Designing a Login Page & Signup Page Using Flutter

Presented by :

1. **Selvavelraj S**
2. **Gowtham T**
3. **Nanthakumar G**
4. **Selvin T**

OVERVIEW

01 Objectives	02 Technologies	03 Goals
04 Introduction	05 Working	06 Testing

Objective

The project aims to design a login and signup page in Flutter that allows users to securely authenticate and create accounts.

Additionally, the pages offer a user-friendly interface with responsive design and navigation between the login and signup screens.

Technologies

01 Android Studio

02 Flutter

03 Dart

04 Form Validation

GOALS

Growth

Building and Growing Your Portfolio:

Building and growing your portfolio involves high-quality projects that highlight your skills & expertise.
Start by including personal, freelance relevant to you field.Continuously add work that demonstrates growth& adaptability, focusing on problem-solving.
Tailor your portfolio to align with your career goals, emphasizing the skills,Present your work visually & clearly toexplain your role and achievements.

Introduction

What is Flutter?

Flutter is an open-source UI toolkit developed by Google for building natively compiled applications for mobile (iOS and Android),web, and desktop (Windows, macOS, Linux) platforms using a single codebase. During my internship at Marcello Tech Company, I extensively worked with Flutter to develop My First Project

Why Choose Flutter?

Cost-effective: Develop once, deploy everywhere. Fast Development: Hot reload and extensive tools speed up the process. Community Support: Backed by Google with an active developers

Installing flutter sdk:

To install the Flutter SDK on your system, follow these steps:

1. Download the Flutter SDK:

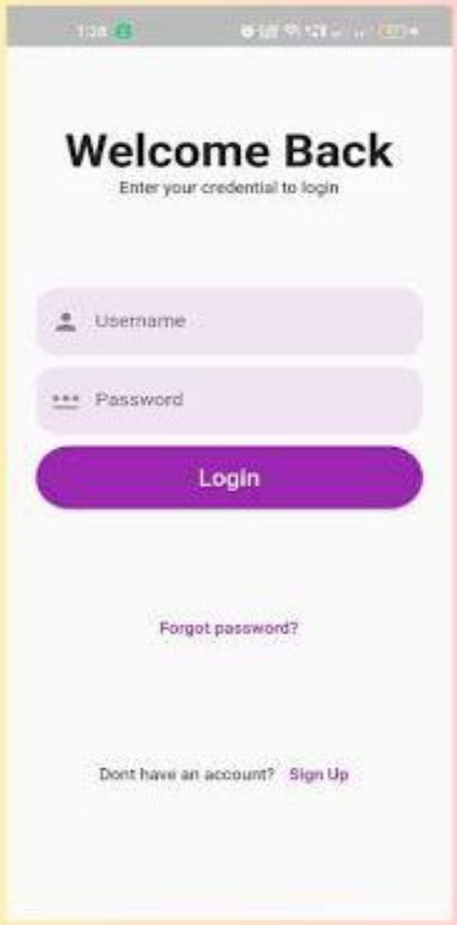
Go to the Flutter download page and select the appropriateinstallation bundle for your operating system (Windows, macOS, or Linux). Download the stable version of the Flutter SDK.

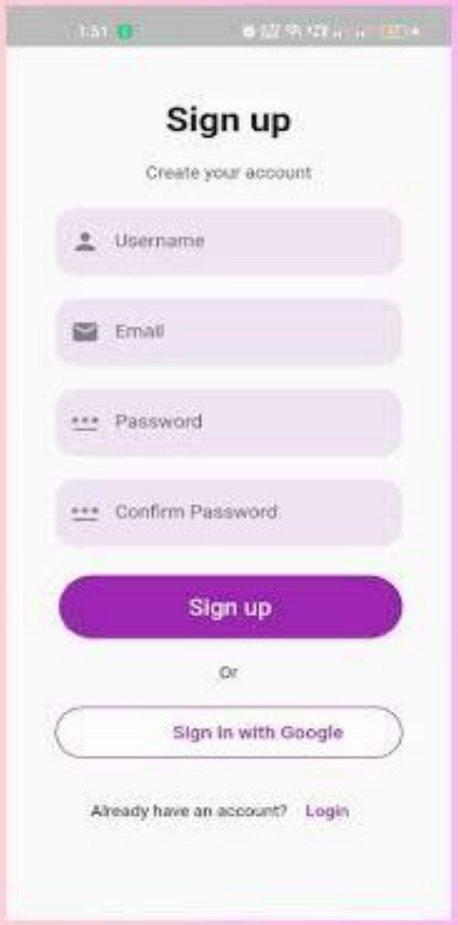
2. Extract the Archive

Windows: Extract the .zip file to a desired location (e.g.,C:\src\flutter).macOS/Linux: Extract the .tar.xz file to a suitable location.

3. Add Flutter to the Path Windows:

Open the Start search, and type "Environment Variables". Click Edit the system environment variables. In the System Properties window, click on the EnvironmentVariables button.


Welcome Back
Enter your credential to login
Username
Password
Login
Forgot password?
Dont have an account? Sign Up


Sign up
Create your account
Username
Email
Password
Confirm Password
Sign up
Or
Sign In with Google
Already have an account? Login

Dart Programming Language:

Variables:

A variable is used to store a value that can be referenced and manipulated in your program. Here, var is used to declare the variable, and Dart infers the type based on the assigned value.

Numbers:

Int: Used to represent whole numbers.

Double: Used to represent floating-point numbers (decimals).

String - Used to represent a sequence of characters.

Boolean - Represents true or false.

List - Used to store an ordered collection of items.

Map - Used to store key-value pairs.

Set - Used to store a collection of unique items.

Dynamic - Can hold any type of value (similar to Object in other languages).

Null - Represents the absence of a value.

1. if-else Statements:

The if-else statement allows you to execute a block of code based on a condition

2. Switch-Case:

The switch statement is used when you have multiple conditions based on the same variable or expression. It's a cleaner alternative to multiple if-else-if statements.

3. Loops

Dart supports several loop structures:

- For
- While
- Do-while.

Functions:

A function is a block of code that performs a task and can return a value (or not). Functions are typically declared outside of a class.

Methods:

A method is essentially a function that is defined within a class. Methods operate on instances (objects) of that class or on the class itself (for static methods).

Object-Oriented Programming With Dart:

Object-Oriented Programming (OOP) is a programming paradigm that uses objects and classes to organize code. Dart, the language used in Flutter, supports OOP principles, including classes, objects, inheritance, polymorphism, encapsulation, andabstraction.

Exception Handling with Dart:

In Dart, exception handling is used to catch and manage errors in a program. Dart provides a try-catch mechanism for handling exceptions that might occur during program execution. This helps ensure that the program can handle errors gracefully without crashing.

Building The User Interface:

Basic Widgets:

1.Text

The Text widget is used to display a string of text on the screen

2.Buttons

Flutter provides several types of buttons, such as ElevatedButton, TextButton, OutlinedButton, and IconButton.

3.Images

Flutter provides several ways to display images, such as Image, Image.network, Image.asset.

Layout Widgets:

1.Column

The Column widget is used to arrange children widgetsvertically (from top to bottom).

2.Row

The Row widget is used to arrange children widgets horizontally (from left to right).

3.Stack

The Stack widget allows you to place widgets on top of each other,The flexibility to position them specifically within the stack.

4.ListView

The ListView widget is used to create a scrollable list ofwidgets

Navigation and Routing:

1.Basic Navigation

The Navigator maintains a stack of pages and handles transitions between them.

2.Named Routes

Named routes are a convenient way to manage navigation, especially for larger applications.

3.Pop and Push

Togetheryou might want to replace the current screen with a new screen,You can use Navigator.pushReplacement() to achieve this.

4.Navigating with Custom Transitions

You can create custom page transitions when navigatingbetween screens using PageRouteBuilder.

5.Bottom Navigation and Tab Bar Navigation

For apps with multiple sections or tabs, you can useBottomNavigationBar or TabBar for navigation.

Working with APIs and Data

Making HTTP Requests in Flutter

In Flutter, making HTTP requests is commonly done using the http package, which allows you to send requests like GET, POST,
PUT, DELETE to web services or APIs.Parsing JSON Data It’s a common task when dealing with data from APIs. We convert JSON response into a Dart object using the dart:convert package, which provides the json.decode() method.

Working with REST APIs :

It's done using HTTP requests to send & receive data. You can use various methods like GET, POST, PUT, and DELETE to
interact with API & typically work with JSON data. Handling Errors And Timeouts: Handling errors and timeouts in Flutter when working with REST APIs is crucial for providing a robust and user-friendly experience.

Testing and Debugging

Writing Unit & Widgets Tests:

Writing Unit and Widget tests in Flutter is an essential part of ensuring your application functions as expected. Below is a guide on how to write both types of tests in Flutter.

Debugging in Flutter:

Debugging is a crucial part of software development, and Flutter provides various tools and techniques to help you diagnose and fix issues in your app. Here's a guide on how to effectively debug your Flutter app.

Test Driven Development with Flutter:

Test-Driven Development (TDD) is a software development methodology in which you write tests for a functionality before writing the actual code. Flutter provides a solid testing framework, making it easy to adopt TDD.

Unit Testing:

Unit tests are used to test a single function, method, or class. These tests are fast and should ideally not depend on any UI, external libraries, or network calls.

Widget Testing:

Widget tests are used to test individual widgets in isolation. These tests help ensure that widgets look and behave as expected. Widget tests are typically more complex than unit tests, but they still don’t require a fully functional app.

Integration Testing in Flutter

Integration tests are used to test an entire app or large parts of it, ensuring that everything works together as expected. These tests can cover multiple widgets, interactions, network calls, and more.

Conclusion

Writing Unit & Widgets Tests:

Writing Unit and Widget tests in Flutter is an essential part of ensuring your application functions as expected. Below is a guide on how to write both types of tests in Flutter.

Debugging in Flutter:

Debugging is a crucial part of software development, and Flutter provides various tools and techniques to help you diagnose and fix issues in your app. Here's a guide on how to effectively debug your Flutter app.

Test Driven Development with Flutter:

Test-Driven Development (TDD) is a software development methodology in which you write tests for a functionality before writing the actual code. Flutter provides a solid testing framework, making it easy to adopt TDD.

Unit Testing:

Unit tests are used to test a single function, method, or class. These tests are fast and should ideally not depend on any UI, external libraries, or network calls.

Widget Testing:

Widget tests are used to test individual widgets in isolation. These tests help ensure that widgets look and behave as expected. Widget tests are typically more complex than unit tests, but they still don’t require a fully functional app.

Integration Testing in Flutter

Integration tests are used to test an entire app or large parts of it, ensuring that everything works together as expected. These tests can cover multiple widgets, interactions, network calls, and more.

Conclusion

Wrapping Up Your Flutter Journey:

Wrapping up your Flutter journey involves consolidating your knowledge and ensuring you're equipped for real-world development. Review key concepts like widgets, state management, animations, and platform integration. Build a polished, portfolio-worthy project to showcase your skills. Lastly, embrace continuous learning, as Flutter evolves rapidly, and adapt your skills to meet the demands of modern app development.

Future Development :

Preparing for Real-World App Development:

Preparing for real-world app development involves gaining practical skills and experience. Start by mastering core programming concepts, frameworks, and tools relevant to your chosen platform. Work on small projects to understand app architecture, state management, and APIs. Learn best practices for responsive design, performance optimization, and secure coding. Finally, stay updated with industry trends and continuously improve your skills to adapt to evolving technologies and user needs.

Building and Growing Your Portfolio:

Building and growing your portfolio involves high-quality projects that highlight your skills & expertise. Start by including personal, freelance relevant to your field.Continuously add work that demonstrates growth& adaptability, focusing on problem-solving. Tailor your portfolio to align with your career goals, emphasizing the skills,Present your work visually & clearly to explain your role and achievements.

THANK YOU !!!