A Demonstration Of Text Input And Validation With Android Compose

1.INTRODUCTION

Overview

 The title "Compose Input: A Demonstration of Text Input and Validation with

Android Compose" suggests that the project is a demonstration of text input and

validation using Android Compose, a modern toolkit for building native Android UIs.

 The project is likely to involve creating a sample app that allows users to input text in

various fields, such as text fields, dropdowns, and radio buttons, and then validate the

input to ensure that it meets certain requirements, such as minimum length, data

format, or consistency with other inputs.

 The project may also showcase how to handle errors and display appropriate feedback

to the user, such as error messages or visual cues. Additionally, it may highlight some

of the benefits of using Android Compose for UI development, such as its declarative

syntax, state management, and composability features.

Purpose

- The purpose of the title "Compose Input: A Demonstration of Text Input and Validation with Android Compose" is to showcase the capabilities of Android Compose for handling text input and validation in a sample application. The title suggests that the project aims to demonstrate how Android Compose can be used to create a modern and efficient UI for text input and validation

- Additionally, the title suggests that the project may provide practical examples of how to implement text input and validation in Android Compose, including best practices for handling user input and displaying validation errors. This could be helpful for developers who are new to Android Compose or who are looking to improve their UI development skills.

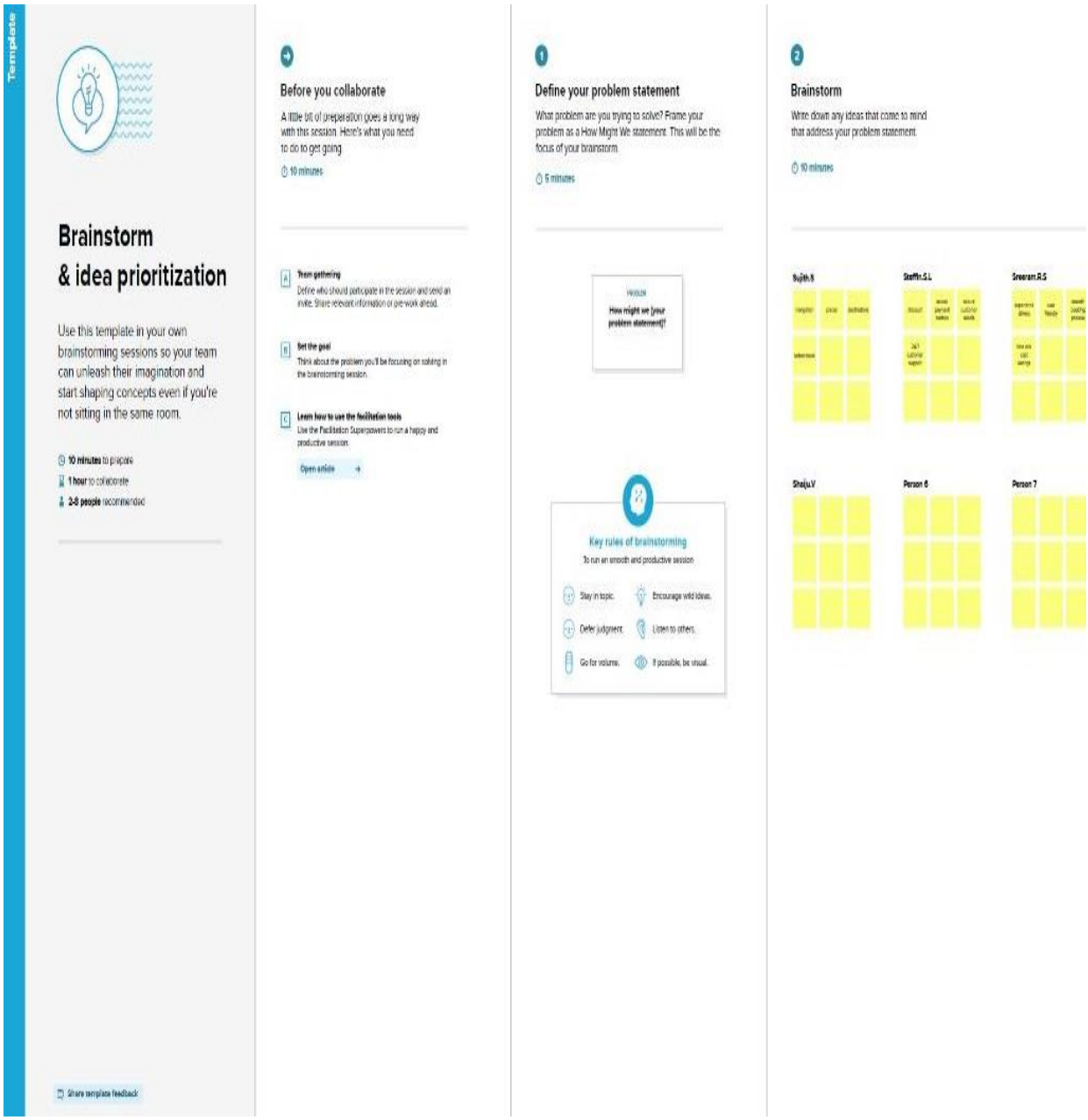
- The project aims to demonstrate the benefits of using Android Compose for UI development, such as its declarative syntax, state management, and features. It will showcase how to create various types of text input fields, including text fields, dropdowns, and radio buttons, and how to validate user input to ensure it meets certain requirements. The project will also provide examples of best practices for handling user input and displaying validation errors, which can be helpful for developers who are new to Android Compose or who want to improve their UI development skills.

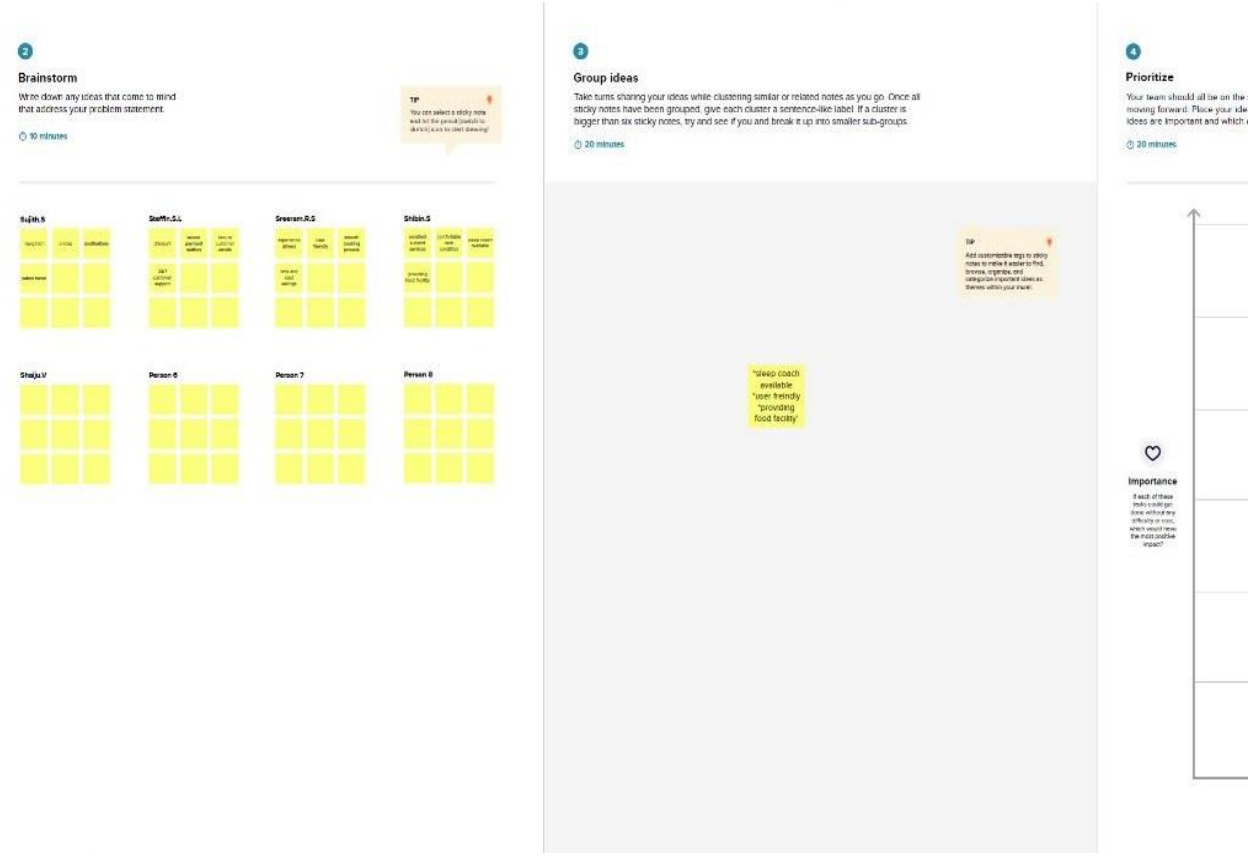
- Overall, the project aims to provide a clear and practical demonstration of how to use Android Compose for text input and validation, and to highlight the benefits of using this modern toolkit for building native Android apps.

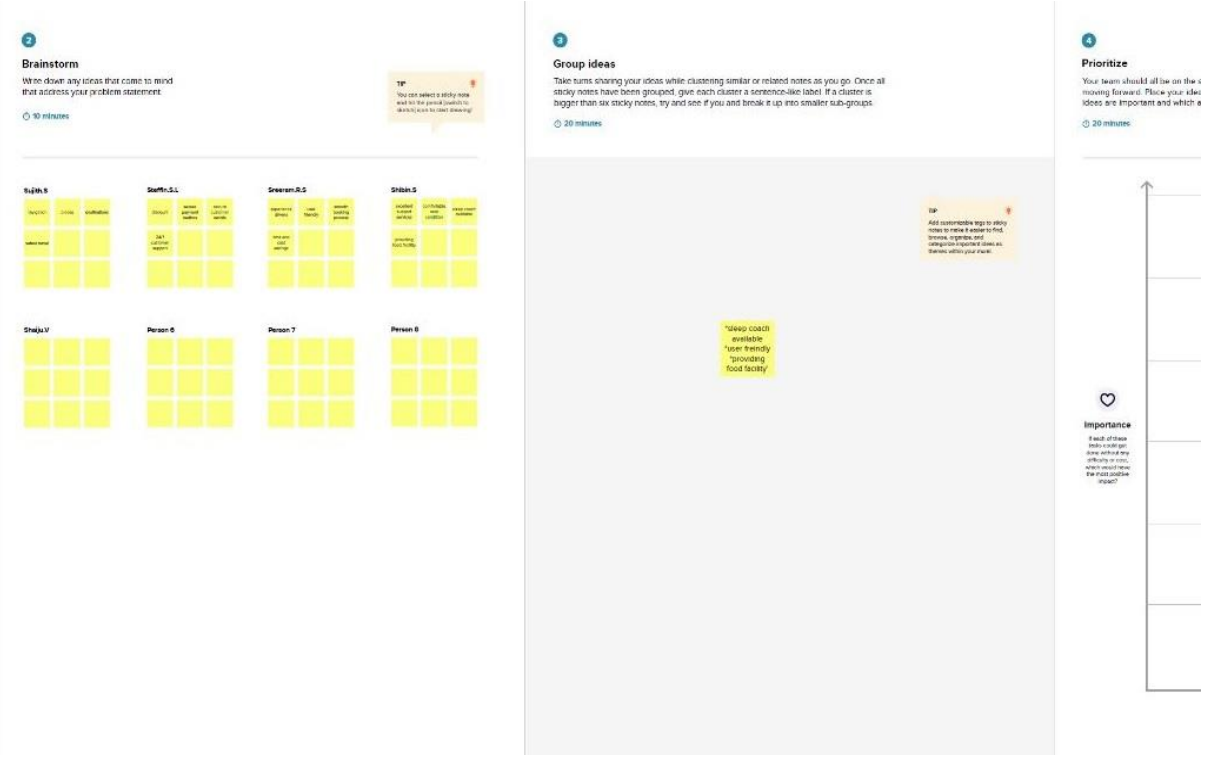
2.Problem Definition & Design Thinking

2.1 Empathy Map

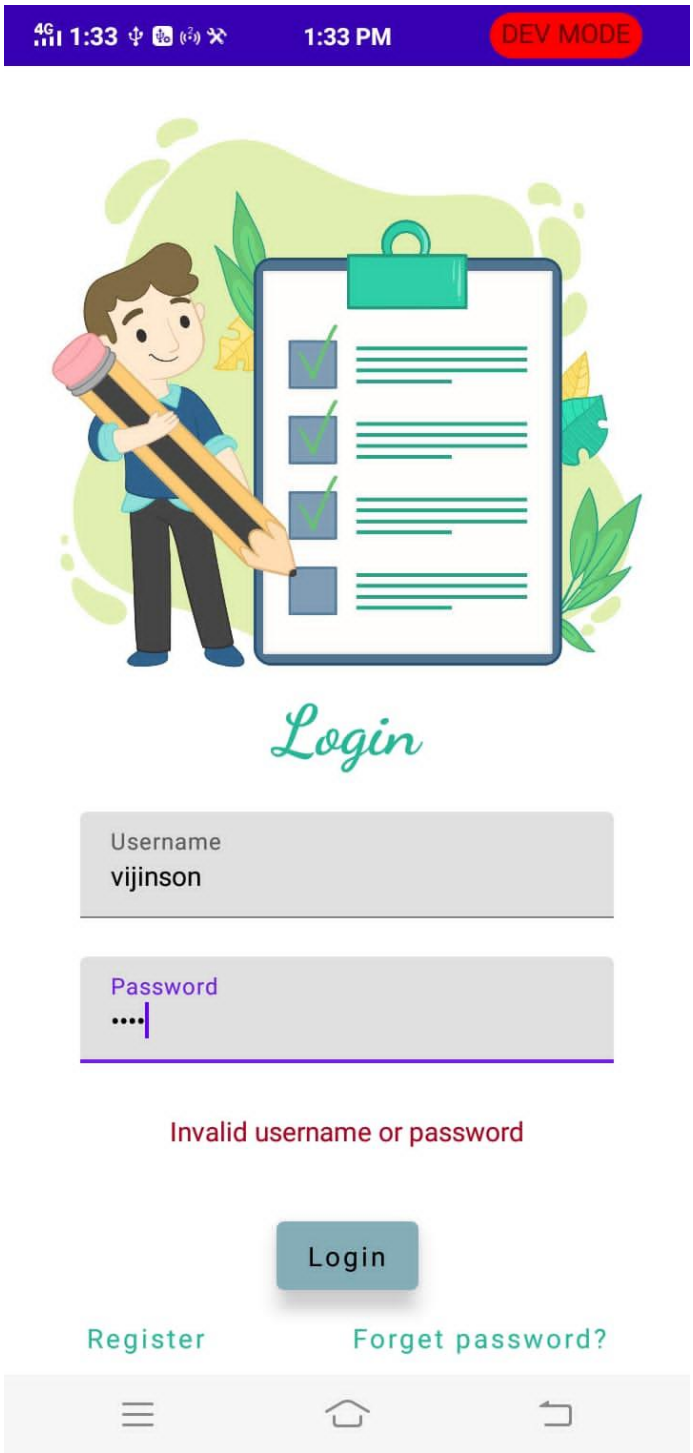
2.2 Ideation and Brainstorming Map

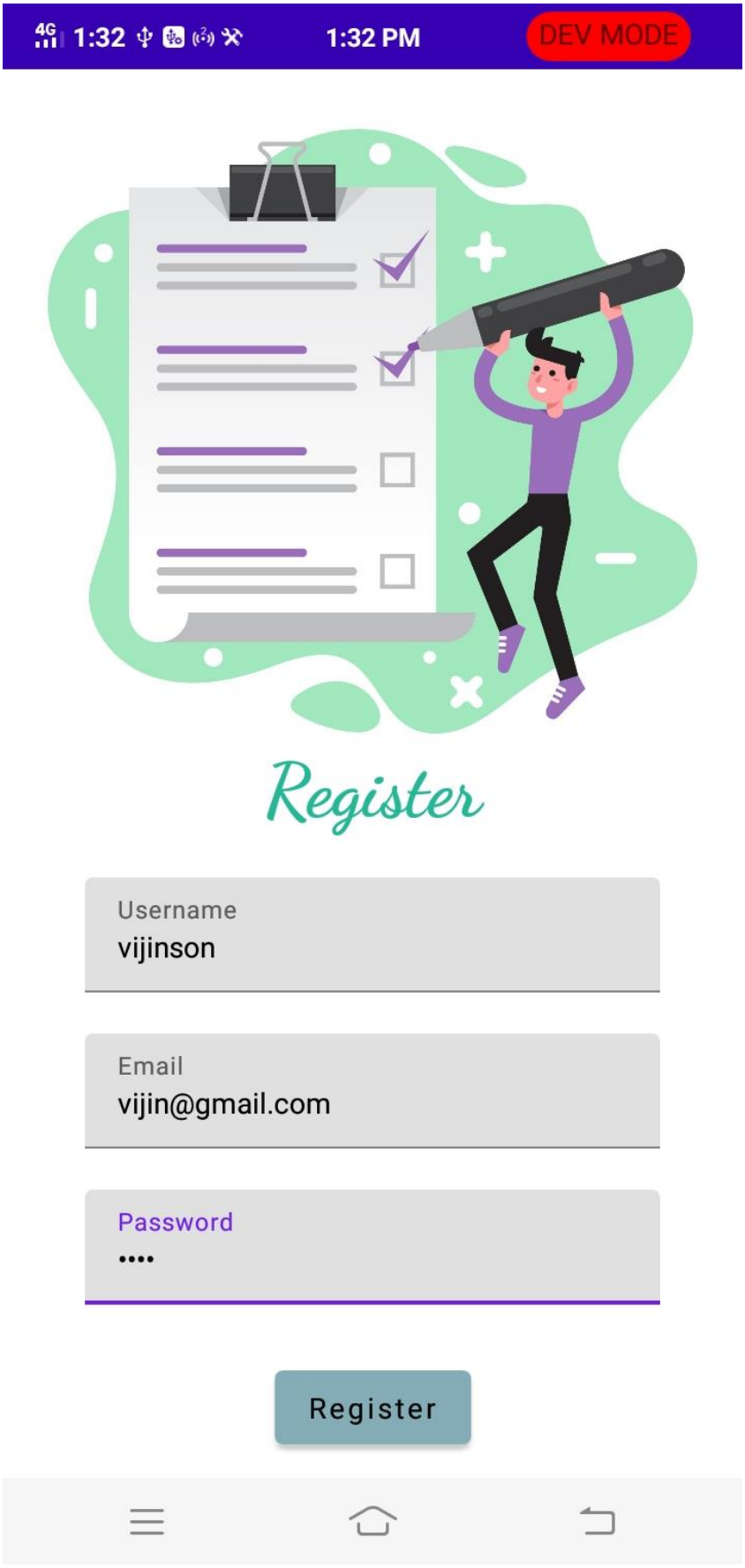


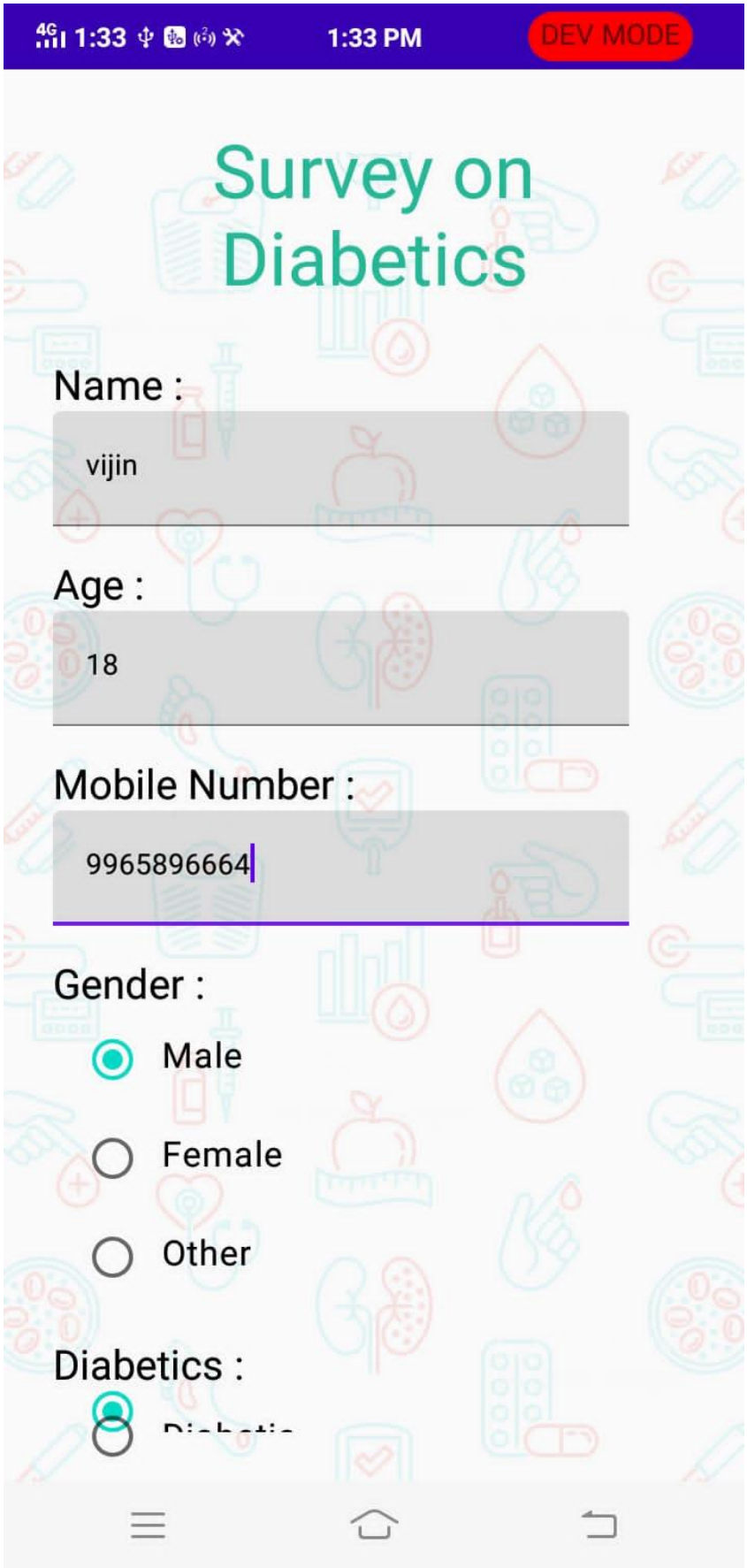

2
Brainstorm
Write down any ideas that come to mind that address your problem statement.
10 minutes
TIP
Sujith.S
Steffin.S.L
Sreeram.R.S
Shibin.S
Shaju.V
Person 6
Person 7
Person 8
3
Group ideas
Take turns sharing your ideas while clustering similar or related notes as you go. Once all sticky notes have been grouped, give each cluster a sentence-like 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
TIP
"sleep coach
available
"user friendly
"providing
food facility"
4
Prioritize
20 minutes
Importance


2
Brainstorm
Write down any ideas that come to mind that address your problem statement.
10 minutes
TIP
Sujith.S
Steffin.S.L
Sreeram.R.S
Shibin.S
Shaju.V
Person 6
Person 7
Person 8
3
Group ideas
Take turns sharing your ideas while clustering similar or related notes as you go. Once all sticky notes have been grouped, give each cluster a sentence-like 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
TIP
"sleep coach
available
"user friendly
"providing
food facility"
4
Prioritize
20 minutes
Importance

RESULT


1:33
1:33 PM
DEV MODE
Login
Username
vijinson
Password
Invalid username or password
Login
Register
Forget password?


1:32
1:32 PM
DEV MODE
Register
Username
vijinson
Email
vijin@gmail.com
Password
••••
Register


1:33
1:33 PM
DEV MODE
Survey on Diabetics
Name :
vijin
Age :
18
Mobile Number :
9965896664
Gender :
Male
Female
Other
Diabetics :

ADVANTAGES & DISADVANTAGES

Advantages

- The project demonstrates how to use Android Compose to build a modern and efficient UI for text input and validation in a native Android app, which can be useful for developers who are new to this toolkit.

- By showcasing the features of Android Compose, the project helps developers understand the benefits of using this toolkit for building native Android apps, including its declarative syntax, state management, and composability features.

- The project provides practical examples of how to implement text input and validation in Android Compose, including best practices for handling user input and displaying validation errors.

- The project can help improve the quality of Android apps by providing guidelines for ensuring that user input is validated before being processed or stored

Disadvantages

- The project may not cover all possible use cases for text input and validation, so developers may need to adapt the examples to fit their specific needs.

- The project may require some familiarity with Android development and the Kotlin programming language, which could be a barrier for developers who are new to Android development.

- The project may not address issues related to security or data privacy, which are important considerations for apps that handle sensitive user data.

APPLICATIONS

- Process Validation is the collection and evaluation of data from the process design stage through commercial production, which establishes scientific evidence that a process is capable of consistently delivering a product.

- Process validation is defined as documented verification that the manufacturing approach operated according to its specifications consistently generates a product complying with its predefined quality attributes and release specifications.

- Process Validation is the collection and evaluation of data from the process design stage through commercial production, which establishes scientific evidence that a process is capable of consistently delivering a product

CONCLUSION

• The data validation process is an important step in data and analytics workflows to filter quality data and improve the efficiency of the overall process. It not only produces data that is reliable, consistent, and accurate but also makes data handling easier.

• Successfully validating a process may reduce the dependence upon intensive in process and finished product testing. Input validation prevents a wide range of attacks that can be performed against a website or application. These cyberattacks can cause the theft of personal information, allow unauthorized access to other components, and/or prevent a website/application from functioning.

FUTURE SCOPE

- Input validation is performed to ensure only properly formed data is entering the workflow in an information system, preventing malformed data from persisting in the database and triggering malfunction of various downstream components. Verification and validation are the main elements of software testing workflow because they: Ensure that the end product meets the design requirements. Reduce the chances of defects and product failure. Ensures that the product meets the quality standards and expectations of all stakeholders involved.

- Continued Process Verification involves ongoing validation during production of the commercial product to ensure the process designed and qualified in the previous stages continues to deliver consistent quality. One of the main aims of this stage is to detect and resolve process drift. Data validation provides accuracy, cleanness, and completeness to the dataset by eliminating data errors from any project to ensure that the data is not corrupted. While data validation can be performed on any data, including data within a single application such as Excel creates better results.

Appendix

A. Source code

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_NU
MBER)),

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
}



}



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

}