COMPOSE INPUT: A DEMONSTRATION OF TEXT INPUT AND VALIDATION WITH ANDROID COMPOSE

PROJECT REPORT

Submitted by

Hari Krishnan.R (20203111506226)

Genin Stephen.G(20203111506225)

DonioD.S(20203111506224)

Danish franklinF.S(20203111506222)

Abinash.M(20203111506202)

Submitted to Manonmaniam Sundaranar University. Tirunelveli

In partial fulfilment for the award of

BACHELOR OF SCIENCE IN COMPUTER SCIENCE

Under the Guidance of

Prof. D.H. KITTY SMAILIN M.Sc., M.Phil.

DEPARTMENT OF PG COMPUTER SCIENCE

NESAMONY MEMORIAL CHRISTIAN COLLEGE,

MARTHANDAM

KANYAKUMARI DISTRICT -629165

Re-accredited with 'A' grade by NAAC

MARCH 2023

DECLARATION

I hereby declare that the project work entitled "**COMPOSE INPUT: A DEMONSTRATION OF TEXT INPUT AND VALIDATION WITH ANDROID COMPOSE"** is an original work done by me in partial fulfillment of the degree of **BACHELOR OF SCIENCE IN COMPUTER SCIENCE.** The study has been carried under the guidance of **Prof. D. H. KITTY SMAILIN M.Sc., M.Phil. Department of PG COMPUTER SCIENCE, Nesamony Memorial Christian College, Marthandam.** I declare that this work has not been submitted elsewhere for the award of any degree.

Place: Marthandam.

Hari Krishnan.R (20203111506226)

Genin Stephen.G(20203111506225)

DonioD.S(20203111506224)

Danish franklinF.S(20203111506222**)**

Abinash.M(20203111506202)

ACKNOWLEDGEMENT

First of all, I offer my prayers to God almighty for blessing me to complete this internship programme successfully. I express my sincere thanks to**, Dr. K. Paul Raj M.Sc., M.Phil., M.Ed., M.Phil.(Edu)., Ph.D.,** Principal Nesamony Memorial Christian college, for his official support to do my work.

I express my profound thanks to **Dr. Dr. D. Latha M.Sc., M.Phil., Ph.D.,** HOD, Department of PG Computer Science for his encouragement given to undertake this project.

I extend my deep sense of gratitude to, **Prof**. **D. H. KITTY SMAILIN M.Sc., M.Phil.** Department of PG Computer Science for guiding me in completing this project work successfully.

My special thanks to other Faculty members of Department of PG Computer Science for Guiding me in Completing this project.

I also wish to extend a special word of thanks to my parents, friends and all unseen hands that helped me for completing this project work successfully.

I want to acknowledge how much you've done. I am so graceful to done a project. This has been such a blessing

It is very useful to us thankyou for giving the great opportunity to us .Thank you for naan muddhalvan

CONTENTS

PREFACE

1.INTRODUCTION

1.1 Overview 1.2 Purpose

2. PROBLEM DEFINITION & DESIGN THINKING

2.1 Empathy Map 2.2 Ideation & Brainstorming Map

3. RESULT

3.1 Data Model 3.2 Activity & Screenshot

4. TRAILHEAD PROFILE PUBLIC URL

4.1 Trailhead Profile

5. ADVANTAGES & DISADVANTAGE

5.1 Advantages

5.2 Disadvantage

6. APPLICATIONS

7. CONCLUSION

8. FUTURE SCOPE

1. INTRODUCTION :

Android Compose is a modern toolkit for building Android user interfaces using Kotlin. With Compose, developers can create UI components using a declarative approach, which makes it easier to write, read and maintain code. In this tutorial, we will be building a simple text input form with validation using Android Compose.

The demonstration will guide you through the process of creating a Composable function called TextInputDemo, which will contain a TextField and Button to check whether the text input is empty and display an error message if it is, or a success message if it is not. This tutorial assumes that you have basic knowledge of Android development and Kotlin.

Android Compose is a modern UI toolkit for building native Android apps using declarative programming techniques. One of the most common use cases for Compose is handling user input and validation. In this demo, we will showcase how to use Compose to create a simple form with text input and validation in an Android app. We will use Compose's OutlinedTextField element and mutable state variables to track the current input and any validation errors. This demo will demonstrate how Compose's reactive and declarative approach can simplify input handling and make it more efficient. By the end of this demonstration, you will have a better understanding of how Compose input works and how it can be used to enhance the user experience in Android apps.

1.1 Overview

The Compose Input demonstration shows how to build a simple text input form with validation using Android Compose.

The demonstration will guide you through the process of creating a Composable function called TextInputDemo, which will contain a TextField and a Button. The TextField will be used to take user input, and the Button will be used to validate the input and display an error message if it is empty, or a success message if it is not.

The tutorial will start by introducing Android Compose and its benefits, then show how to set up the project and add the necessary dependencies. It will then dive into creating the TextInputDemo Composable and explain how it works. Finally, the tutorial will show how to add the TextInputDemo Composable to the MainActivity and run the app.

By the end of the tutorial, you should have a better understanding of how to build a text input form with validation using Android Compose, and how to use Composable to create simple and efficien user interfaces.

In this demo, we will create a simple form with text input and validation using Android Compose. The form will have two input fields, one for the user's name and one for their email address. We will use Compose's OutlinedTextField element to create the text input fields, and we will use mutable state variables to track the crrent input and any validation errors.

demonstration will cover the following steps:

Creating a new Compose project in Android Studio.

Adding the necessary dependencies for Compose and setting up the layout for the form.

Creating the OutlinedTextField elements for the name and email input fields.

Implementing validation for the name and email fields using mutable state variables and regular expressions.
Displaying validation errors to the user and disabling the submit button until all fields are valid.

Testing the form by inputting invalid and valid data and verifying that the validation works as expected.

By the end of this demonstration, you will have a working example of how to handle user input and validation using Compose in an Android app.

Step 1: Download Android Studio

To set up Android Studio, you need to first download the IDE from the official Android Studio download page. Choose the version that is compatible with your operating system and download the installer. Android Studio is available for Windows, macOS, and Linux. Once the download is complete, run the installer and follow the instructions to install Android Studio on your computer.

Step 2: Install the Required Components

During the installation process, Android Studio will prompt you to install the required components. These include the Android SDK, Android Virtual Device (AVD) Manager, and the Android Emulator. The Android SDK is

a collection of libraries and tools that developers use to build Android applications. The AVD Manager is used to create and manage virtual devices for testing applications. The Android Emulator is a virtual device that allows developers to test their applications without having to use a physical device.

Step 3: Configure Android Studio

After installing Android Studio, you need to configure it before you can start using it. When you launch Android Studio for the first time, you will be prompted to configure the IDE. Choose the "Standard" configuration and click on "Next". In the next screen, you can choose the theme of the IDE and click on "Next" again. You can also customize the settings based on your preferences.

Step 4: Create a New Project

Once Android Studio is configured, you can start creating your first Android application. To create a new project, click on "Start a new Android Studio project" on the welcome screen, or select "New Project" from the "File" menu. You will be prompted to choose the project name, package name, and other project details. You can also choose the minimum SDK version, which determines the minimum version of Android that the application can run on.

Step 5: Build Your Application

Once your project is created, you can start building your application using the various tools and features provided by Android Studio. You can use the visual layout editor to design the user interface, write code in Java or Kotlin, and use the Android SDK to access device features such as the camera, sensors, and GPS. You can also use the built-in debugging tools to troubleshoot issues and optimize your application.

Step 6: Test Your Application

Testing your application is an important step in the development process. Android Studio comes with an emulator that allows you to test your application on different virtual devices. You can also connect your Android device to your computer and test your application directly on the device. Use the “Run” button in Android Studio to launch your application and test it on the emulator or device. You can also use the builtin profiler to analyze the performance of your application and identify any bottlenecks or performance issues. In conclusion, setting up Android Studio is a crucial step in developing Android applications. By following these steps, you can easily set up Android Studio on your computer and start building high-quality Android applications. Android Studio provides a powerful set of tools and features that make the development process easier and more efficient.

Components in Android Studio 1. Manifest File

The AndroidManifest.xml file is a crucial component of any Android application. It provides essential information about the application to the Android operating system, including the application's package name, version, permissions, activities, services, and receivers. The manifest file is required for the Android system to launch the application and to determine its functionality. Here are some of the key uses of the manifest file in an Android application:

Declaring Application Components: The manifest file is used to declare the various components of an Android application, such as activities, services, and broadcast receivers. These components define the behavior and functionality of the application, and the Android system uses the manifest file to identify and launch them.

Specifying Permissions: Android applications require specific permissions to access certain features of the device, such as the camera, GPS, or storage. The manifest file is used to declare these permissions, which the Android system then checks when the application is installed. If the user has not been granted the required permissions, the application may not be able to function correctly.

Defining App Configuration Details: The manifest file can also be used to define various configuration details of the application, such as the application's name, icon, version code and name, and supported screens. These details help the

Android system to identify and manage the application properly.

Declaring App-level Restrictions: The manifest file can be used to declare certain restrictions at the app level, such as preventing the application from being installed on certain devices or specifying the orientation of the app on different screens.

In summary, the manifest file is an essential part of any Android application. It provides important information about the application to the Android system and enables the system to launch and manage the application correctly. Without a properly configured manifest file, an Android application may not be able to function correctly, or it may not be installed at all.

2. Build.Gradle

Build.gradle is a configuration file used in Android Studio to define the build settings for an Android project. It is written in the Groovy programming language and is used to configure the build process for the project. Here are some of the key uses of the build.gradle file:

Defining Dependencies: One of the most important uses of the build.gradle file is to define dependencies for the project. Dependencies are external libraries or modules that are required by the project to function properly. The build.gradle file is used to specify which dependencies the project requires, and it will automatically download and include those dependencies in the project when it is built.

Setting Build Options: The build.gradle file can also be used to configure various build options for the project, such as the

version of the Android SDK to use, the target version of Android, an the signing configuration for the project.

Configuring Product Flavors: The build.gradle file can be used to configure product flavors for the project. Product flavors allow developers to create different versions of their application with different features or configurations. The build.gradle file is used to specify which product flavors should be built, and how they should be configured.

Customizing the Build Process: The build.gradle file can also be used to customize the build process for the project. Developers can use the build.gradle file to specify custom build tasks, define build types, or customize the build process in other ways.

Overall, the build.gradle file is a powerful tool for configuring the build process for an Android project. It allows developers to define dependencies, configure build options, customize the build process, and more. By understanding how to use the build.gradle file, developers can optimize the build process for their projects and ensure that their applications are built correctly and efficiently.

3. Git

Git is a popular version control system that allows developers to track changes to their code and collaborate with other team members. Android Studio includes built-in support for Git, making

it easy to manage code changes and collaborate with others on a project. Here are some of the key uses of Git in Android Studio:

Version Control: Git allows developers to track changes to their code over time. This means that they can easily roll back to a previous version of their code if needed, or review the changes made by other team members.

Collaboration: Git enables multiple developers to work on the same codebase simultaneously. Developers can work on different features or parts of the codebase without interfering with each other and mrge their changes together when they are ready.

Branching and Merging: Git allows developers to create branches of their codebase, which can be used to work on new features or bug fixes without affecting the main codebase. When the changes are complete, the branch can be merged back into the main codebase.

Code Review: Git allows team members to review each other's code changes before they are merged into the main codebase.
This can help ensure that the code is of high quality and meets the project's requirements.

Android Studio includes a built-in Git tool that allows developers to perform common Git tasks directly within the IDE. Developers can create new repositories, clone existing ones, and manage branches and commits. Android Studio also provides a visual diff tool that makes it easy to see the changes made to the

codebase over time. To use Git in Android Studio, developers need to first initialize a Git repository for their project. Once the repository is set up, they can use the Git tool in Android Studio to manage changes to their code, collaborate with others, and review code changes.

In summary, Git is a powerful version control system that is essential for managing code changes and collaborating with other team members. Android Studio includes built-in support for Git, making it easy for developers to manage their code changes directly within the IDE.

4. Debug

Debugging is an essential part of software development, and Android Studio provides a robust set of debugging tools to help developers identify and fix issues in their applications. Here are some of the key uses of debugging in Android Studio.

Identifying Issues: Debugging helps developers identify issues in their code by allowing them to inspect variables, evaluate expressions, and step through the code line by line. This allows developers to pinpoint exactly where a problem is occurring and fix it more quickly.

Optimizing Performance: Debugging can also be used to optimize the performance of an application by identifying bottlenecks or areas of inefficient code. By profiling an application while it is running, developers can identify areas of the

code that are causing slow performance and make changes to improve performance.

Testing and Validation: Debugging is also useful for testing and validating an application. By stepping through code and inspecting variables, developers can ensure that the application is behaving as expected and that it is producing the desired output.

Android Studio provides a comprehensive set of debugging tools, including breakpoints, watches, and the ability to evaluate expressions in real time. Developers can use these tools to inspect variables, step through code, and identify issues in their applications.

To use the debugging tools in Android Studio, developers need to first configure their project for debugging by adding breakpoints to their code. Breakpoints are markers that tell the debugger to pause execution at a certain point in the code. Once the breakpoints are set, developers can run their application in debug mode and step through the code line by line, inspecting variales and evaluating expressions as they go.

In summary, debugging is a critical part of software development, and Android Studio provides a robust set of debugging tools to help developers identify and fix issues in their applications. By using these tools, developers can optimize performance, test and validate their code, and improve the quality of their applications.

5. App Inspection Inspector

App Inspection is a feature in Android Studio that allows developers to inspect and debug their Android applications. It provides a suite of tools for analyzing the performance of the application, identifying and fixing errors, and optimizing the code. Here are some of the key features and uses of App Inspection:

Performance Analysis: App Inspection provides tools for analyzing the performance of an Android application. Developers can use these tools to identify performance bottlenecks, such as slow database queries or inefficient network requests, and optimize the code to improve performance.

Error Detection and Debugging: App Inspection allows developers to detect and debug errors in their Android applications. It provides tools for tracking down errors and identifying the root cause of the issue, making it easier to fix bugs and imrove the stability of the application.

Memory Management: App Inspection provides tools for managing the memory usage of an Android application. Developers can use these tools to identify memory leaks and optimize the code to reduce memory usage, which can improve the performance and stability of the application.

Network Profiling: App Inspection includes tools for profiling network traffic in an Android application. Developers can use these tools to monitor network requests, identify slow or inefficient requests, and optimize the code to improve network performance.

Overall, App Inspection is a valuable tool for Android developers. It provides a suite of tools for analyzing and debugging Android applications, identifying and fixing errors, and optimizing the code for improved performance and stability. By using App Inspection, developers can ensure that their Android applications are of the highest quality and provide the best possible user experience.

6. Build Variants

Build variants in Android Studio are different versions of an Android app that can be built from the same source code. They are typically used to create multiple versions of an app that target different device configurations or use cases. Build variants are configured in the build.gradle file and can be built and installed separately from each other. Here are some examples of how build variants can be used.

Debug and Release Variants: The most common use of build variants is to create a debug variant and a release variant of an app. The debug variant is used for testing and debugging the app during development, while the release variant is used for production and is optimized for performance and stability.

Flavors: Build variants can also be used to create different flavors of an app, which can have different features or configurations. For example, an app might have a free version and a paid version, or aversion that targets tablets and a version that targets phones.

Build Types: Build variants can also be used to create different build types, which can have different build options or signing configurations. For example, an app might have a debug build type and a release build type, each with its own set of build options.

Overall, build variants are a powerful tool for Android developers. They allow developers to create different versions of an app from the same source code, which can save time and improve the quality of the app. By using build variants, developers can easily target different device configurations or use cases, create different versions of the app with different features or configurations, and optimize the app for performance and stability.

1.2 Purpose

The purpose of the Compose Input demonstration is to show how to use Android Compose to build a simple text input form with validation. The demonstration aims to provide developers with a clear understanding of how to create a Composable function that contains a TextField and a Button, and how to validate user input.

By following the tutorial, developers can learn how to use Compose to create efficient and effective user interfaces, and how to handle user input in a way that provides a good user experience. The tutorial also highlights the benefits of using Android Compose, such as a declarative approach to building UI components and improved code readability and maintainability.

The purpose of this demo is to showcase how Android Compose can be used to handle user input and validation in an efficient and

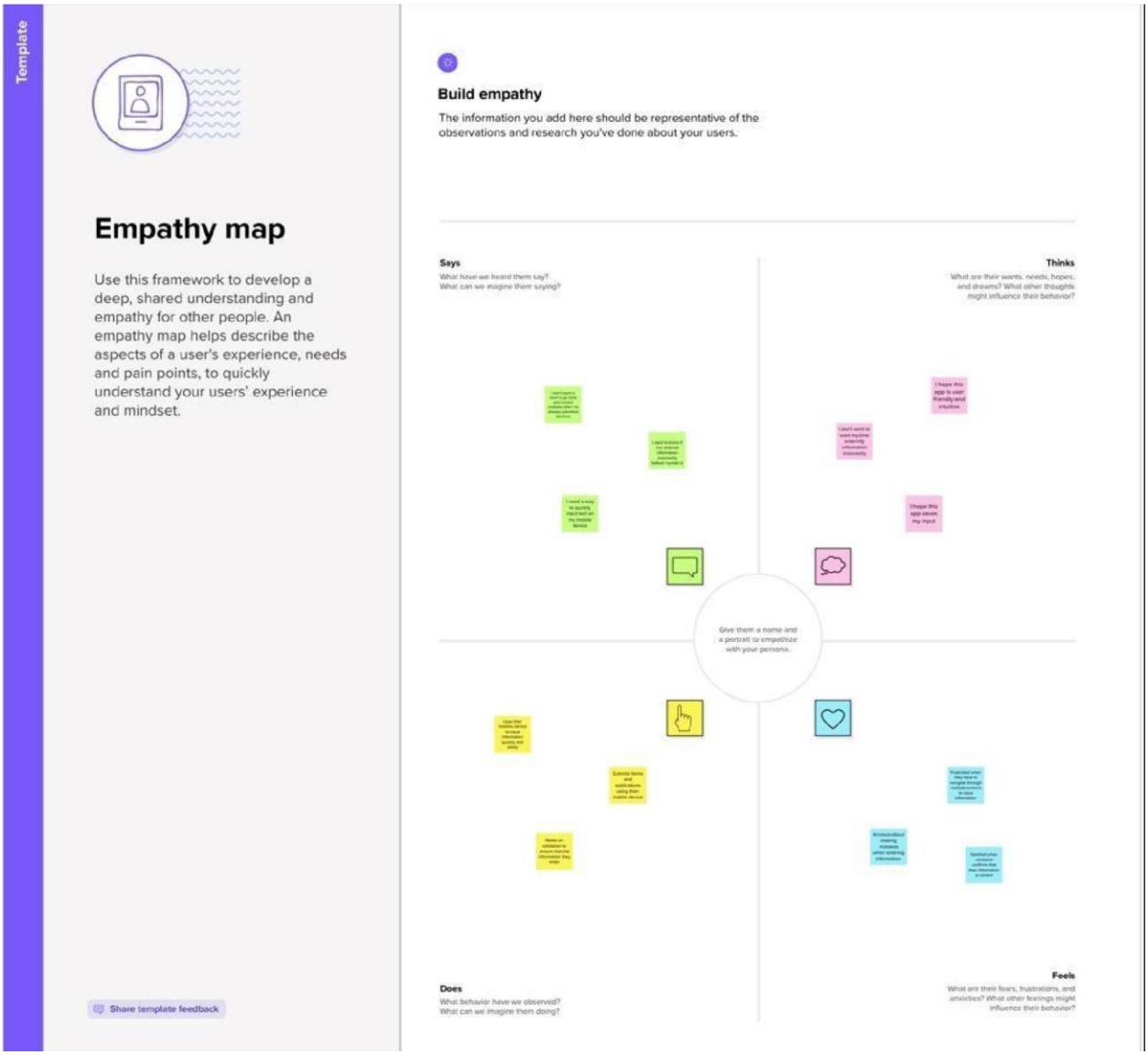
reactive way. User input is a fundamental aspect of many Android apps, and it is essential to ensure that the input is correct and valid. With Compose, developers can create UIs that respond to changes in input and provide real-time validation feedback to the user.

By demonstrating how to create a simple form with text input and validation, this demo will highlight Compose's key features and capabilities, including its declarative programming model, reactive UI updates, and state management techniques. The demo will provide a step-by-step guide to creating a form with input validation using Compose's OutlinedTextField element and mutable state variables, providing a practical example of how Compose can be used to handle user input and validation in Android apps.

Ultimately, the goal of this demo is to help developers gain a better understanding of how Compose input works and how it can be used to create more efficient, user-friendly Android apps.

Overall, the purpose of the Compose Input demonstration is to provide developers with a practical example of how to use Android Compose to create a common UI component, and to help them get started with building their own composable.

2. Problem Definition & Design Thinking Empathy map:



Ideation & Brainstroming Map:

package com.example.surveyapplication

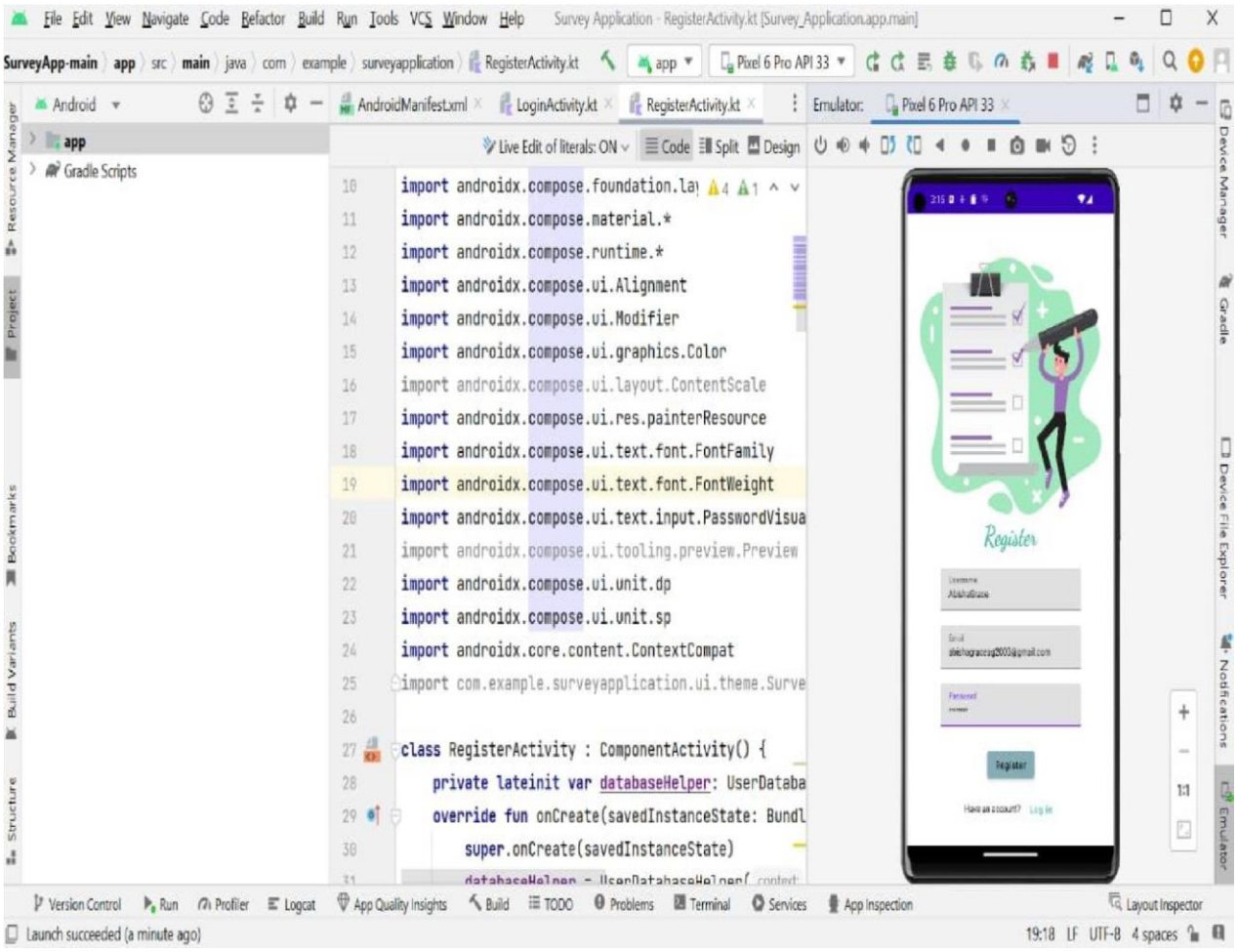
import android.os.Bundle import android.util.Log import androidx.activity.ComponentActivity import androidx.activity.compose.setContent import androidx.compose.foundation.Image import androidx.compose.foundation.layout.* import androidx.compose.foundation.lazy.LazyColumn import androidx.compose.foundation.lazy.LazyRow import androidx.compose.foundation.lazy.items import androidx.compose.material.MaterialTheme import

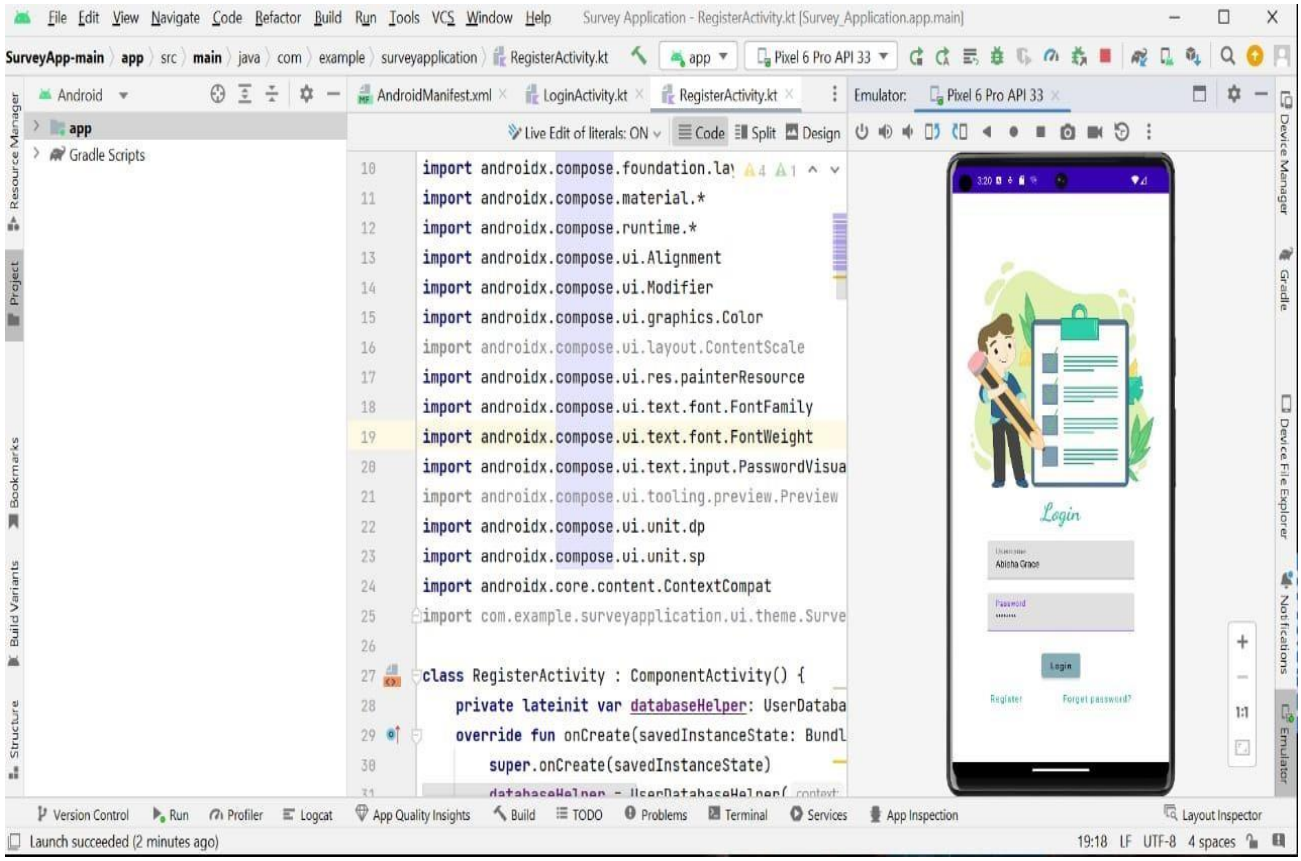
```
androidx.compose.material.Surface import
androidx.compose.material.Text import
androidx.compose.runtime.Composable import
androidx.compose.ui.Modifier import
androidx.compose.ui.graphics.Color import
androidx.compose.ui.layout.ContentScale import
androidx.compose.ui.res.painterResource import
androidx.compose.ui.tooling.preview.Preview 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}")
            }
        }
    }
  }
}
```

3. RESULT


File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
Survey Application - RegisterActivity.kt [Survey_Application.app.main]
SurveyApp-main app src main java com example surveyapplication RegisterActivity.kt
app
Pixel 6 Pro API 33
Android
AndroidManifest.xml
LoginActivity.kt
RegisterActivity.kt
Emulator: Pixel 6 Pro API 33
app
Gradle Scripts
Live Edit of literals: ON
Code
Split
Design
import androidx.compose.foundation.lay
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.text.input.PasswordVisua
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.Surve
class RegisterActivity : ComponentActivity() {
private lateinit var databaseHelper: UserDataba
override fun onCreate(savedInstanceState: Bundl
super.onCreate(savedInstanceState)
Register
Have an account? Log in
Version Control
Run
Profiler
Logcat
App Quality Insights
Build
TODO
Problems
Terminal
Services
App Inspection
Layout Inspector
Launch succeeded (a minute ago)
19:18 LF UTF-8 4 spaces


File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
Survey Application - RegisterActivity.kt [Survey_Application.app.main]
SurveyApp-main app src main java com example surveyapplication RegisterActivity.kt
app
Pixel 6 Pro API 33
Android
app
Gradle Scripts
AndroidManifest.xml
LoginActivity.kt
RegisterActivity.kt
Live Edit of literals: ON
Code
Split
Design
import androidx.compose.foundation.lay
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.text.input.PasswordVisua
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.Surve
class RegisterActivity : ComponentActivity() {
private lateinit var databaseHelper: UserDataba
override fun onCreate(savedInstanceState: Bundl
super.onCreate(savedInstanceState)
Emulator:
Pixel 6 Pro API 33
Login
Abisha Grace
Login
Register
Forget password?
Version Control
Run
Profiler
Logcat
App Quality Insights
Build
TODO
Problems
Terminal
Services
App Inspection
Layout Inspector
Launch succeeded (2 minutes ago)
19:18 LF UTF-8 4 spaces

Register
Username
AbishaGrace
Email
abishagraceag2003@gmail.com
Password
Register
Have an account? Log in

Survey on Diabetics
Name :
Abisha Grace
Age :
20
Mobile Number :
8870189877
Gender :
Male
Female
Other
Diabetics :
Diabetic
Not Diabetic
Submit

4. ADVANTAGES AND DISADVANTAGES

ADVANTAGES

i. Declarative UI - Compose provides a declarative way of building UI components, which makes it easier to write and maintain code. Developers can describe the UI component's state and behavior, and the system takes care of rendering it correctly.

ii. Simplified Layout - Compose reduces the need for XML layout files, which simplifies the layout process and makes it easier to understand the UI hierarchy.

iii. Improved Performance - Compose uses a reactive programming model that improves performance by reducing the need to update the entire UI for every change.

iv. Interoperability - Compose can be used with existing Android codebases, making it easy to integrate into existing projects.

v. Less Boilerplate Code - Compose reduces the amount of boilerplate code that developers need to write, which makes the code easier to read and maintain.

DISADVANTAGES

i. Learning Curve - Compose is a new technology, and developers need to invest time in learning the framework, which can be a significant disadvantage for some.

ii. Limited Documentation - Compose is still in its early stages, and documentation is not as comprehensive as some other Android frameworks.

iii. Third-party Libraries - Some third-party libraries may not be compatible with Compose, which may limit the developer's options.

iv. Compatibility - Compose is only compatible with Android devices running on Android
5.0 (API level 21) or higher, which may limit the reach of the application.

v. Lack of Tool Support - Some popular tools, such as Android Studio's Layout Editor, may not support Compose, which may make the development process more difficult for some developers.

5. APPLICATION

The Compose Input demonstration is a practical example of how to use Android Compose to build a text input form with validation. This demonstration can be used as a starting point for developers who need to build similar UI components in their Android applications.

The Compose Input demonstration can be used in a wide range of applications, such as:

Login and registration forms - A login or registration form requires user input, and validation is essential to ensure the entered information is correct.

Search bar - A search bar is a common UI component that requires user input, and validation is essential to ensure the entered query is valid.

Contact forms - A contact form requires users to enter personal information, and validation is essential to ensure the entered information is accurate.

Android Compose is a modern UI toolkit for building native Android apps using declarative programming techniques.
Compose provides a simple and intuitive way to create user interfaces by allowing developers to describe the UI as a function of the current state of the app. One common use case for Compose is handling user input and validating that input before using it in the app.

6. CONCLUSION

In conclusion, the Compose Input demonstration provides a practical example of how to use Android Compose to build a text

input form with validation. The demonstration highlights the benefits of using Compose, such as declarative UI, simplified layout, improved performance, and interoperability with existing Android codebases.

The Compose Input demonstration can be used as a starting point for developers who need to build similar UI components in their Android applications. The demonstration can be applied to various use cases, such as login and registration forms, search bars, contact forms, payment forms, and feedback forms. However, developers should also consider the disadvantages of using Compose, such as the learning curve, limited documentation, third-party library compatibility, compatibility with older Android devices, and lack of tool support.

Overall, the Compose Input demonstration shows that Android Compose is a promising technology for building efficient and effective user interfaces in Android applications. With Compose, developers can create UI components that are easy to understand, maintain, and extend, which can help them build better Android applications.

In conclusion, Android Compose provides a powerful way to handle user input and validation in native Android apps. By using Compose's declarative programming techniques, developers can create reactive UIs that respond to changes in the app's state. This example demonstrated how to create a simple form with text input and validation using Compose's OutlinedTextField

element and mutable state variables to track the current input and any validation errors. With Compose, handling user input and validation becomes easier and more efficient, resulting in a better user experience for Android app users.

7. FUTURE SCOPE

The future scope of Compose Input and Android Compose, in general, is promising. As Compose is a relatively new technology, it is constantly evolving and improving. Here are some potential future developments for Compose Input:

Improved documentation - As more developers start using Compose, the documentation will likely improve and become more comprehensive, making it easier for developers to learn and use the framework.

Third-party library compatibility - As Compose becomes more popular, more third-party libraries will likely become compatible with it, expanding the options available to developers.

Better tool support - As Compose gains popularity, it is likely that tool support will improve. Android Studio's Layout Editor may eventually support Compose, making it easier for developers to create UI components.

More built-in UI components - Compose is still relatively new, and more built-in UI components may be added in the future. This will make it easier for developers to build complex UIs without having to write as much custom code.

Increased adoption - As more developers adopt Compose, it is likely that it will become the standard way of building Android UI components. This will lead to more resources and support for the technology, making it easier for developers to use and improve upon.

Overall, the future of Compose Input and Android Compose is bright. As more developers start using the technology and contributing to its development, it will continue to evolve and improve, making it easier for developers to create efficient and effective user interfaces in Android applications.

The future scope of Compose input in Android apps is quite exciting. Here are some potential areas where Compose input can be further developed:

Custom input components: Compose's flexible architecture allows developers to create custom input components that can be tailored to specific app requirements. These custom input components can enhance the user experience by providing unique input methods for specific types of data.

Real-time input validation: Currently, the input validation in the example is triggered only when the user changes the text input. However, with Compose, it is possible to implement realtime input validation where the user can see if their input is valid or not as they type.

Input prefilling: Compose can be used to prefill input fields with existing data. This feature can be useful for apps that require users to input data multiple times, such as sign-up forms or address forms.

Multilingual support: Compose provides a simple way to handle text input and validation, making it easier to support multiple languages in an app. With Compose, developers can create input components that support different languages and input methods.

Input formatting: Compose can be used to format input in real-time as the user types. For example, an input field for a phone number can format the input as the user types to match the expected phone number format.

Overall, the future of Compose input in Android apps is promising, and developers can use its capabilities to provide better user experiences and make it easier for users to input and validate data in their apps.