof

BACHELOR OF SCIENCE(Computer Science)

Department of Computer Science

SRI PARAMAKALYANI COLLEGE

ALWARKURICHI– 627412

APRIL-2023

TABLEOFCONTENTS

ChapterNo.	Title	PageNo.
I	**INTRODUCTION**	1
	Overview Purpose	
II	**PROBLEM DEFINITION &DESIGN THINKING**	7
	EmpathyMap Ideation&BrainstormingMap	
III	**RESULT**	12
IV	**ADVANTAGES & DISADVANTAGES**	15
V	**APPLICATIONS**	18
VI	**CONCLUSION**	20
VII	**FUTURESCOPE**	21
VIII	**APPENDIX**	23
	SourceCode	

CHAPTER-I

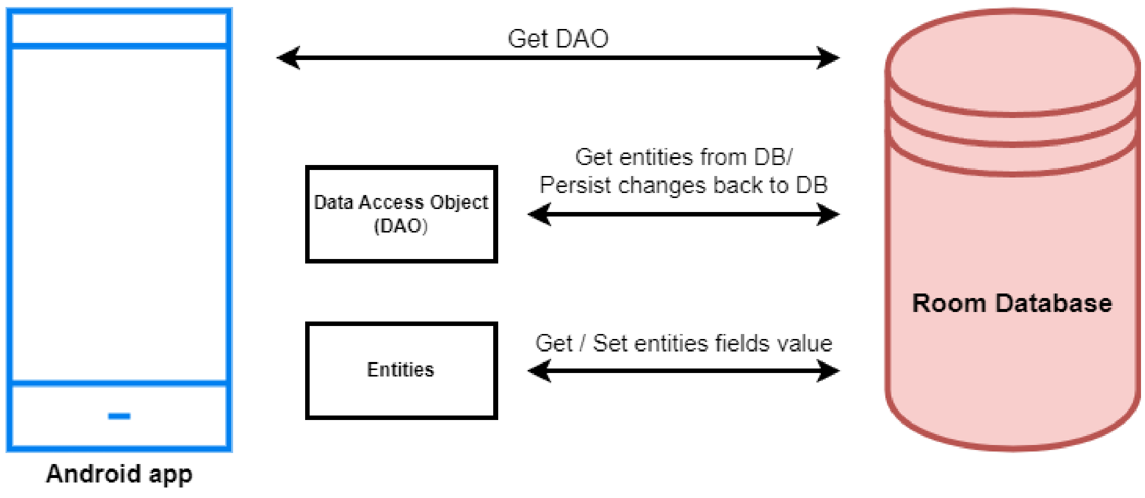
INTRODUCTON

Overview

In today’s world, Smart phones have changed our lives and have become an indispensable part of our lives because of its specialty to simplify our routine work and thereby saving our time. A Smartphone with an Android OS offers excellent functionality to the users offering a distinct experience. Android is a Linux based operating system and it was bought by Google in 2007.There are tons of application available and one of the prime reason for this vast number is android being an open source. On the other hand, android based device like mobile, tab are very user friendly. A survey has done by “Light Castle Partners” research wing which indicates that though other operating system mobile users exist but the majority users are goes with android operating system.In this context, Project application is developed based on android platform. The name of application is define as ‘Survey Application’.

The app is a sample project that demonstrates how to use the Android Compose UI toolkit to build a survey app. The app allows the user answer a series of questions. It showcases some of the key features of the Compose UI toolkit, data management, and user interactions.

Architecture


Get DAO
Data Access Object (DAO)
Get entities from DB/ Persist changes back to DB
Entities
Get / Set entities fields value
Room Database
Android app

System Requirements:

1. Smartphone with Android OS version 4.4 (Kitkat) or higher

2. Minimum 512 MB of RAM

3. A processor with speeds above 1.2 GHz (any make)

4. 16 MB of storage for the app and extra for the data stored, the size of the app increases as the number of entries are increased

5. Android API version 19

6. Permission to install applications over USB and installation from unknown sources from 'Developer Options'

Applications and Technology Used:

Android Application Development is possible with a couple of software and development kits to support the software and execution, they are as follows,

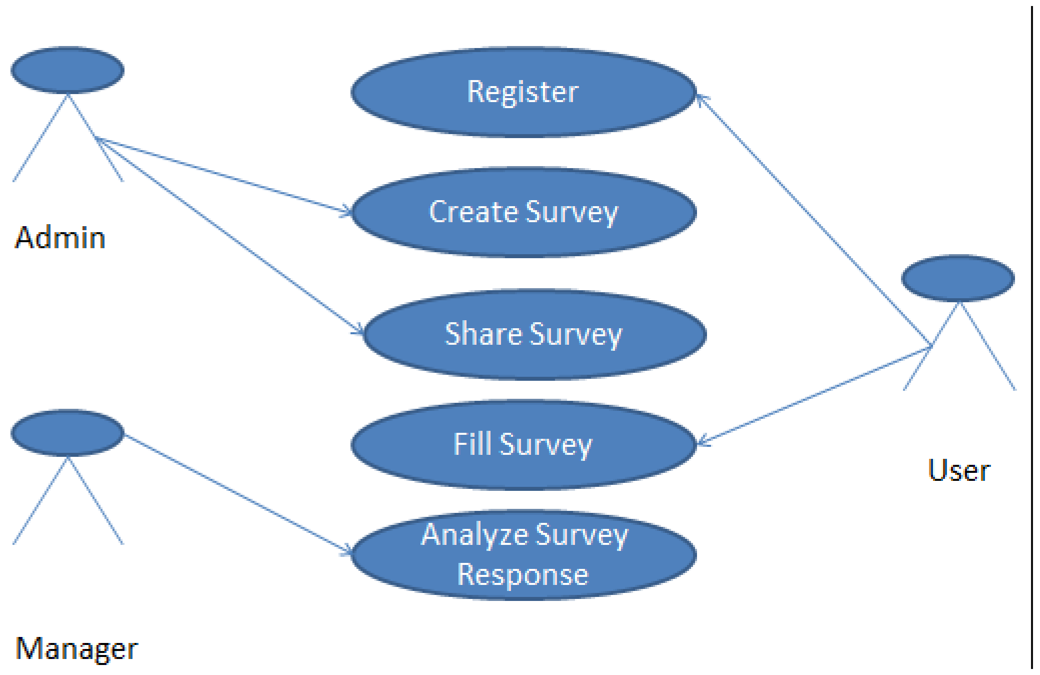
1. Android Studio: Android studio is the official Integrated Development Environment (IDE) for designing, coding, debugging and executing applications for Google's Smartphone operating system, Android. It has all keywords inbuilt for ease in back end programming and also design statements, for ease in designing as well.

2. Java Development Kit (JDK): Since, Android applications require Java programming for its backend programming; it needs a JAVA environment to support its functions, executions and syntax.

3. Xml: The front end design of the application involves xml statements for the Relative layouts, Radio buttons, Radio Group, buttons, text boxes and text views.

Purpose

This app allow the user to answer a series of question and view their result. The question stores in data model and each question is updated after the user answers all the question. The user has the option to save the result by entering their name and E-Mail address. Throughout the app, the user can interact with the User Interface by selecting answer, clicking buttons and text.



Mobile communication is a world wide used communication method and has a large ratio over all alternative communication methods. At first stages of this technology, mobile phone models that have only calling function were used.

But today's technology gives us the opportunity to use mobile phones with many functions like internet access, calling functions, video players and etc. One of the properties of mobile phones is the operating system. By the operating systems of these phones, we are able to run many software that have been designed and coded for different purposes and these phones are called smart mobile phones. Mostly used operating systems on smart mobile phones are Android, IOS and Windows Phone. Android is the most preferred operating system in this sector. This document provides a base to all the functionalities which should be carried out by the application, how that works the outputs available to the end user.

Main Features:

- Beautiful UI
- Multilingual support
- Easy to Customize/Re-skin
- Support all Screen size
- Set limited Questions

Survey for Students

A new survey system aimed to use on education to run on smart mobile phones with Android operating system is presented in this paper. It provides the administrator and the attendees of the survey the ability to answer the questions. The presented application has the potential of double usage perspective. One of them is running as a conventional survey that the questions and answers can be used as a way to learn the opinions of the attendees about the questions. In this usage method, there is not a correct or false answer. Only the user opinions are gathered and classified according to the answers. The other usage type of the system is an online exam platform for a teacher and students. It allows both the teacher and the students meet on the same application. The students or attendees are wanted to answer the questions on the survey which were inserted to the database before about a specified subject, by using their smart phones. Each student/attendee answers the questions one by one. No extra equipment for answering is needed. At the end of the answering stage, the teacher can analyze the answers on different views. For example how many students/attendees have chosen each answer for each question, the correct and wrong answers ratio to evaluate the results of the class in general, the answers of a specified student to evaluate the performance of that student. This survey system can also be used in crowded attendee group like civil meetings on different aims like discussion groups, members of an enterprise company or any other social meeting. This smart phone based survey system is faster, safer, more easy to use, more useful and more technological than classical survey systems on paper. As the answers of the students and answering times are inserted into a database with the user name, it also provides the teacher to analyze the results in many ways that can be wanted.

CHAPTER–II

Problem Definitions & Design Thinking

EmpathyMap

Empathy maps are an efficient tool used by designers to not only understand user behaviour , but also visually communicate those findings to colleagues, uniting the team under one shared understanding of the user. Originally invented by Dave Gray at Xplane , the empathy map was made in an attempt to limit miscommunication and misunderstanding about target audiences, including customers and users.

Essentially, an empathy map is a square divided into four quadrants with the user or client in the middle. Each of the four quadrants comprises a category that helps us delve into the mind of the user. The four empathy map quadrants look at what the user says, thinks, feels, and does.

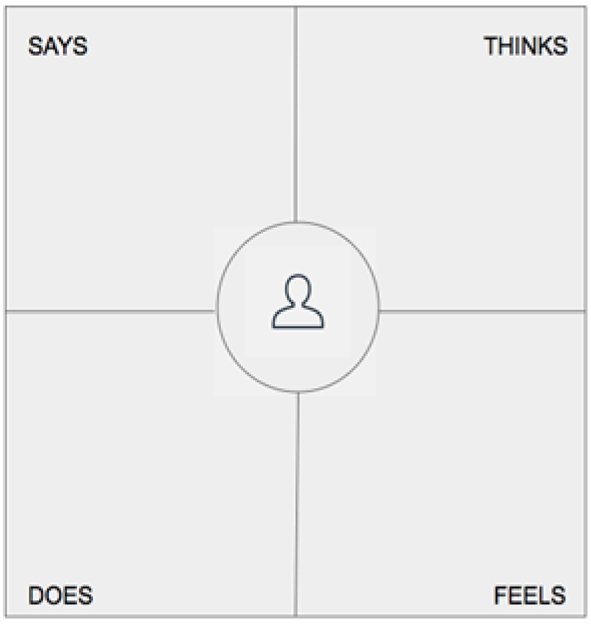
With the user at the center and the categories in each of the four surrounding quadrants, an empathy map arranges all of your research about the user into an easy-to-read visual. Let's take a closer look at the four quadrants and how they can capture an accurate image of the user.

Says

This section contains direct quotes from the user that have been gathered from the research phase or previous data. It might feature statements like "I need something fast," or "I'm not sure where to go from here."

Thinks

While this quadrant may have similar content to the "Says" section, it is more focused on what a user is thinking and doesn't choose to say out loud. Use your qualitative research to ask what matters to the user and what is on their mind. Looking at why they might be hesitant to share their thoughts out loud can reveal even further insight into the user and how they relate to the product or experience at hand. Example: "This is boring," or "Am I doing this right?"



Feels

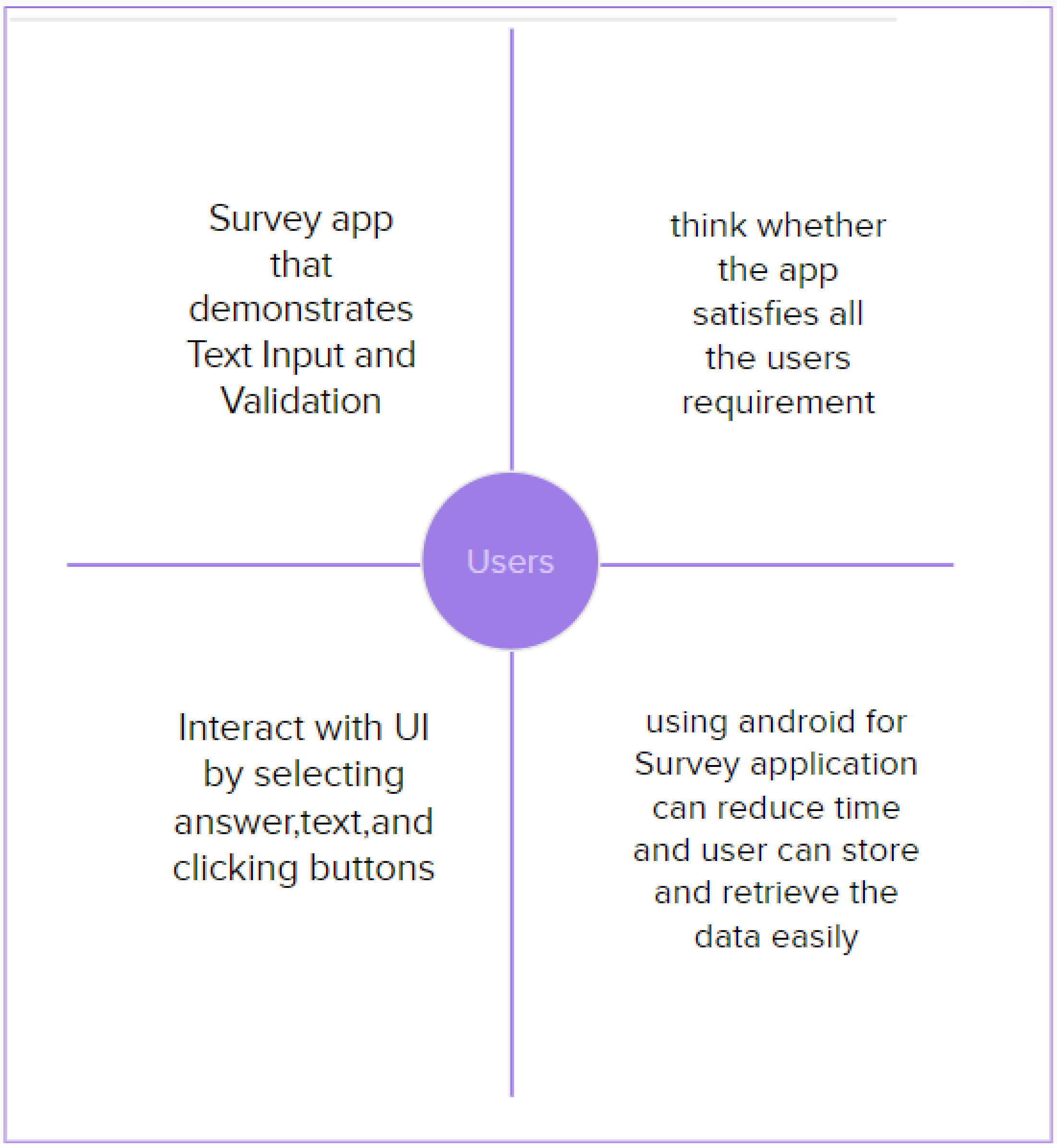
This category addresses the user's emotional state and answers questions like "What is the user feeling during this product experience?" and "What worries or excites the user?" An easy way to organize this information is to list the emotions being elicited followed by a short description of what is making the user feel this way. For example;

“Overwhelmed—too many decisions to make,” or, “Anxious—doesn’t want to waste their time.”

Does

This quadrant captures what the user physically does and how they do it. In other words, what actions does the user take and how do they take them?

Empathy map for Survey Application:


Survey app that demonstrates Text Input and Validation
think whether the app satisfies all the users requirement
Users
Interact with UI by selecting answer,text,and clicking buttons
using android for Survey application can reduce time and user can store and retrieve the data easily

The detailed explanation for the empathy map for survey application is given below:

Says:

The user says about the survey application that must be validate the input and checks for the correct results

Thinks:

The user think whether the survey application satisfies all their user requirements and easy to handle that is user friendly.

Feels:

The user feels that compare to paper work, using survey application can reduce time and cost. They can simply take survey at anywhere in the world by simply installing the survey application. The users data stored at the database, they can retrieve data by entering registered Email and Password.

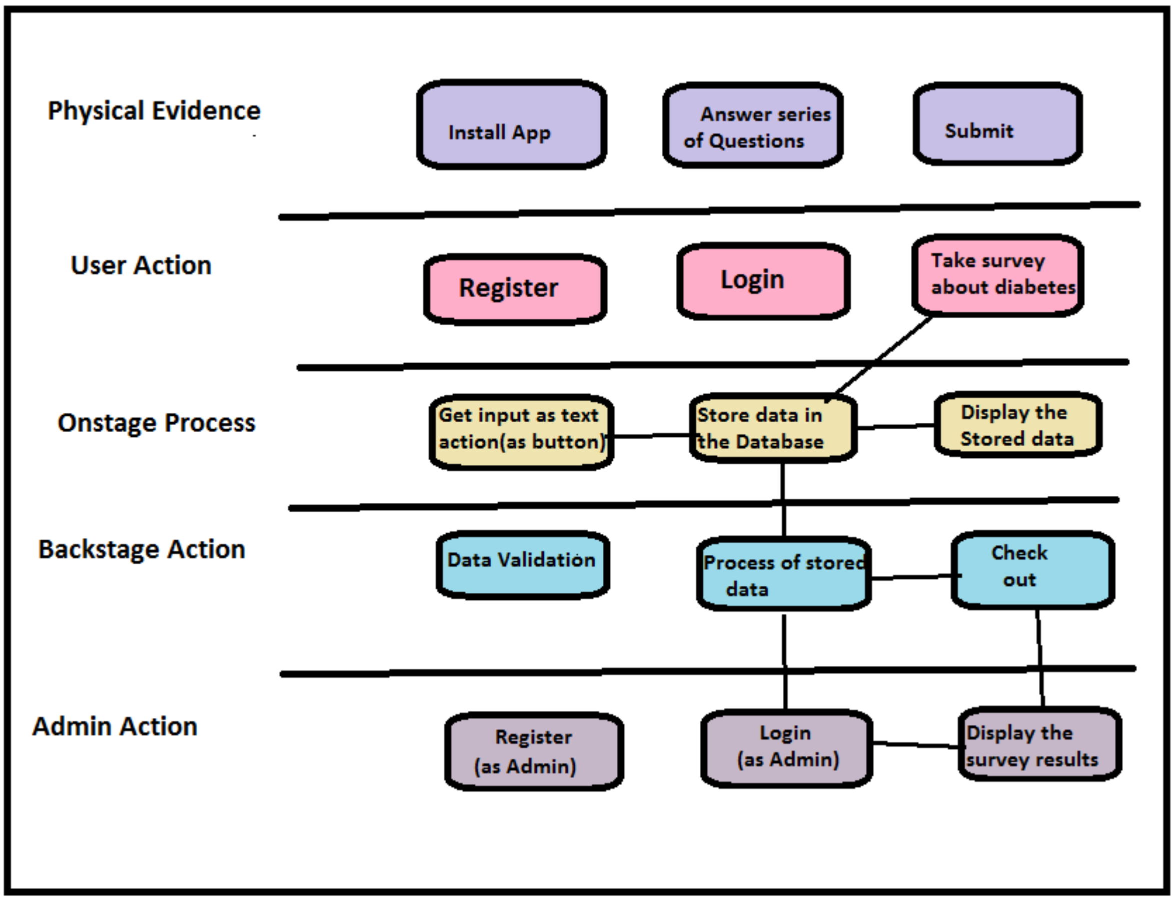
Does:

The User physically interact with the survey application with the help of User Interface by clicking buttons, entering the text and retrieveing the stored data.

Ideation & BrainstormingMap

Brainstorming is a situation where a group of people meet to generate new ideas and solutions around a specific domain of interest by removing inhibitions. People are able to think more freely and they suggest as many spontaneous new ideas as possible.

All the ideas are noted down without criticism and after the brainstorming session the ideas are evaluated.When brainstorming, or attempting to see a particular topic from all angles, linear tools — like lists — aren't always the best solution. In fact, thinking this way can hinder creativity. The solution is mind map. This organizational structure allows you to explore a central topic on a deep level through associated ideas and concepts which branch off from the center organically. Brainstorming is a group creativity technique by which effortsare made to find a conclusion for a specific problem by gathering a list of ideas spontaneously contributed by its members

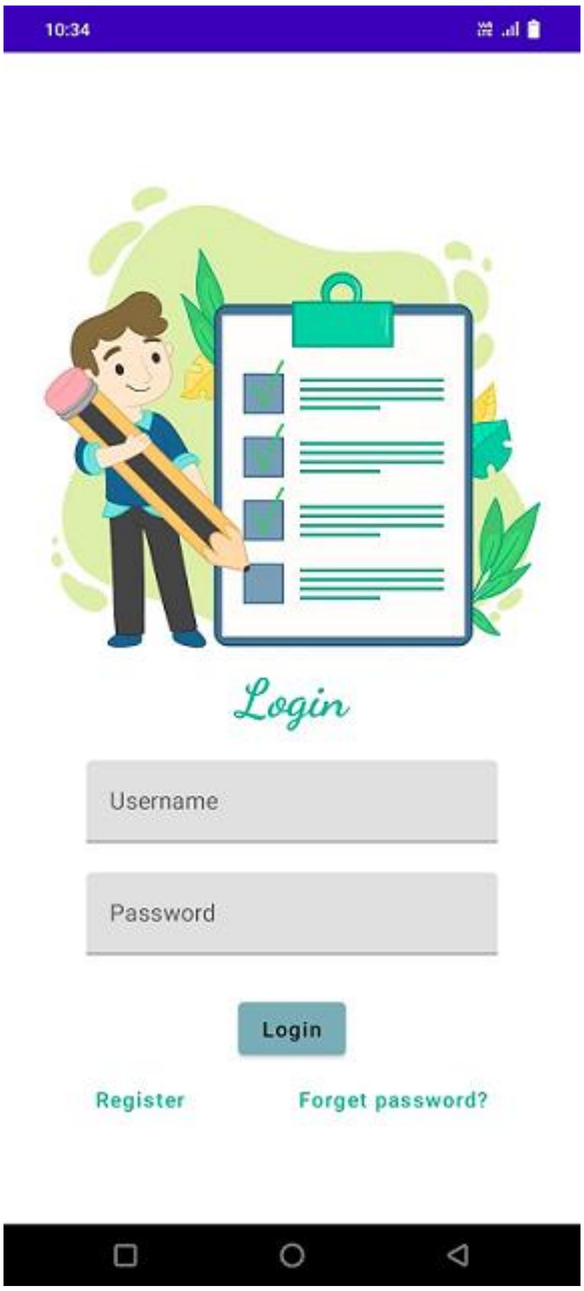


CHAPTER- III

RESULT

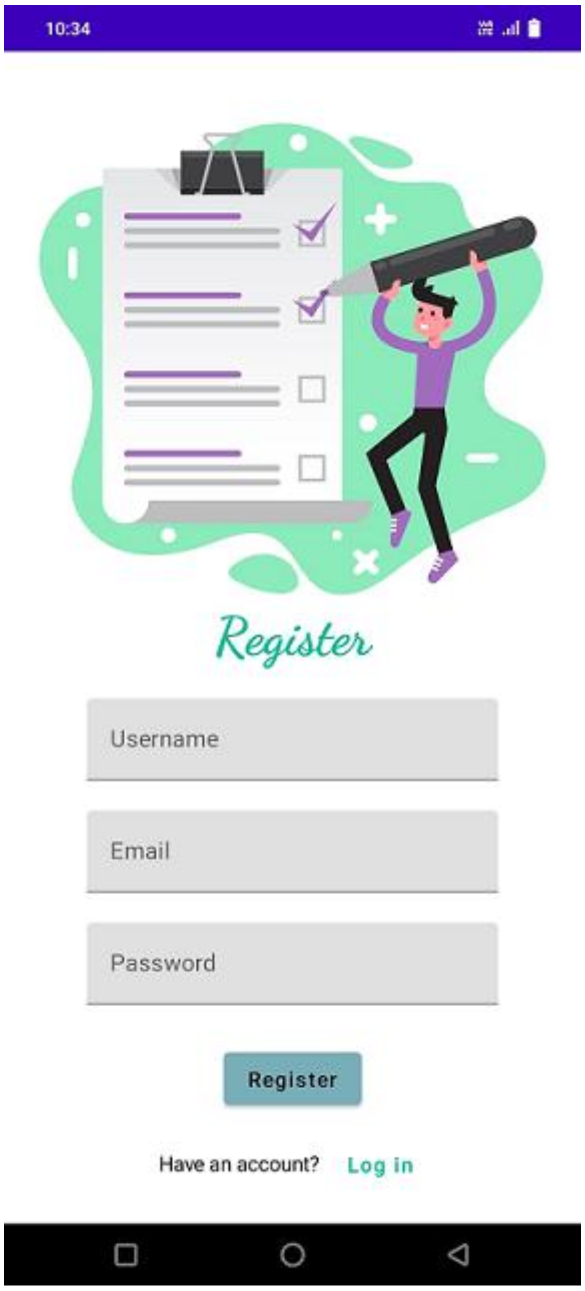
LOGIN PAGE

Starting point of the application is the Loginscreen.After registering with the user name and passwords defined by the user, the user logins.



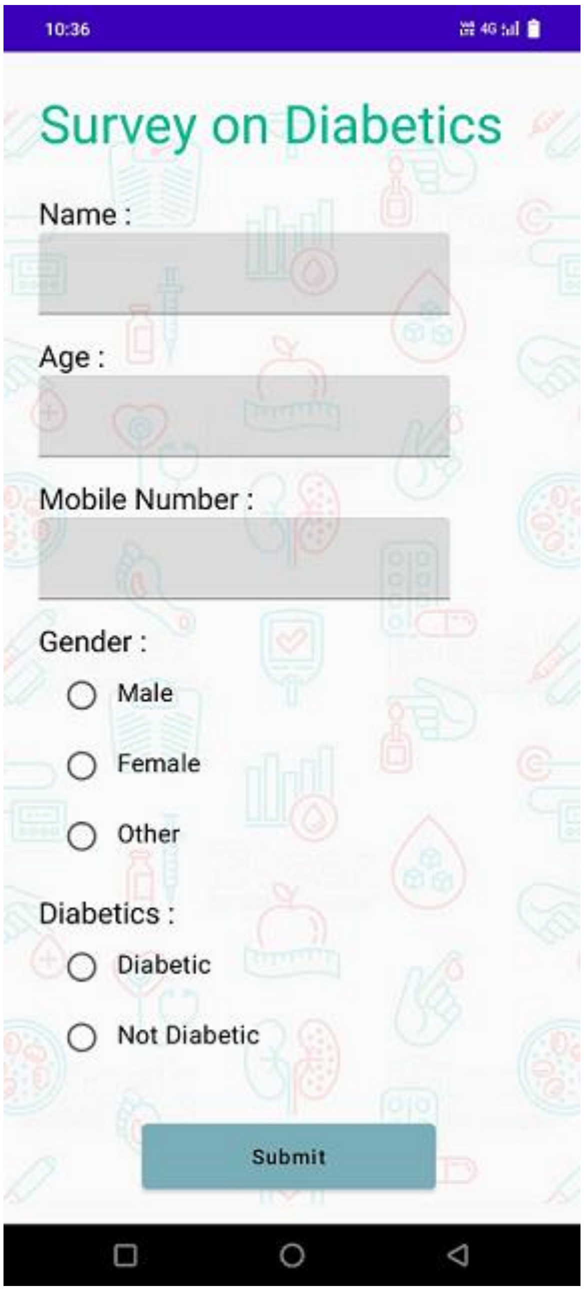
REGISTER PAGE

Firstly , user want to register by providing neccesary details.



MAIN PAGE

1.After LOGIN, Users directs to the main page. User can fill the series of question and click “SUBMIT” button to complete the Survey.



CHAPTER-IV

ADVANTAGE AND DISADVANTAGE

Advantage

1.Relatively easy to administer

2.Can be developed in less time (compared to other data-collection methods)

3.Cost-effective, but cost depends on survey mode

4.Can be administered remotely via online, mobile devices , mail, email, kiosk, or telephone.

5.Conducted remotely can reduce or prevent geographical dependence

6.Capable of collecting data from a large number of respondents

7.Numerous questions can be asked about a subject, giving extensive flexibility in data analysis

8.With survey software, advanced statistical techniques can be utilized to analyze survey data to determine validity, reliability, and statistical significance, including the ability to analyze multiple variables.

9.A broad range of data can be collected (e.g., attitudes, opinions, beliefs, values, behavior, factual).

10.Standardized surveys are relatively free from several types of Errors

11.Surveys provide a high level of general capabilityin representing a large population. Due to the usual hugenumber of people who answers survey, the data beinggathered possess a better description of the relative characteristics of the general population involved in the study. As compared to other methods of data gathering, surveys are able to extract data that are near to the exact attributes of the larger population.

12.When conducting surveys, you only need to pay for the production of survey questionnaires. If you need a larger sample of the general population, you can allot an incentive in cash or kind, which can be as low as $2 per person. On the other hand, other data gathering methods such as focus groups and personal interviews require researchers to pay more.

13.As questions in the survey should undergo careful scrutiny and standardization, they provide uniform definitions to all the subjects who are to answer the questionnaires. Thus, there is a greater precision in terms of measuring the data gathered.

14.Because of the high representativeness brought about by the survey method, it is often easier to find statisticallysignificant results than other data gathering methods

15.. Multiple variables can also be effectively analyzed using surveys

Disadvantage

The reliability of survey data may depend on the following factors:

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.

4.Surveys with closed-ended questions may have a lower validity rate than other question types.

4.Data errors due to question non-responses may exist. The number of respondents who choose to respond to a survey question may be different from those who chose not to respond, thus creating bias.

5.Survey question answer options could lead to unclear data because certain answer options may be interpreted differently by respondents. For example, the answer option “somewhat agree” may represent different things to different subjects, and have its own meaning to each individual respondent. ‘Yes’ or ‘no’ answer options can also be problematic. Respondents may answer “no” if the option “only once” is not available.

6.Customized surveys can run the risk of containing certain types of errors

CHAPTER- V

APPLICATIONS

Understanding the prevalence of diabetes: Diabetes survey applications can be used to collect data on the prevalence of diabetes among a particular population. This information can help healthcare professionals and policymakers to better understand the scope of the diabetes problem and develop targeted interventions.

Identifying risk factors: Diabetes survey applications can be used to identify risk factors for diabetes, such as lifestyle behaviors, family history, and demographic factors. This information can be used to develop interventions to prevent or delay the onset of diabetes in high-risk populations.

Assessing diabetes knowledge and self-management: Diabetes survey applications can be used to assess patients' knowledge of diabetes and their ability to manage the disease. This information can be used to develop patient education materials and support programs that address patients' specific needs

Evaluating diabetes treatment and outcomes: Diabetes survey applications can be used to evaluate the effectiveness of diabetes treatments and outcomes. This information can be used to identify areas where improvements can be made and develop strategies to improve patient outcomes.

Monitoring trends and changes over time: Diabetes survey applications can be used to monitor trends and changes in diabetes prevalence, risk factors, knowledge, self-management, treatment, and outcomes over time. This information can be used to evaluate the impact of interventions and inform future policy and practice decisions.

CHAPTER-VI

CONCLUSION

The project has been successfully completed by having established the users with the help of Android Studio tool. It consists of design plots, layouts plots, java codes, oncreate plots and computing plots. At the same time there is some scope for improvement in the future. It can be possible to make it more users friendly by adding more variety of functions to it.

CHAPTER-VII

FUTURE SCOPE

Integration with electronic health records (EHR): Diabetes survey applications could be integrated with EHR systems to allow for more efficient and accurate data collection and analysis. This could also facilitate the sharing of data among healthcare providers and improve patient care coordination

Use of artificial intelligence (AI): AI could be used to analyze diabetes survey data and identify patterns or trends that may not be immediately apparent. This could help healthcare providers to develop more targeted interventions and improve patient outcomes.

Incorporation of wearable technology: Diabetes survey applications could be integrated with wearable technology, such as glucose monitors and fitness trackers, to provide more comprehensive data on patients' health status and behaviors. This could help healthcare providers to develop personalized treatment plans and improve patient outcomes.

Gamification: Diabetes survey applications could be gamified to make the survey-taking experience more engaging and motivating for patients. This could improve patient participation and data quality.

Mobile app development: Diabetes survey applications could be developed as mobile apps to make them more accessible to patients. This could increase patient participation and improve the accuracy and completeness of data collection.

Overall, future enhancements to diabetes survey applications should focus on improving the accuracy, efficiency, and patient engagement of data collection and analysis, with the goal of improving patient outcomes and reducing the burden of diabetes on individuals and healthcare systems.

CHAPTER-VIII

APPENDIX

SOURCE CODE:

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)
}
```

REGISTRATION ACTIVITY.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)
}
```

MAIN ACTIVITY.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
            )
          }
        }
    }
}
```

ADMIN ACTIVITY.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}")
                    }
                }
            }
        }
    }
}
```

ANDROID MANIFEST.XML CODE

```
<?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>
```

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
  }

}
```