Lab: Create a Basic Flutter App with Navigation

Estimated time: **30 minutes**

Welcome to this hands-on lab where you will learn how to create a user registration form in Flutter and implement navigation between different screens to display the entered data. 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 a user registration form with various input fields.
- Navigate between multiple screens using `Navigator` and `Routes`.
- Transfer and display data across different screens.
- Use common Flutter widgets to create a visually appealing and responsive layout.

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 IDE (Integrated Development Environment).

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_navigation_lab
cd flutter_navigation_lab
```

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

Remove all content from this file. You will write this code from scratch.

Step 1: Set up basic navigation

In this first step, we will create a form on the main screen to capture user registration information.

1. Import the `material.dart` package:

```
import 'package:flutter/material.dart';
//import 'second_screen.dart'; // Import the new file
```

- `import` statement is used to include the Flutter material library that provides built-in widgets to design the user interface.
- `second_screen.dart` is imported to use the `SecondScreen` widget from the new file. It is commented out right now as we haven't created this file yet.

2. Define the main entry point function:

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

```
//import 'second_screen.dart';
void main() {
  runApp(const MyApp());
}
class MyApp extends StatelessWidget {
  const MyApp({super.key});
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Navigation Lab',
      initialRoute: '/',
      routes: {
        '/': (context) => const HomeScreen(),
        //'/second': (context) => const SecondScreen(),
      },
    );
  }
}
```

- `MaterialApp` is a convenience widget that wraps several commonly used widgets to design a material app.
- `initialRoute` defines the default route for the app.
- `routes` is a map of route names to the corresponding widget builders. The route to the second screen is commented out as we haven't created that screen yet.

6. Create the `HomeScreen` widget:

```
import 'package:flutter/material.dart';
//import 'second_screen.dart';
void main() {
  runApp(const MyApp());
}
class MyApp extends StatelessWidget {
  const MyApp({super.key});
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Navigation Lab',
      initialRoute: '/',
      routes: {
        '/': (context) => const HomeScreen(),
        //'/second': (context) => const SecondScreen(),
      },
    );
  }
}
class HomeScreen extends StatefulWidget {
  const HomeScreen({super.key});
  @override
  _HomeScreenState createState() => _HomeScreenState();
}
class _HomeScreenState extends State<HomeScreen> {
  final _formKey = GlobalKey<FormState>();
  final TextEditingController _nameController = TextEditingController();
  final TextEditingController _ageController = TextEditingController();
  DateTime _selectedDate = DateTime.now();
  String _selectedGender = 'Male';
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('User Registration'),
      ),
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Form(
          key: _formKey,
          child: Column(
            children: [
              TextFormField(
                controller: _nameController,
                decoration: const InputDecoration(labelText: 'Name'),
                validator: (value) {
                  if (value == null || value.isEmpty) {
                    return 'Please enter your name';
                  }
                  return null;
                },
              ),
              TextFormField(
                controller: _ageController,
                decoration: const InputDecoration(labelText: 'Age'),
                keyboardType: TextInputType.number,
                validator: (value) {
                  if (value == null || value.isEmpty) {
                    return 'Please enter your age';
                  }
                  return null;
                },
```

```
                ),
                ListTile(
                  title: const Text('Date of Birth'),
                  subtitle: Text('${_selectedDate.toLocal()}'.split(' ')[0]),
                  trailing: const Icon(Icons.calendar_today),
                  onTap: () async {
                    final DateTime? picked = await showDatePicker(
                      context: context,
                      initialDate: _selectedDate,
                      firstDate: DateTime(1900),
                      lastDate: DateTime.now(),
                    );
                    if (picked != null && picked != _selectedDate) {
                      setState(() {
                        _selectedDate = picked;
                      });
                    }
                  },
                ),
                DropdownButtonFormField<String>(
                  value: _selectedGender,
                  decoration: const InputDecoration(labelText: 'Gender'),
                  items: ['Male', 'Female', 'Other']
                      .map((String value) => DropdownMenuItem<String>(
                            value: value,
                            child: Text(value),
                          ))
                      .toList(),
                  onChanged: (String? newValue) {
                    setState(() {
                      _selectedGender = newValue!;
                    });
                  },
                ),
                const SizedBox(height: 20),
                ElevatedButton(
                  onPressed: () {
                    if (_formKey.currentState!.validate()) {
                      Navigator.pushNamed(
                        context,
                        '/second',
                        arguments: {
                          'name': _nameController.text,
                          'age': _ageController.text,
                          'dateOfBirth': _selectedDate,
                          'gender': _selectedGender,
                        },
                      );
                    }
                  },
                  child: const Text('Register'),
                ),
              ],
            ),
          ),
        ),
      ),
    );
  }
}
```

- `HomeScreen` contains the registration form with various input fields and a button to navigate to the second screen.
- When the form is submitted, the data is passed to the `SecondScreen`.

Step 2: Run the app

Flutter provides `hot reload` that makes it very easy to test the code as you are developing it. That funcationality 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 setps 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_navigation_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.

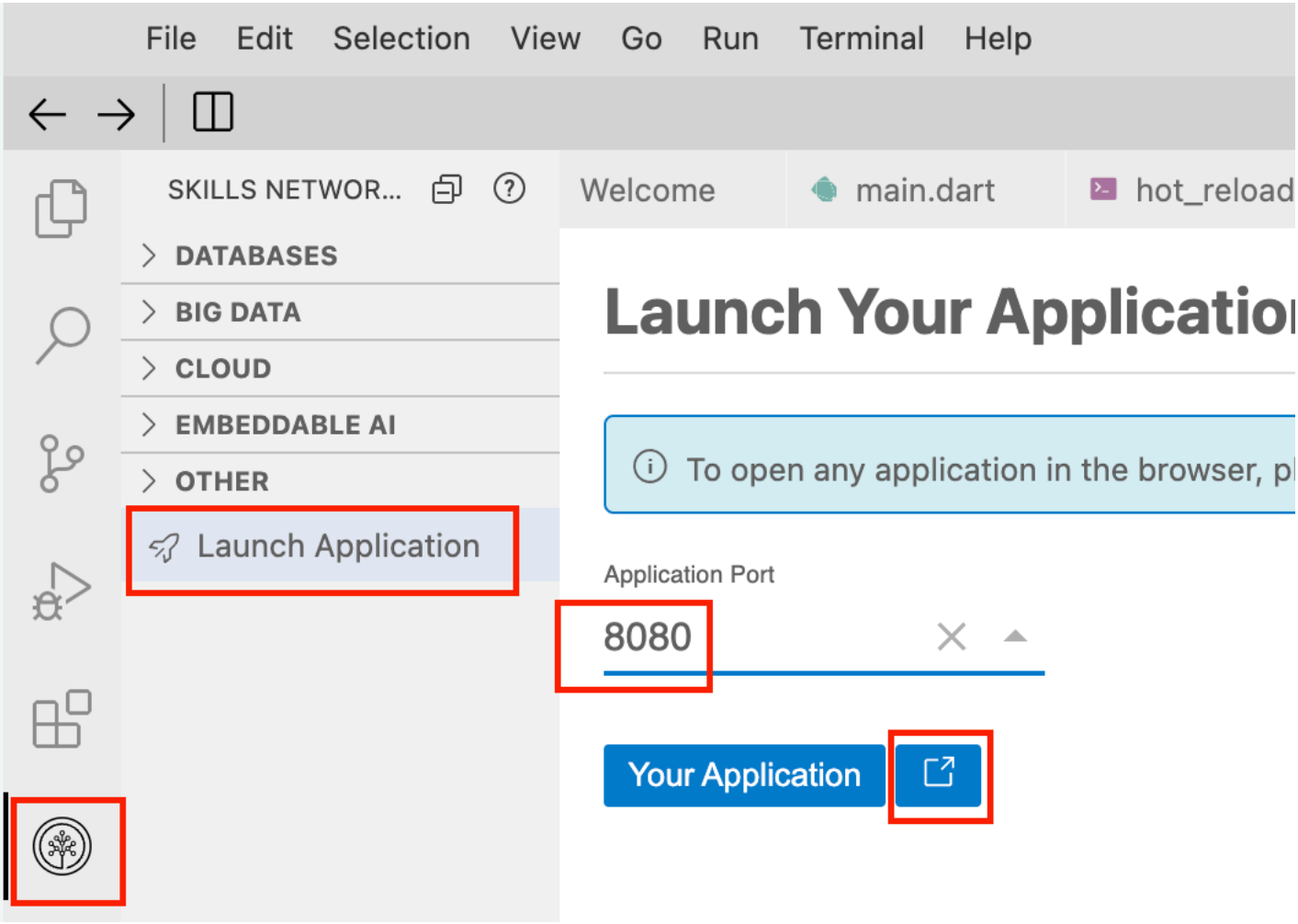
```
./hot_reload.sh
```

You should see output like this:

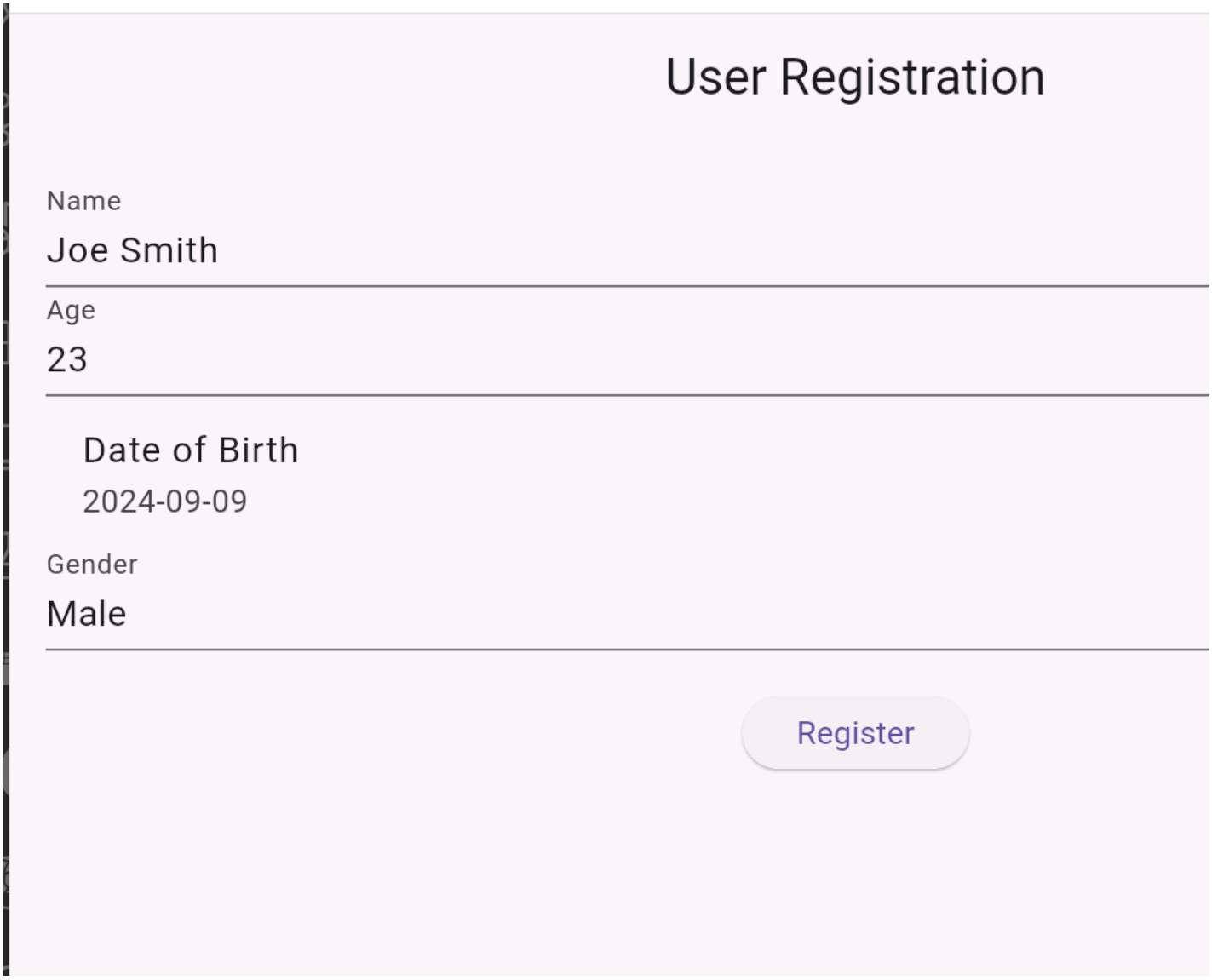
```
theia@theia-captainfedo1:/home/project$ ./hot_reload.sh
Using PROJECT_DIR: /home/project/flutter_navigation_lab/lib
inotify-tools is already installed.
lsof is already installed.
Starting Flutter web server...
Port 8080 is free.
Compiling lib/main.dart for the Web...                             984ms
✓ 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 let bar of the Cloud IDE.
- click on `Launch Application` button
- enter `8080` as the port
- select the `launch in new window` to open the applicaiton in a new tab



You should see the first screen with different form elements. Play around with the various widgets like dropdown button and date.



Step 3: Set up the details screen

In this second step, you will create a second screen that displays the user information captured in the first screen. You will add a flutter route to navigate the user to the second screen when they select the `submit` button after filling out the form.

1. In your `lib` directory, create a new file named `second_screen.dart`:

```
import 'package:flutter/material.dart';
class SecondScreen extends StatelessWidget {
  const SecondScreen({super.key});
  @override
  Widget build(BuildContext context) {
    final Map<String, dynamic> userData =
        ModalRoute.of(context)?.settings.arguments as Map<String, dynamic>;
    return Scaffold(
      appBar: AppBar(
        title: const Text('User Information'),
      ),
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Text('Name: ${userData['name']}'),
            Text('Age: ${userData['age']}'),
            Text('Date of Birth: ${userData['dateOfBirth']}'),
            Text('Gender: ${userData['gender']}'),
          ],
        ),
      ),
    );
  }
}
```

- `second_screen.dart` file contains the `SecondScreen` widget, which displays the user data passed from the `HomeScreen`.
- The data is retrieved using `ModalRoute.of(context)?.settings.arguments`.

2. Save the `second_screen.dart` file. Let's go back to the `main.dart` file where we had commented out some code for the second screen. We will remove the comments now.

   Change `//import 'second_screen.dart';` to `import 'second_screen.dart';`. Also change `//'/second': (context) => const SecondScreen(),` to `'/second': (context) => const SecondScreen(),`.

Step 4: Run the final app

Run your code again to ensure both screens are working correctly and that user data is correctly displayed on the second screen:

If you already have the app running from before, you don't have to do anything. The app should rebuild and redeploy to the server automatically. If you quit that process for some reason or do not see the changes, 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 the information you added in the first screen with the form:

Conclusion and next steps

Congratulations on completing this lab! You have learned how to create a user registration form with multiple input types, navigate between screens, and transfer data between them in a Flutter application.

Author

Skills Network