

Lab: Create a Layout in Flutter with Navigation

Estimated time: **30 minutes**



Welcome to this hands-on lab where you will learn how to create a layout in Flutter and implement navigation between different screens. This lab will help you understand how to use various Flutter widgets and manage navigation within your Flutter application.

Objectives

After completing this lab, you will be able to:

- Build a basic layout in Flutter using widgets like `Column`, `Row`, `Container`, and `Text`
- Implement navigation between multiple screens using `Navigator` and `Routes`
- Use common Flutter widgets to create a visually appealing and responsive layout
- Understand and manage state within a Flutter app
- Enhance layouts with interactive elements such as text fields and form validation

About Cloud IDE

Running the lab

This lab is designed to be completed using a Cloud IDE environment. You will be writing and executing Dart code directly within the IDE.

About Skills Network Cloud IDE

Skills Network Cloud IDE (based on Theia and Docker) provides an environment for hands-on labs for course and project-related labs. It is an open-source Integrated Development Environment (IDE).

Important notice about this lab environment

Please be aware that sessions for this lab environment are not persisted. Every time you connect to this lab, a new environment is created for you. Any data you may have saved in the earlier session would get lost. Plan to complete these labs in a single session to avoid losing your data.

Key terms

- **Widget:** The basic building block of the Flutter app's user interface. Everything in Flutter is a widget.
- **Layout:** How the visual elements are arranged in an app.
- **Navigator:** A widget that manages a stack of route objects and provides methods for navigating between routes.
- **Route:** An abstraction for a screen or page in a Flutter app.
- **Form:** A group of input fields with validation logic.

Prerequisites

Before starting this lab, ensure you have the following setup in your Cloud IDE:

1. Create a new Flutter project:

```
flutter create flutter_layout_lab
cd flutter_layout_lab
```

2. Open the `lib/main.dart` file in your project.

[Open `main.dart` in IDE](#)

Step 1: Build a basic layout

Remove all code from `main.dart`. You will start from scratch.

1. Import the material.dart package:

```
import 'package:flutter/material.dart';
```

- import statement is used to include the Flutter material library that provides built-in widgets to design the user interface.

2. Define the main entry point function:

```
import 'package:flutter/material.dart';
void main() {
  runApp(const MyApp());
}
```

- The main function is the entry point for every Dart application.
- runApp takes the provided widget (MyApp) and makes it the root of the widget tree.

3. Create a stateless widget named MyApp:

```
import 'package:flutter/material.dart';
void main() {
  runApp(const MyApp());
}
class MyApp extends StatelessWidget {
  const MyApp({super.key});
}
```

- MyApp is a custom widget that extends StatelessWidget, which is used when the widget doesn't manage any state of its own.
- const constructor is used here to indicate that MyApp is immutable.

4. Override the build method to define the app structure:

```
import 'package:flutter/material.dart';
void main() {
  runApp(const MyApp());
}
class MyApp extends StatelessWidget {
  const MyApp({super.key});
  @override
  Widget build(BuildContext context) {
    // Placeholder for MaterialApp properties
  }
}
```

- build method describes how to display the widget in the app.
- The context parameter provides information about the location of this widget in the widget tree.

5. Add the MaterialApp widget to the build method to start building the app:

```
import 'package:flutter/material.dart';
void main() {
  runApp(const MyApp());
}
class MyApp extends StatelessWidget {
  const MyApp({super.key});
```

```

@override
Widget build(BuildContext context) {
  return MaterialApp(
    title: 'Flutter Layout Lab',
    // Placeholder for home property
  );
}
}

```

- MaterialApp is a convenience widget that wraps several commonly used widgets to design a material app.

6. Define the home property with a Scaffold widget:

```

import 'package:flutter/material.dart';
void main() {
  runApp(const MyApp());
}
class MyApp extends StatelessWidget {
  const MyApp({super.key});
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Layout Lab',
      home: Scaffold(
        // Placeholder for Scaffold properties
      ),
    );
  }
}

```

- Scaffold provides a framework that implements the basic material design layout structure of the visual interface.

7. Add an AppBar widget to the Scaffold:

```

import 'package:flutter/material.dart';
void main() {
  runApp(const MyApp());
}
class MyApp extends StatelessWidget {
  const MyApp({super.key});
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Layout Lab',
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Flutter Layout Example'),
        ),
        // Placeholder for body
      ),
    );
  }
}

```

- AppBar is a material design app bar that can be used to give the app a consistent look and feel at the top of the screen.
- title is a property to add text to the app bar.

8. Define the body of the Scaffold using a Column widget:

```

import 'package:flutter/material.dart';
void main() {
  runApp(const MyApp());
}
class MyApp extends StatelessWidget {
  const MyApp({super.key});
  @override

```

```
Widget build(BuildContext context) {  
  return MaterialApp(  
    title: 'Flutter Layout Lab',  
    home: Scaffold(  
      appBar: AppBar(  
        title: const Text('Flutter Layout Example'),  
      ),  
      body: const Column(  
        children: const <Widget>[  
          Text('Welcome to Flutter!'),  
          Text('Building a layout is easy.'),  
        ],  
      ),  
    ),  
  );  
}
```

- Column is a layout widget that arranges its children in a vertical direction.
- children property takes a list of widgets to display.
- Text widgets display text on the screen.

Step 2: Run the app

Flutter provides `hot reload` that makes it very easy to test the code as you are developing it. That functionality however does not work in the Cloud IDE. We have developed a `hot_reload.sh` script that essentially does the same thing. Please follow these steps to run the flutter application:

1. Change to the home directory

```
cd /home/project
```

2. Set the `PROJECT_DIR` variable. This should be set to the `lib` directory of your flutter app folder.

```
export PROJECT_DIR=/home/project/flutter_layout_lab/lib
```

3. Get the script and save to the `/home/project` folder.

```
wget -O /home/project/hot_reload.sh https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/S3cw83zxbxBhfixSSMK1Uw/hot-reload.sh
```

4. Make the script executable by using the `chmod` command.

```
chmod +x ./hot_reload.sh
```

5. Run the command.

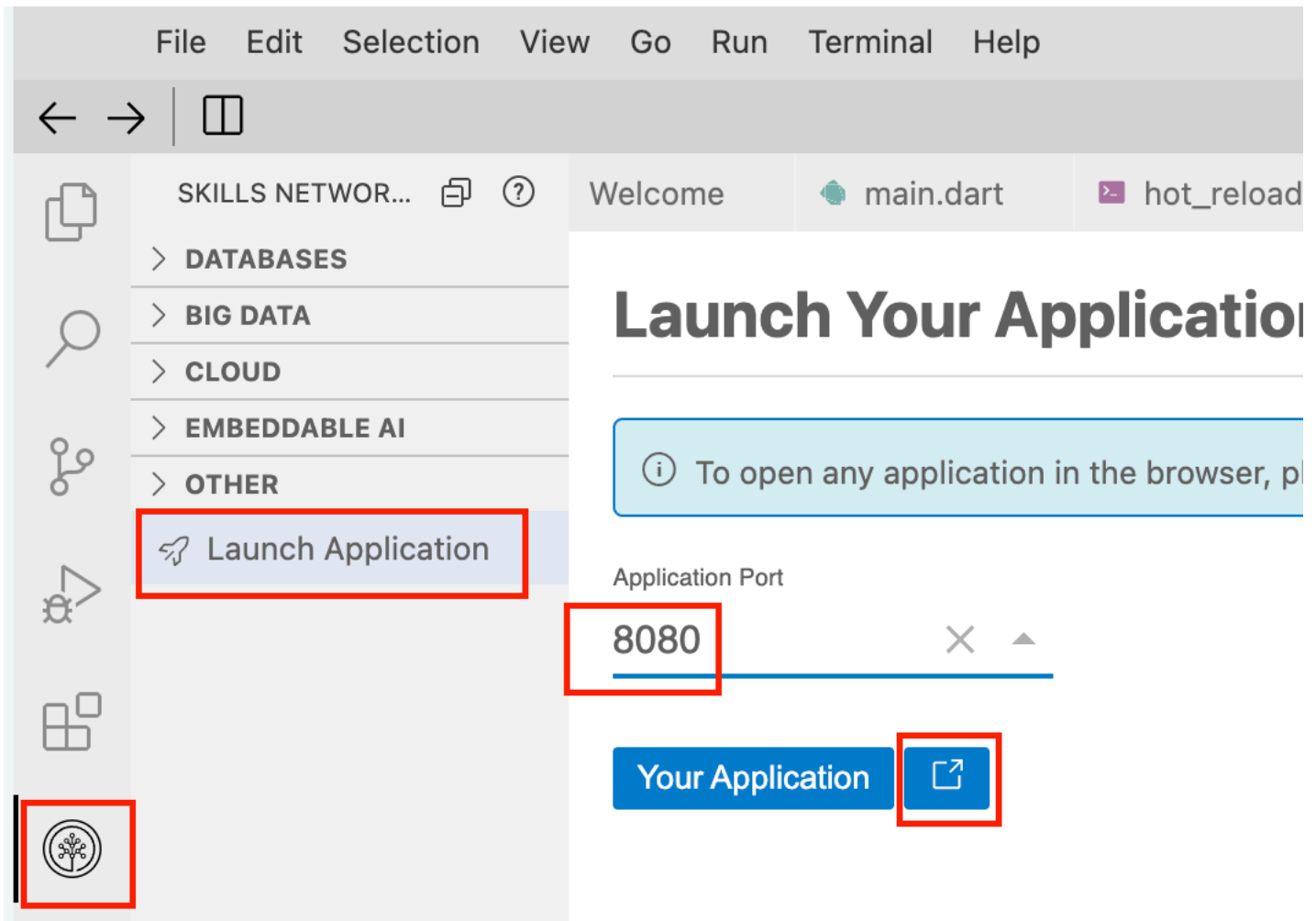
```
./hot_reload.sh
```

You should see output like this:

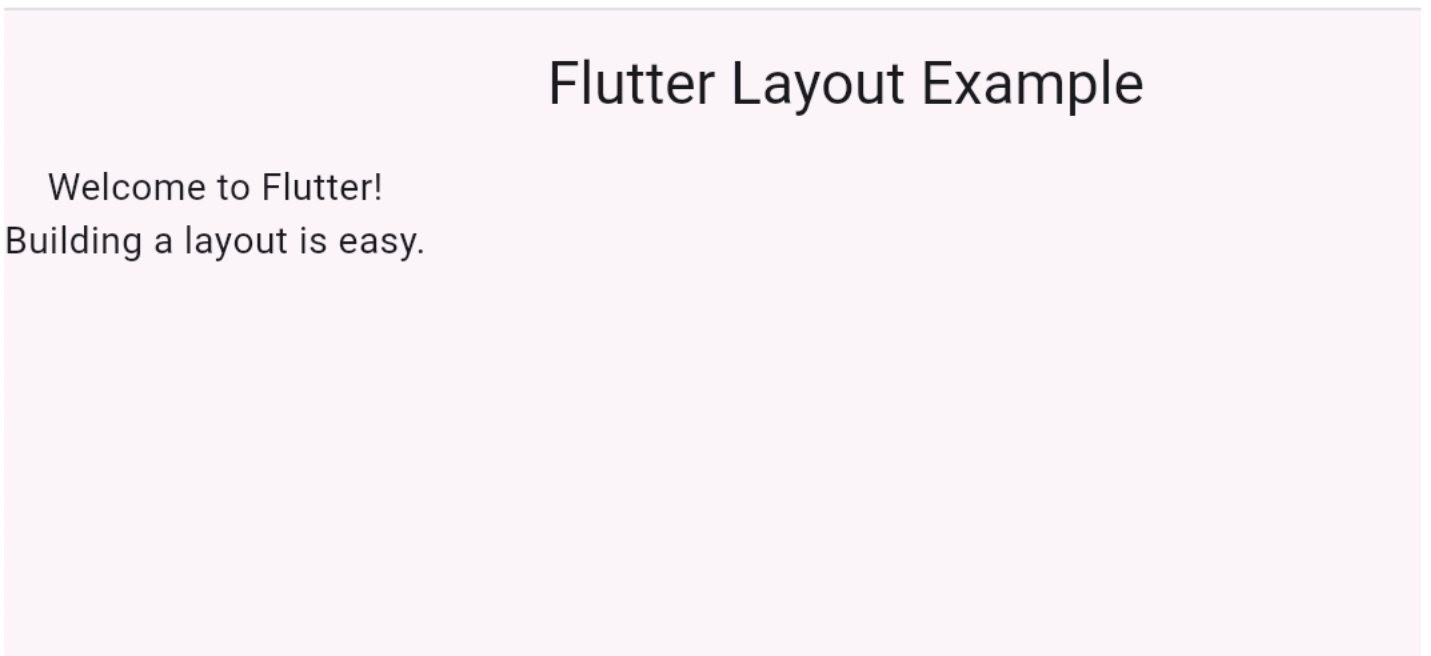
```
theia@theia-captainfed01:/home/project$ ./hot_reload.sh
Using PROJECT_DIR: /home/project/flutter_layout_lab/lib
inotify-tools is already installed.
lsuf is already installed.
Starting Flutter web server...
Port 8080 is free.
Compiling lib/main.dart for the Web... 984ms
✓ Built build/web
Flutter server started with PID 17675
Setting up watches. Beware: since -r was given, this may take a while!
Watches established.
Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...
```

6. Launch the application

- open the Skills Network Toolbox in the left bar of the Cloud IDE.
- click on Launch Application button
- enter 8080 as the port
- select the launch in new window to open the application in a new tab



You should see your basic application now. Notice that it has the content but no layout. We will add layout in the next step:



Step 3: Enhance the layout with more widgets

1. Update the `main.dart` file to add a `Row` widget inside the `Column`:

```
import 'package:flutter/material.dart';
void main() {
  runApp(const MyApp());
}
```

```

class MyApp extends StatelessWidget {
  const MyApp({super.key});
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Layout Lab',
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Flutter Layout Example'),
        ),
        body: const Column(
          children: <Widget>[
            Row(
              // Placeholder for Row properties and children
            ),
            Text('Welcome to Flutter!'),
            Text('Building a layout is easy.'),
          ],
        ),
      ),
    );
  }
}

```

- Row is a layout widget that arranges its children in a horizontal direction.

2. Add mainAxisAlignment to control alignment in the Row:

```

import 'package:flutter/material.dart';
void main() {
  runApp(const MyApp());
}
class MyApp extends StatelessWidget {
  const MyApp({super.key});
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Layout Lab',
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Flutter Layout Example'),
        ),
        body: const Column(
          children: <Widget>[
            Row(
              mainAxisAlignment: MainAxisAlignment.spaceEvenly,
              // Placeholder for children
            ),
            Text('Welcome to Flutter!'),
            Text('Building a layout is easy.'),
          ],
        ),
      ),
    );
  }
}

```

- mainAxisAlignment defines how the children should be placed along the main axis (horizontal).

3. Add children widgets to the Row:

```

import 'package:flutter/material.dart';
void main() {
  runApp(const MyApp());
}
class MyApp extends StatelessWidget {
  const MyApp({super.key});
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Layout Lab',
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Flutter Layout Example'),
        ),
        body: const Column(

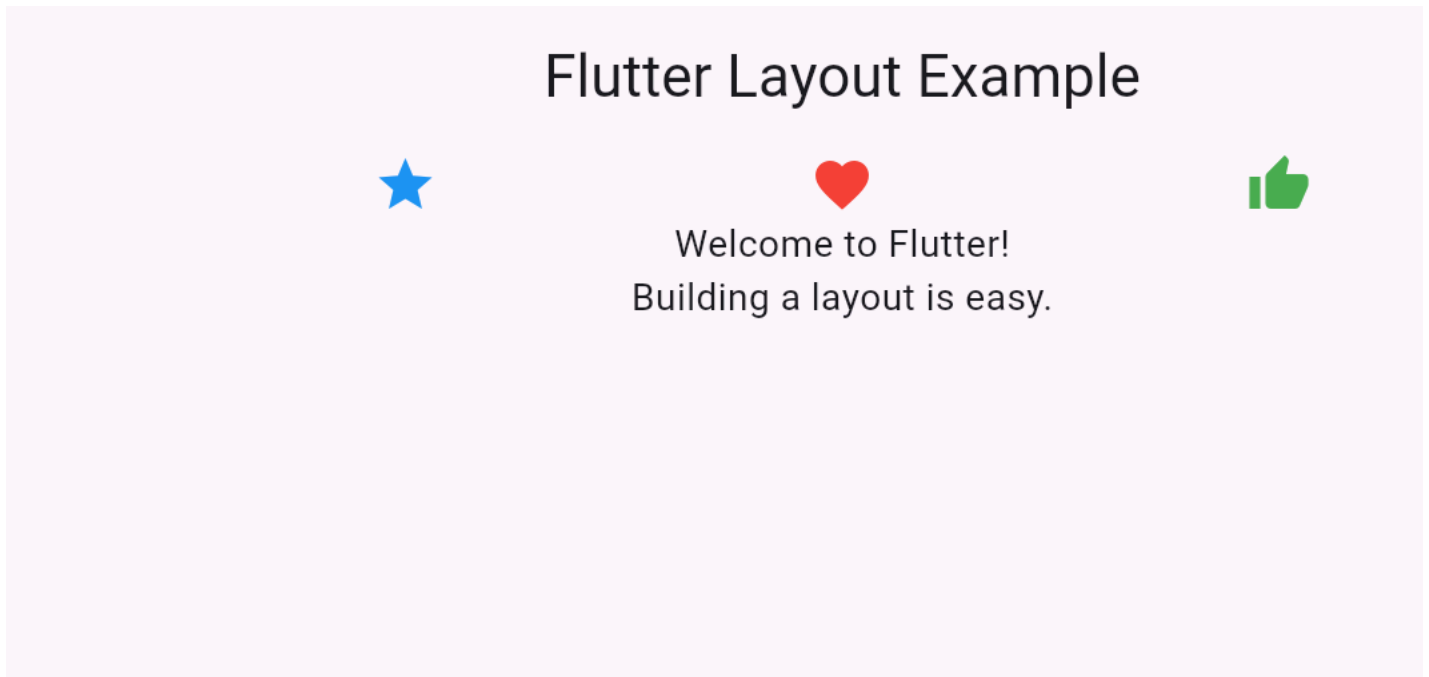
```

```

children: <Widget>[
  Row(
    mainAxisAlignment: MainAxisAlignment.spaceEvenly,
    children: const<Widget>[
      Icon(Icons.star, color: Colors.blue),
      Icon(Icons.favorite, color: Colors.red),
      Icon(Icons.thumb_up, color: Colors.green),
    ],
  ),
  Text('Welcome to Flutter!'),
  Text('Building a layout is easy.'),
],
),
),
);
}
}

```

- Icon widget is used to display icons.
 - Icons.star, Icons.favorite, and Icons.thumb_up are predefined icons in Flutter.
4. If you already have the app running with the hot_reload.sh script, simply open the tab and reload to see the code changes. The app will take up to 4 minutes to reload in the other terminal. If you closed the script for some reason, you will have to run it again as explained earlier.



Step 4: Advanced layout with interactions, logger, and forms

1. Update the pubspec.yaml file to include the logger package:

```

dependencies:
  flutter:
    sdk: flutter
  logger: ^1.1.0

```

2. Install the logger package by running the following command in your terminal:

```
cd /home/project/flutter_layout_lab && flutter pub get
```


3. Update the main.dart file incrementally to use the Logger:

3.1 Import the logger package:

```
import 'package:flutter/material.dart';
import 'package:logger/logger.dart'; // Import the logger package
```

- The logger package provides a simple logging utility to print log messages.

3.2 Initialize the Logger in your MyForm state:

```
import 'package:flutter/material.dart';
import 'package:logger/logger.dart'; // Import the logger package
void main() {
  runApp(const MyApp());
}
class MyApp extends StatelessWidget {
  const MyApp({super.key});
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Layout Lab',
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Flutter Layout Example'),
        ),
        body: MyForm(),
      ),
    );
  }
}
class MyForm extends StatefulWidget {
  @override
  _MyFormState createState() => _MyFormState();
}
class _MyFormState extends State<MyForm> {
  final _formKey = GlobalKey<FormState>();
  final Logger _logger = Logger(); // Initialize the logger
```

- Logger is initialized to create log messages for debugging.

3.3 Use the logger to log form data when the button is pressed:

```
import 'package:flutter/material.dart';
import 'package:logger/logger.dart'; // Import the logger package
void main() {
  runApp(const MyApp());
}
class MyApp extends StatelessWidget {
  const MyApp({super.key});
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Layout Lab',
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Flutter Layout Example'),
        ),
        body: MyForm(),
      ),
    );
  }
}
class MyForm extends StatefulWidget {
  @override
```

```

    _MyFormState createState() => _MyFormState();
  }
  class _MyFormState extends State<MyForm> {
    final _formKey = GlobalKey<FormState>();
    final Logger _logger = Logger(); // Initialize the logger
    @override
    Widget build(BuildContext context) {
      return Form(
        key: _formKey,
        child: Column(
          children: <Widget>[
            const Row(
              mainAxisAlignment: MainAxisAlignment.spaceEvenly,
              children: const<Widget>[
                Icon(Icons.star, color: Colors.blue),
                Icon(Icons.favorite, color: Colors.red),
                Icon(Icons.thumb_up, color: Colors.green),
              ],
            ),
            const Text('Welcome to Flutter!'),
            const Text('Building a layout is easy.'),
            TextFormField(
              decoration: const InputDecoration(labelText: 'Enter your name'),
              validator: (value) {
                if (value == null || value.isEmpty) {
                  return 'Please enter some text';
                }
                return null;
              },
            ),
            ElevatedButton(
              onPressed: () {
                if (_formKey.currentState!.validate()) {
                  _logger.i('Form is valid. User input is logged.');// Log the user input
                }
              },
              child: const Text('Submit'),
            ),
          ],
        ),
      );
    }
  }
}

```

- Logger is used to print a log message when the form is valid and the button is pressed.

3.4 Since we using a special script to run the application, the logger is not able to log back to the terminal. Instead let's add a function to show an alert box when the submit button is selected.

```

void _showAlertDialog(BuildContext context, String title, String message) {
  showDialog(
    context: context,
    builder: (BuildContext context) {
      return AlertDialog(
        title: Text(title),
        content: Text(message),
        actions: [
          TextButton(
            child: const Text('OK'),
            onPressed: () {
              Navigator.of(context).pop(); // Close the dialog
            },
          ),
        ],
      );
    },
  );
}

```

Now, call this method when the button is clicked

```

ElevatedButton(
  onPressed: () {
    if (_formKey.currentState!.validate()) {
      _logger.i('Form is valid. User input is logged.');// Log the user input
      _showAlertDialog(context, 'Form Submitted', 'User input is logged.');// Call the alert dialog
    }
  },
  child: const Text('Submit'),
)

```

```
    },  
  },  
  child: const Text('Submit'),  
),
```

Step 5: Run the app

If you already have the app running from before, you don't have to do anything. Since we are not using the native Flutter hot reload, this step might take some time (up to 5 minutes). You will see a message `Compiling lib/main.dart for the Web...` and the app is ready once this finished.




If you quit that process for some reason or get an error message, run the script again from the `/home/project` folder.

```
cd /home/project && ./hot_reload.sh
```

Once the app starts, you can refresh the page if the tab is already open, or follow the instructions to launch application on port 8080 as before.

You will see a form on the screen asking for your name:

Flutter Layout Example



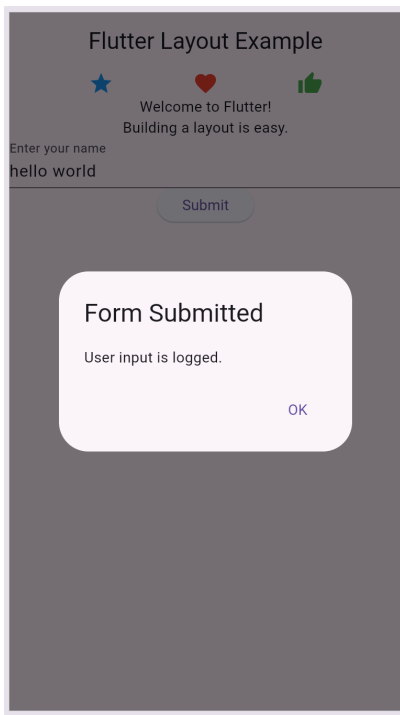
Welcome to Flutter!
Building a layout is easy.

Enter your name

hello world

Submit

When you enter you name and hit submit, you should see the alert dialog is triggered.



Complete Code

Complete code after this step:

```
import 'package:flutter/material.dart';
import 'package:logger/logger.dart'; // Import the logger package
void main() {
  runApp(const MyApp());
}
class MyApp extends StatelessWidget {
  const MyApp({super.key});
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Layout Lab',
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Flutter Layout Example'),
        ),
        body: MyForm(),
      ),
    );
  }
}
class MyForm extends StatefulWidget {
  @override
  _MyFormState createState() => _MyFormState();
}
class _MyFormState extends State<MyForm> {
  final _formKey = GlobalKey<FormState>();
  final Logger _logger = Logger(); // Initialize the logger
  @override
  Widget build(BuildContext context) {
    return Form(
      key: _formKey,
      child: Column(
        children: <Widget>[
          const Row(
            mainAxisAlignment: MainAxisAlignment.spaceEvenly,
            children: const<Widget>[
              Icon(Icons.star, color: Colors.blue),
              Icon(Icons.favorite, color: Colors.red),
              Icon(Icons.thumb_up, color: Colors.green),
            ],
          ),
          const Text('Welcome to Flutter!'),
          const Text('Building a layout is easy.'),
          TextFormField(
            decoration: const InputDecoration(labelText: 'Enter your name'),
            validator: (value) {
              if (value == null || value.isEmpty) {
                return 'Please enter some text';
              }
              return null;
            },
          ),
        ],
      ),
    ),
    ElevatedButton(
      onPressed: () {
```

```

        if (_formKey.currentState!.validate()) {
          _logger.i('Form is valid. User input is logged.');// Log the user input
          _showAlertDialog(context, 'Form Submitted', 'User input is logged.');//
        }
      },
      child: const Text('Submit'),
    ),
  ],
),
);
}

void _showAlertDialog(BuildContext context, String title, String message) {
  showDialog(
    context: context,
    builder: (BuildContext context) {
      return AlertDialog(
        title: Text(title),
        content: Text(message),
        actions: [
          TextButton(
            child: const Text('OK'),
            onPressed: () {
              Navigator.of(context).pop(); // Close the dialog
            },
          ),
        ],
      );
    },
  );
}
}

```

Conclusion and next steps

Congratulations on completing this lab! You have now practiced using various Flutter widgets and learned how to implement a layout, navigate between screens, and handle user input with form validation.

Now that you've completed the layout creation and interaction lab in Flutter, here are some suggested next steps to further enhance your skills:

- Customize the visual design of your Flutter app by creating themes for colors, typography, and shapes using ThemeData.
- Example: Create a light and dark mode theme for the application and switch between them based on user preference.
- Explore Flutter's animation capabilities to add interactive animations and transitions between screens, or animate widget properties for enhanced user experience.
- Learn how to test your Flutter apps using the built-in testing framework for unit tests, widget tests, and integration tests to ensure app reliability.

Author(s)

Skills Network

© IBM Corporation. All rights reserved.