Flutter

Training Assignments

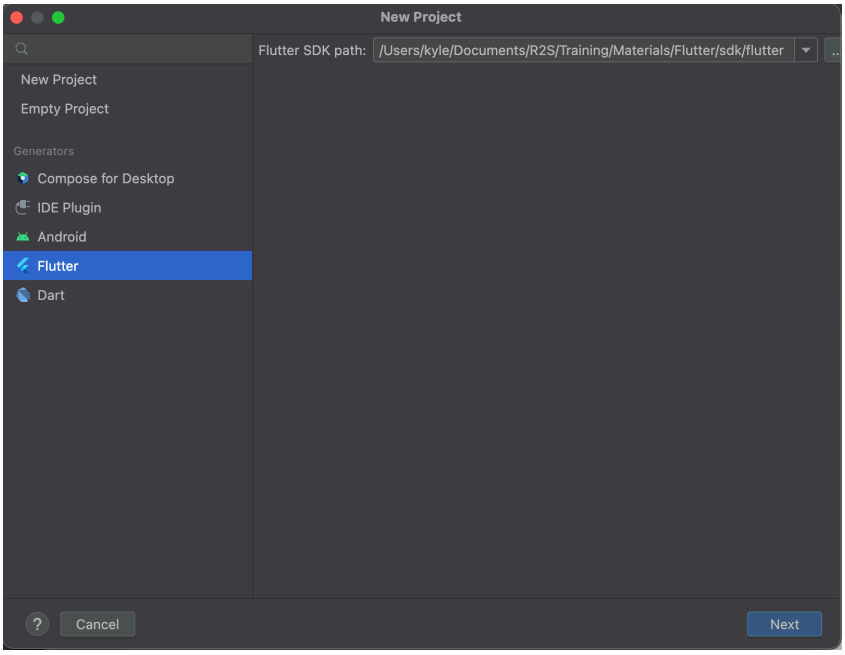
Creating Simple Application

Overview

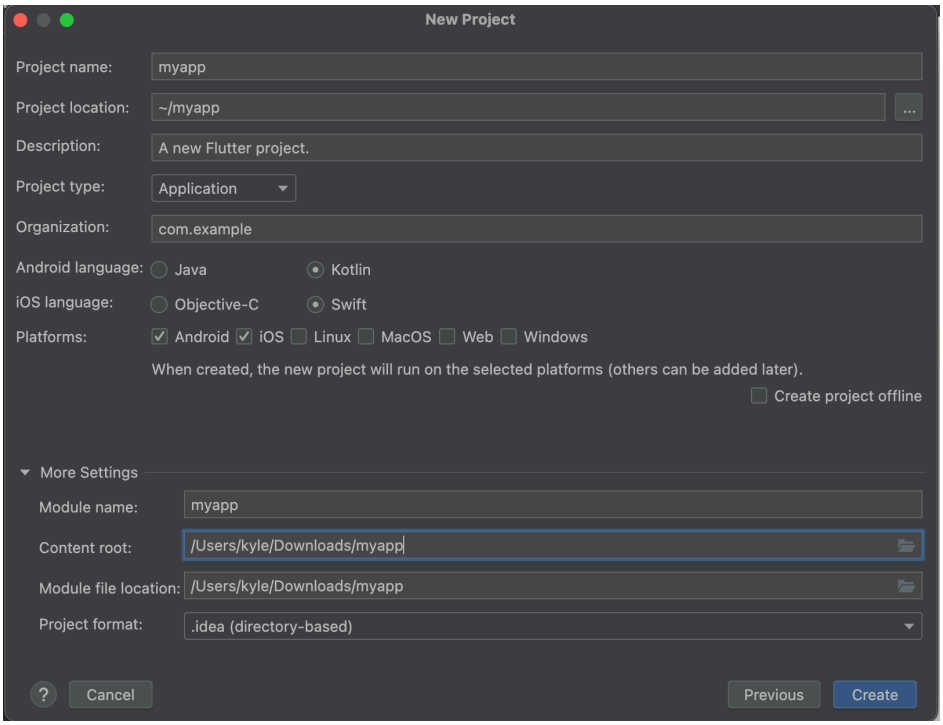
In this practice, you will learn how to create a Simple Hello World Fuller application.

Tasks

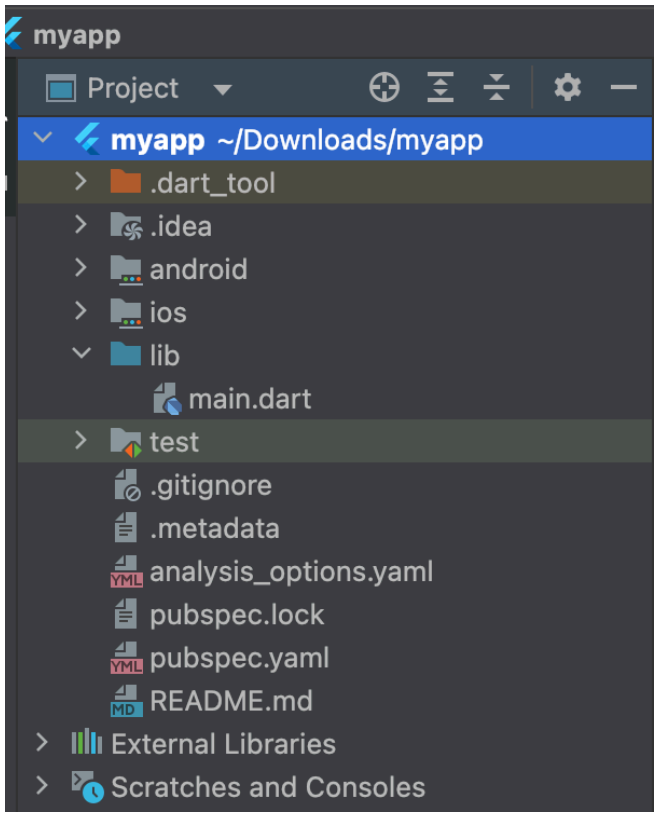
1. Open Android Studio and select Select **Flutter**, verify the Flutter SDK path with the SDK's location. Then click Next.



2. Enter a project name (for example, **myapp**). Select **Application** as the project type. Then click **Finish**.



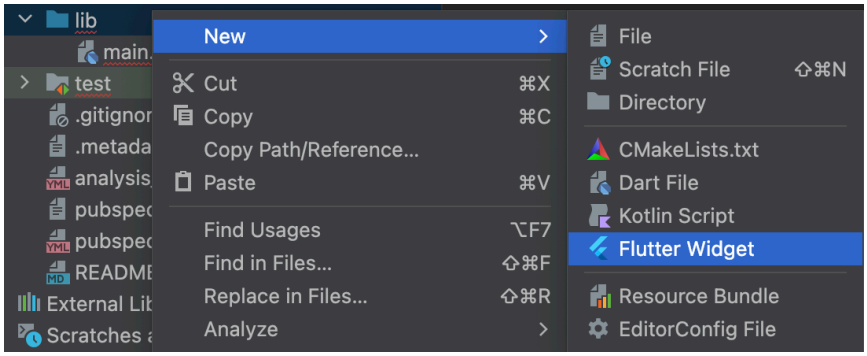
3. Wait for Android Studio to create the project.



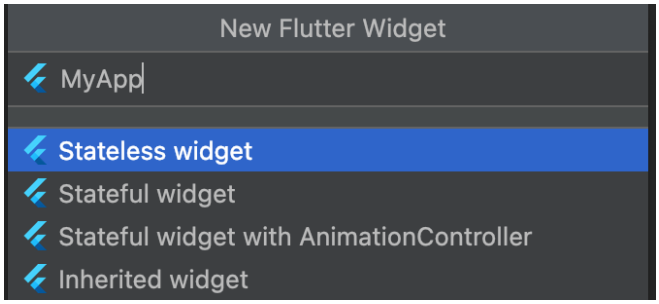
4. Open the **main.dart** file and modify the given code below:

```
1. import 'my_app.dart';
2.
3. void main() {
4.   runApp(const MyApp());
5. }
```

5. Create **Flutter Widget**



Type **MyApp**

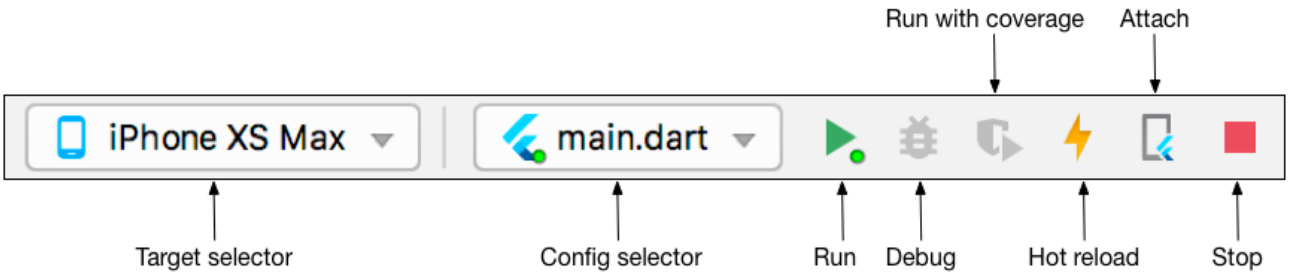


Open the **my_app.dart** file and write the given code below:

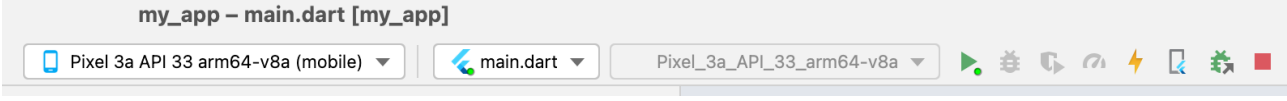
```
1.  class MyApp extends StatelessWidget {
2.    const MyApp({super.key});
3.
4.    @override
5.    Widget build(BuildContext context) {
6.      return MaterialApp(
7.        home: _HomePage(),
8.      );
9.    }
10. }
11.
12. class _HomePage extends StatelessWidget {
13.   @override
14.   Widget build(BuildContext context) {
15.     return Scaffold(
16.       appBar: AppBar(title: const Text('Flutter Demo Home Page'),),
17.       body: const Center(
18.         child: Text('Hello World'),
19.       ),
20.     );
21.   }
22. }
```

6. Run the app

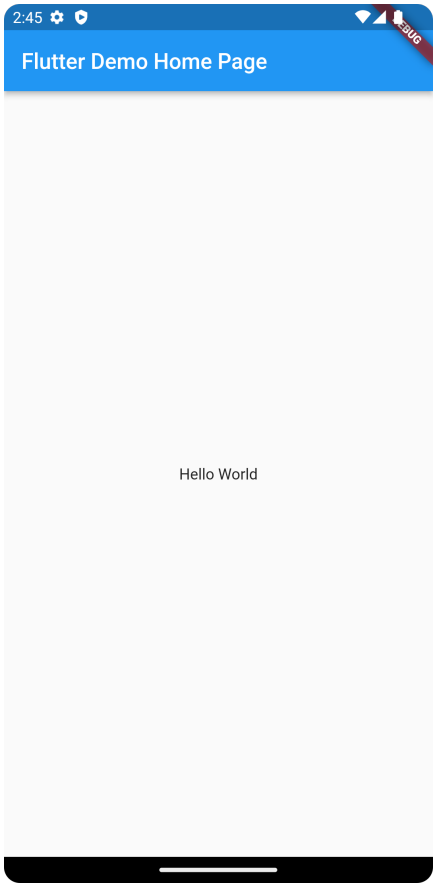
- Locate the main Android Studio toolbar



- In the target selector, select an Android device for running the app. If none are listed as available, select Tools > AVD Manager and create one there.



- Click the run icon in the toolbar. After the app build completes, you'll see the starter app on your device.



Android emulator

iOS simulator

Working with Widgets

Overview

In this practice, you will learn about Flutter platform specific widgets.

Tasks

1. Open the Android Studio, navigate to the project folder and create file named **pavlova_recipe.dart**

2. Open the **pavlova_recipe.dart** file and write code below

```
class PavlovaRecipe extends StatelessWidget {
const PavlovaRecipe({super.key}) ;

 @override
 Widget build(BuildContext context) {
  return  MaterialApp(
    home: buildHomePage(),
  );
 }

Widget buildHomePage() {
  const titleText = Padding(
    padding: EdgeInsets.only(bottom: 10),
    child: Text(
      'Strawberry Pavlova',
      style: TextStyle(
        fontWeight: FontWeight.w800,
        letterSpacing: 0.5,
        fontSize: 25,
      ),
    ),
  );

  const subTitle = Text(
    'Pavlova is a meringue-based dessert named after the Russian ballerina '
        'Anna Pavlova. Pavlova features a crisp crust and soft, light inside, '
        'topped with fruit and whipped cream.',
    textAlign: TextAlign.center,
    style: TextStyle(
      fontFamily: 'Georgia',
      fontSize: 20,
```

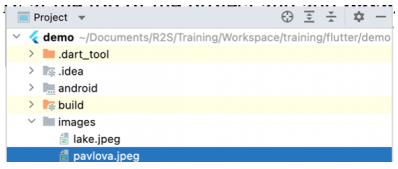
```
    ),
  );

  // #docregion ratings, stars
  var stars = Row(
    mainAxisSize: MainAxisSize.min,
    children: [
      Icon(Icons.star, color: Colors.green[500]),
      Icon(Icons.star, color: Colors.green[500]),
      Icon(Icons.star, color: Colors.green[500]),
      const Icon(Icons.star, color: Colors.black),
      const Icon(Icons.star, color: Colors.black),
    ],
  );

  // #enddocregion stars
  var ratings = Container(
    padding: const EdgeInsets.all(20),
    child: Row(
      mainAxisAlignment: MainAxisAlignment.spaceEvenly,
      children: [
        stars,
        const Text(
          '170 Reviews',
          style: TextStyle(
            color: Colors.black,
            fontWeight: FontWeight.w800,
            fontFamily: 'Roboto',
            letterSpacing: 0.5,
            fontSize: 20,
          ),
        ),
      ],
    ),
  );
  // #enddocregion ratings

  // #docregion iconList
  const descTextStyle = TextStyle(
    color: Colors.black,
```

```
72.       fontWeight: FontWeight.w800,
73.       fontFamily: 'Roboto',
74.       letterSpacing: 0.5,
75.       fontSize: 18,
76.       height: 2,
77.     );
78.
79.     // DefaultTextStyle.merge() allows you to create a default text
80.     // style that is inherited by its child and all subsequent children.
81.     final iconList = DefaultTextStyle.merge(
82.       style: descTextStyle,
83.       child: Container(
84.         padding: const EdgeInsets.all(20),
85.         child: Row(
86.           mainAxisAlignment: MainAxisAlignment.spaceEvenly,
87.           children: [
88.             Column(
89.               children: [
90.                 Icon(Icons.kitchen, color: Colors.green[500]),
91.                 const Text('PREP:'),
92.                 const Text('25 min'),
93.               ],
94.             ),
95.             Column(
96.               children: [
97.                 Icon(Icons.timer, color: Colors.green[500]),
98.                 const Text('COOK:'),
99.                 const Text('1 hr'),
100.              ],
101.            ),
102.            Column(
103.              children: [
104.                Icon(Icons.restaurant, color: Colors.green[500]),
105.                const Text('FEEDS:'),
106.                const Text('4-6'),
107.              ],
108.            ),
109.          ],
110.        ),
111.      ),
```

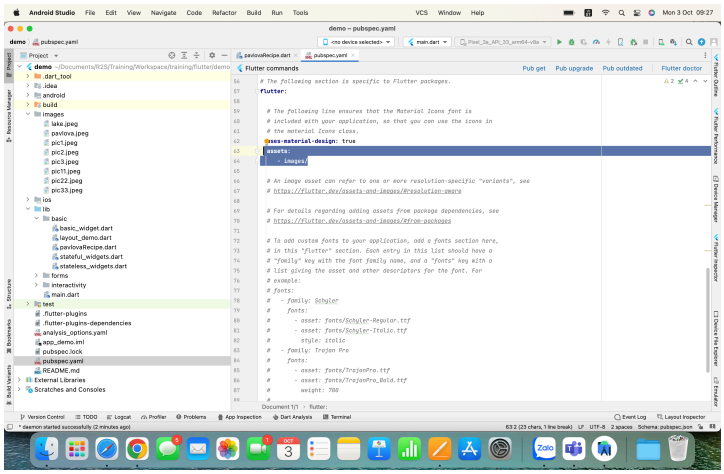
```
112.   );
113.   // #enddocregion iconList
114.
115.   final mainImage = Image.asset(
116.     'images/pavlova.jpeg',
117.   );
118.
119.   // #docregion mainColumn
120.   final mainColumn = Container(
121.     padding: const EdgeInsets.all(20),
122.     child: Column(
123.       children: [
124.         titleText,
125.         subTitle,
126.         ratings,
127.         iconList,
128.         //mainImage,
129.         Expanded(child: mainImage),
130.       ],
131.     ),
132.   );
133.   // #enddocregion mainColumn
134.
135.   return Scaffold(
136.     appBar: AppBar(
137.       title: const Text('Pavlova Recipe'),
138.     ),
139.     // #docregion body
140.     body: mainColumn,
141.     // #enddocregion body
142.   );
143. }
```

- Implement the **mainImage** at buildHomePage

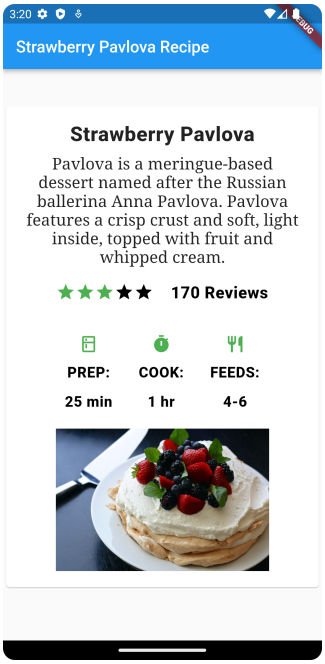
+ Create an **images** directory at the top of the project and add **pavlova.jpeg**



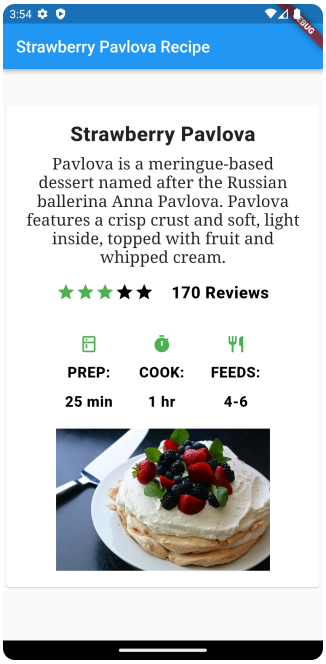
+ Update the **pubspec.yaml** file to include an **assets** tag. This makes the image available to your code.



3. Run the app



Android emulator



iOS simulator

Building layouts

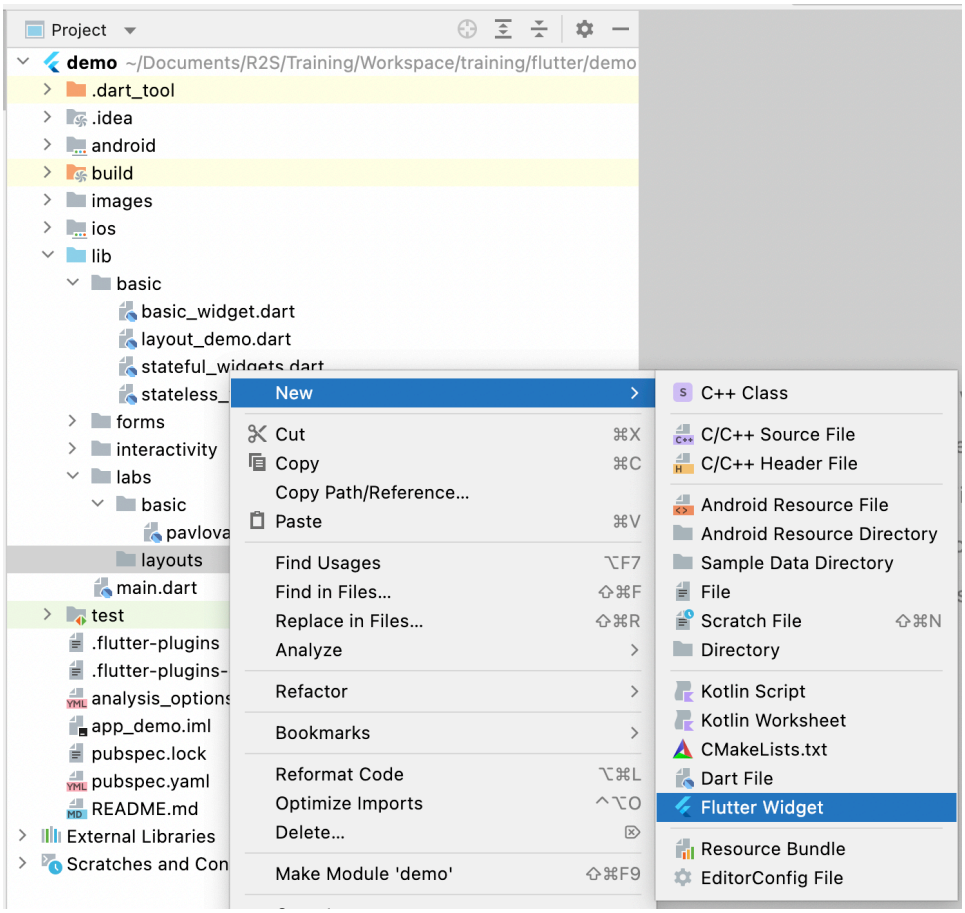
Overview

In this practice, you'll create a simple mobile Flutter app.

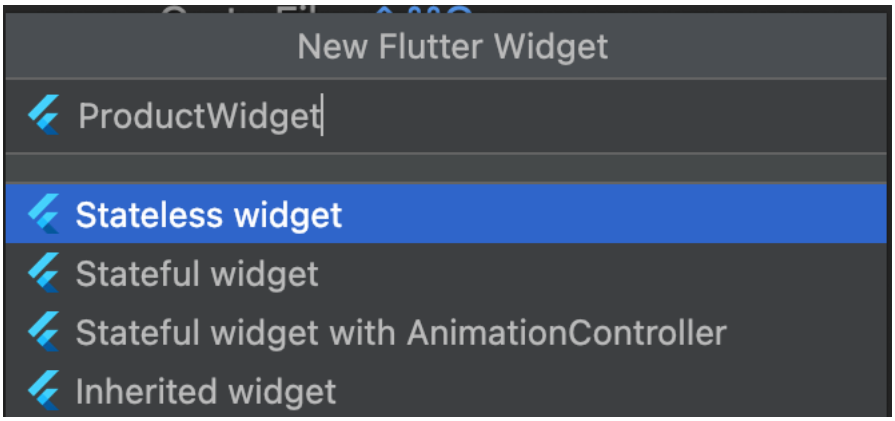
Tasks

1. Open the Android Studio, navigate to the project folder

+ Create Flutter Widget



+ Type named **ProductWidget** and press enter



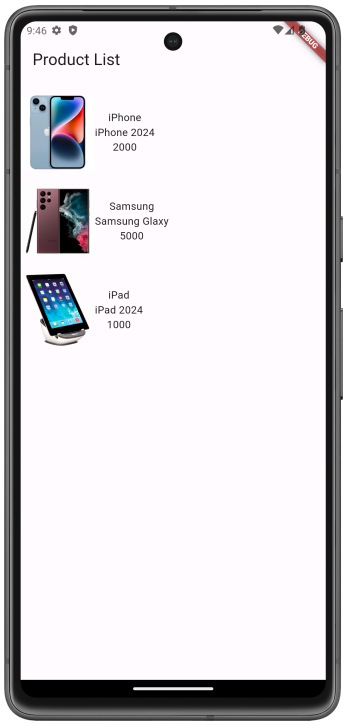
+ Replace the contents of **product_widget.dart**

```
class ProductWidget extends StatelessWidget {
  const ProductWidget(
      {super.key,
      required this.imagePath,
      required this.name,
      required this.description,
      required this.price});

  final String imagePath;
  final String name;
  final String description;
  final num price;

  @override
  Widget build(BuildContext context) {
    return Container(
      margin: const EdgeInsets.only(top: 20),
      child: Row(
        children: [
          SizedBox(
            width: 100,
            height: 100,
            child: Image.asset(imagePath),
          ),
          Column(
            children: [
              Text(name),
              Text(description),
              Text('$price'),
            ],
          )
        ],
      ),
    );
  }
}
```

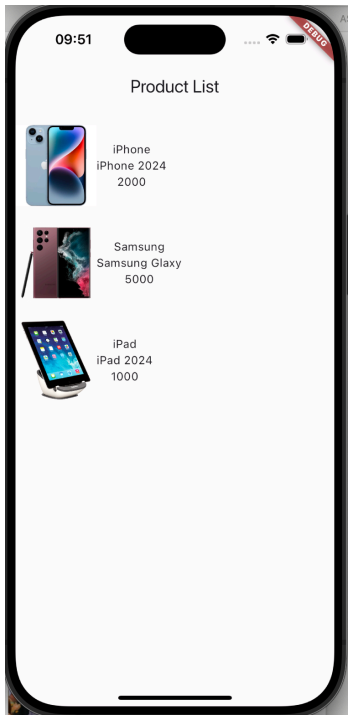
+ Create Flutter Widget named **ProductList** and replace the contents of **product_list.dart**

```
class ProductList extends StatelessWidget {
  const ProductList({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: _buildHomePage(),
    );
  }

  Widget _buildHomePage() {
    return Scaffold(
      appBar: AppBar(
          title: const Text(
        'Product List',
      )),
      body: const Column(
        children: [
          ProductWidget(
              imagePath: 'images/products/iphone.png',
              name: 'iPhone',
              description: 'iPhone 2024',
              price: 2000),
          ProductWidget(
              imagePath: 'images/products/samsung.png',
              name: 'Samsung',
              description: 'Samsung Glaxy',
              price: 5000),
          ProductWidget(
              imagePath: 'images/products/tablet.png',
              name: 'iPad',
              description: 'iPad 2024',
              price: 1000),
        ],
      ),
    );
  }
}
```

2. Run the app

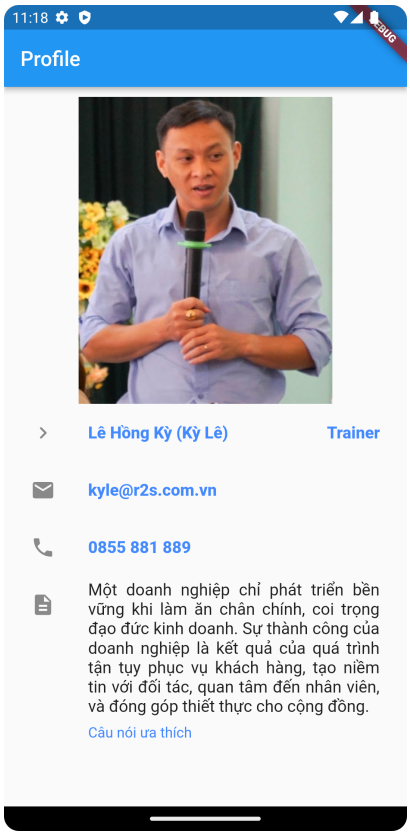


Android emulator

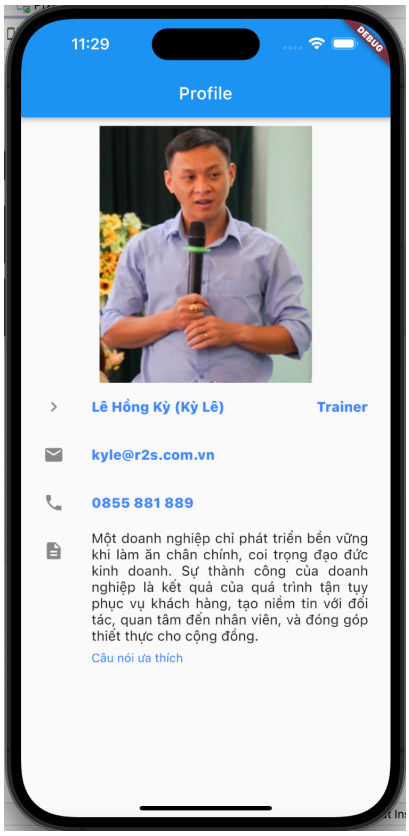


iOS simulator

Extra tasks



Android emulator



iOS simulator

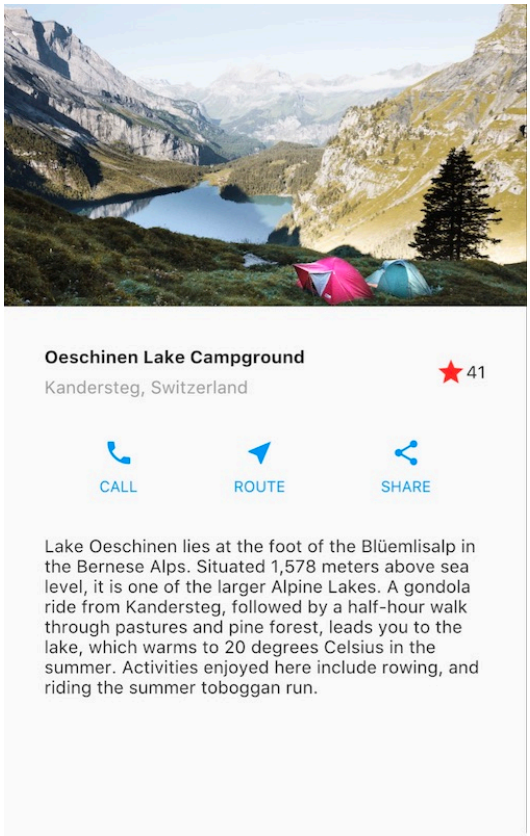
StatefulWidget to your Flutter app

Overview

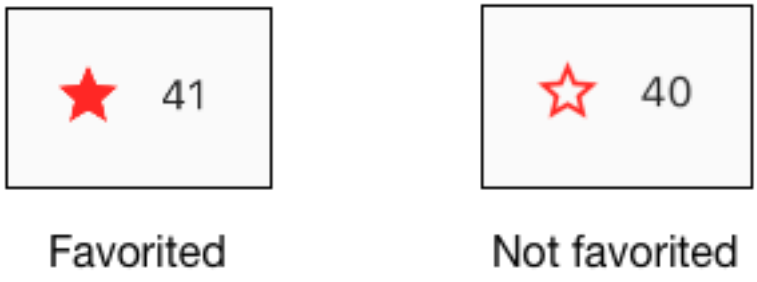
What you'll learn

- How to respond to taps.
- How to create a custom widget.
- The difference between stateless and stateful widgets.

The building layouts tutorial showed you how to create the layout for the following screenshot.



When the app first launches, the star is solid red, indicating that this lake has previously been favorited. The number next to the star indicates that 41 people have favorited this lake. After completing this tutorial, tapping the star removes its favorited status, replacing the solid star with an outline and decreasing the count. Tapping again favorites the lake, drawing a solid star and increasing the count.



Tasks

1. Subclass StatefulWidget

lib/main.dart (FavoriteWidget)

```
class FavoriteWidget extends StatefulWidget {
  const FavoriteWidget({super.key});

  @override
  State<FavoriteWidget> createState() => _FavoriteWidgetState();
}
```

2. Subclass State

lib/main.dart (_FavoriteWidgetState fields)

```
class _FavoriteWidgetState extends State<FavoriteWidget> {
  bool _isFavorited = true;
  int _favoriteCount = 41;

  // ···
}
```

+ The class also defines a build() method, which creates a row containing a red IconButton, and Text. You use IconButton (instead of Icon) because it has an onPressed property that defines the callback function (_toggleFavorite) for handling a tap.

lib/main.dart (_FavoriteWidgetState build)

```
class _FavoriteWidgetState extends State<FavoriteWidget> {
  // ···
  @override
  Widget build(BuildContext context) {
    return Row(
      mainAxisSize: MainAxisSize.min,
      children: [
        Container(
          padding: const EdgeInsets.all(0),
          child: IconButton(
            padding: const EdgeInsets.all(0),
            alignment: Alignment.centerRight,
            icon: (_isFavorited
                ? const Icon(Icons.star)
                : const Icon(Icons.star_border)),
            color: Colors.red[500],
            onPressed: _toggleFavorite,
          ),
        ),
        SizedBox(
          width: 18,
          child: SizedBox(
            child: Text('$_favoriteCount'),
          ),
        ),
      ],
    );
  }
}
```

+ The _toggleFavorite() method

```
void _toggleFavorite() {
  setState(() {
    if (_isFavorited) {
      _favoriteCount -= 1;
      _isFavorited = false;
    } else {
      _favoriteCount += 1;
      _isFavorited = true;
    }
  });
}
```

3. Plug the stateful widget into the widget tree

```
class FavoriteApp extends StatelessWidget {
  const FavoriteApp({super.key});

  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    Widget titleSection = Container(
      padding: const EdgeInsets.all(32),
      child: Row(
        children: [
          Expanded(child: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              Container(
                padding: const EdgeInsets.only(bottom: 8),
                child: const Text(
                  'Oeschinen Lake Camground',
                  style: TextStyle(
                    fontWeight: FontWeight.bold,
                  ),  // TextStyle
                ),  // Text
              ),  // Container
              const Text(
                'Kandersteg, Switzerland',
                style: TextStyle(
                  fontWeight: FontWeight.bold,
                ),  // TextStyle
              )  // Text
            ],
          )  // Column
          ),  // Expanded
          const FavoriteWidget(),
        ],
      ),  // Row
    );  // Container
```

Full code

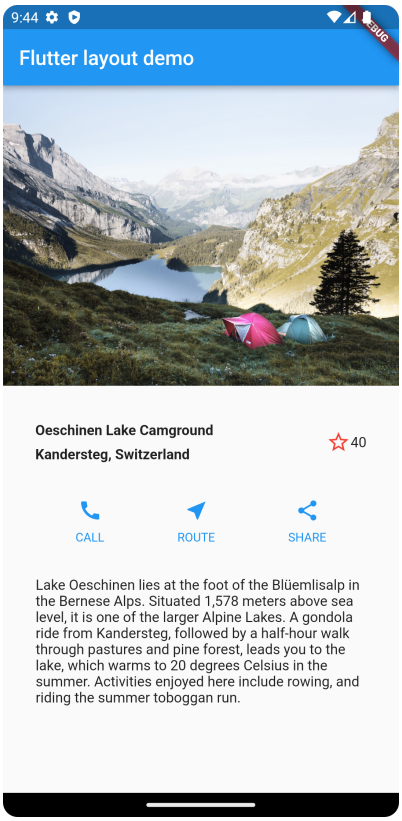
```
class FavoriteApp extends StatelessWidget {
  const FavoriteApp({super.key});

  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    Widget titleSection = Container(
      padding: const EdgeInsets.all(10),
      child: Row(
        children: [
          Expanded(
            child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Container(
              padding: const EdgeInsets.only(bottom: 8),
              child: const Text(
                'Oeschinen Lake Camground',
                style: TextStyle(
                  fontWeight: FontWeight.bold,
                ),
              ),
            ),
            const Text(
              'Kandersteg, Switzerland',
              style: TextStyle(
                fontWeight: FontWeight.bold,
              ),
            )
          ],
        )),
        const FavoriteWidget(),
      ],
    ),
  );

  Color color = Theme.of(context).primaryColor;

  Widget buttonSection = Row(
```

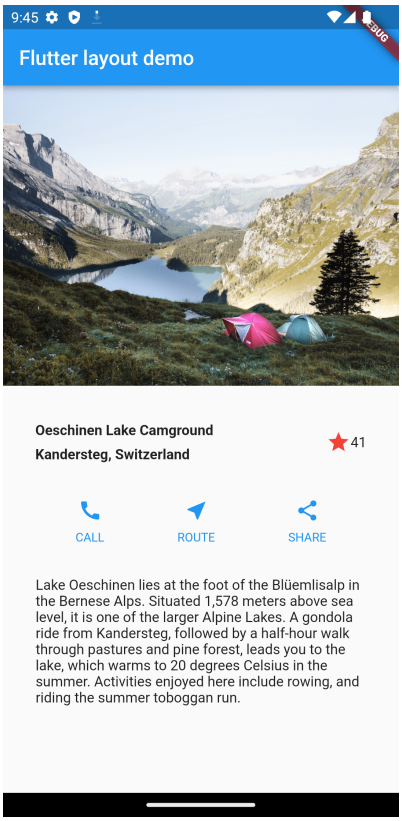
```
    mainAxisAlignment: MainAxisAlignment.spaceEvenly,
    children: [
      _buildButtonColumn(color, Icons.call, 'CALL'),
      _buildButtonColumn(color, Icons.near_me, 'ROUTE'),
      _buildButtonColumn(color, Icons.share, 'SHARE'),
    ],
  );

  Widget textSection = const Expanded(
    child: Padding(
      padding: EdgeInsets.all(10),
      child: Text(
        'Lake Oeschinen lies at the foot of the Blüemlisalp in the Bernese '
        'Alps. Situated 1,578 meters above sea level, it is one of the '
        'larger Alpine Lakes. A gondola ride from Kandersteg, followed by a '
        'half-hour walk through pastures and pine forest, leads you to the '
        'lake, which warms to 20 degrees Celsius in the summer. Activities '
        'enjoyed here include rowing, and riding the summer toboggan run.',
        softWrap: true,
        textAlign: TextAlign.justify,
      ),
    ),
  );

  return MaterialApp(
    debugShowCheckedModeBanner: false,
    home: Scaffold(
      appBar: AppBar(
        title: const Text('Favorite App'),
      ),
      body: Column(
        children: [
          Image.asset(
            'images/lake.jpeg',
          ),
          titleSection,
          buttonSection,
          textSection,
        ],
      ),
```

```
      ),
    );
  }

  Column _buildButtonColumn(Color color, IconData icon, String label) {
    return Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        Icon(
          icon,
          color: color,
        ),
        Container(
          margin: const EdgeInsets.only(top: 8),
          child: Text(
            label,
            style: TextStyle(
              fontSize: 12,
              fontWeight: FontWeight.w400,
              color: color,
            ),
          ),
        )
      ],
    );
  }
}

class FavoriteWidget extends StatefulWidget {
  const FavoriteWidget({super.key});

  @override
  State<StatefulWidget> createState() => _FavoriteWidgetState();
}

class _FavoriteWidgetState extends State<FavoriteWidget> {
  bool _isFavorited = true;
  int _favoriteCount = 41;

  @override
```

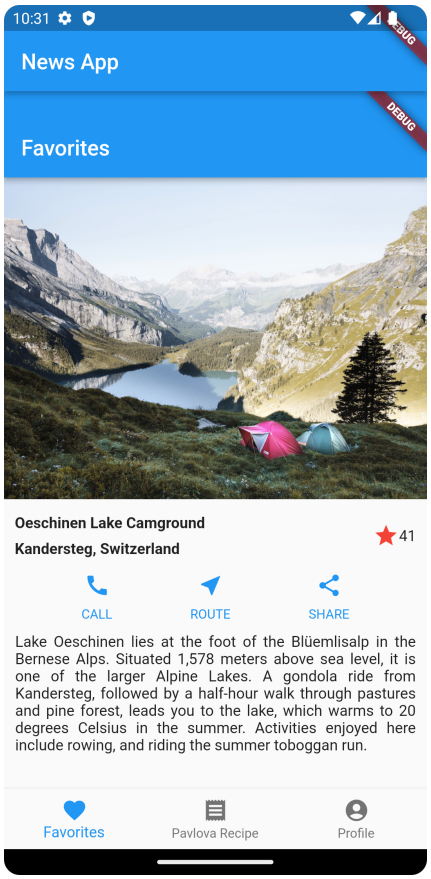
```
  Widget build(BuildContext context) {
    return Row(
      mainAxisSize: MainAxisSize.min,
      children: [
        Container(
          padding: const EdgeInsets.all(0),
          child: IconButton(
            padding: const EdgeInsets.all(0),
            alignment: Alignment.centerRight,
            icon: (_isFavorited
                ? const Icon(Icons.star)
                : const Icon(Icons.star_border)),
            color: Colors.red,
            onPressed: _toggleFavorite,
          ),
        ),
        Text('$_favoriteCount'),
      ],
    );
  }

  void _toggleFavorite() {
    setState(() {
      if (_isFavorited) {
        _favoriteCount -= 1;
        _isFavorited = false;
      } else {
        _favoriteCount += 1;
        _isFavorited = true;
      }
    });
  }
}
```

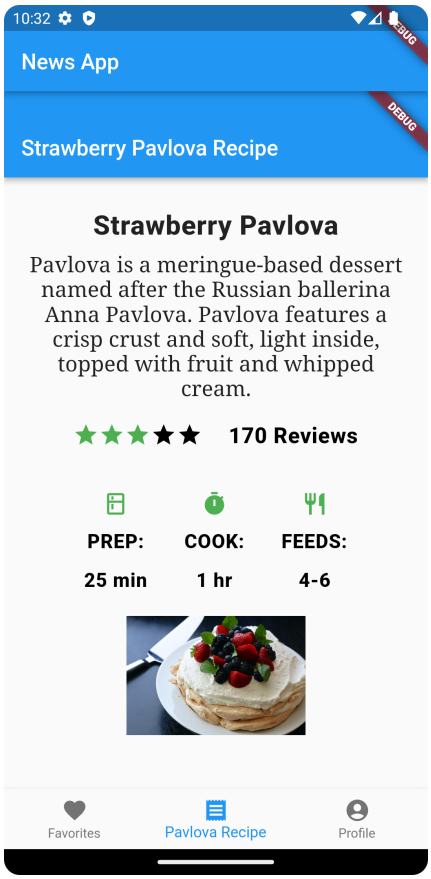
4. Run the app

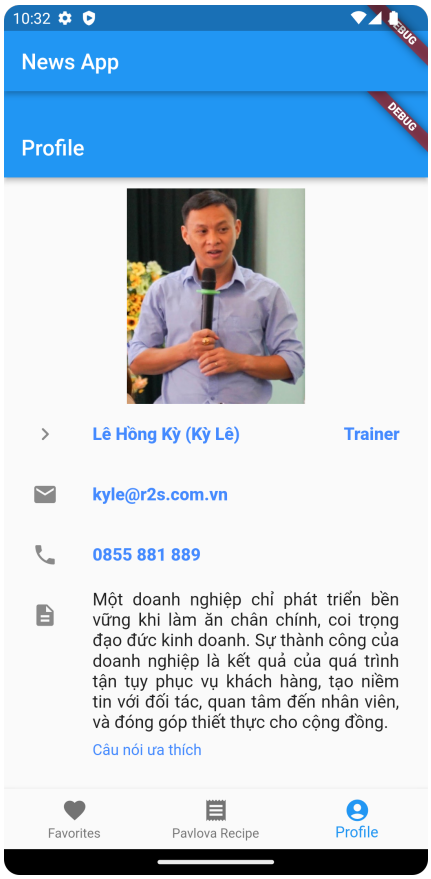




Extra tasks







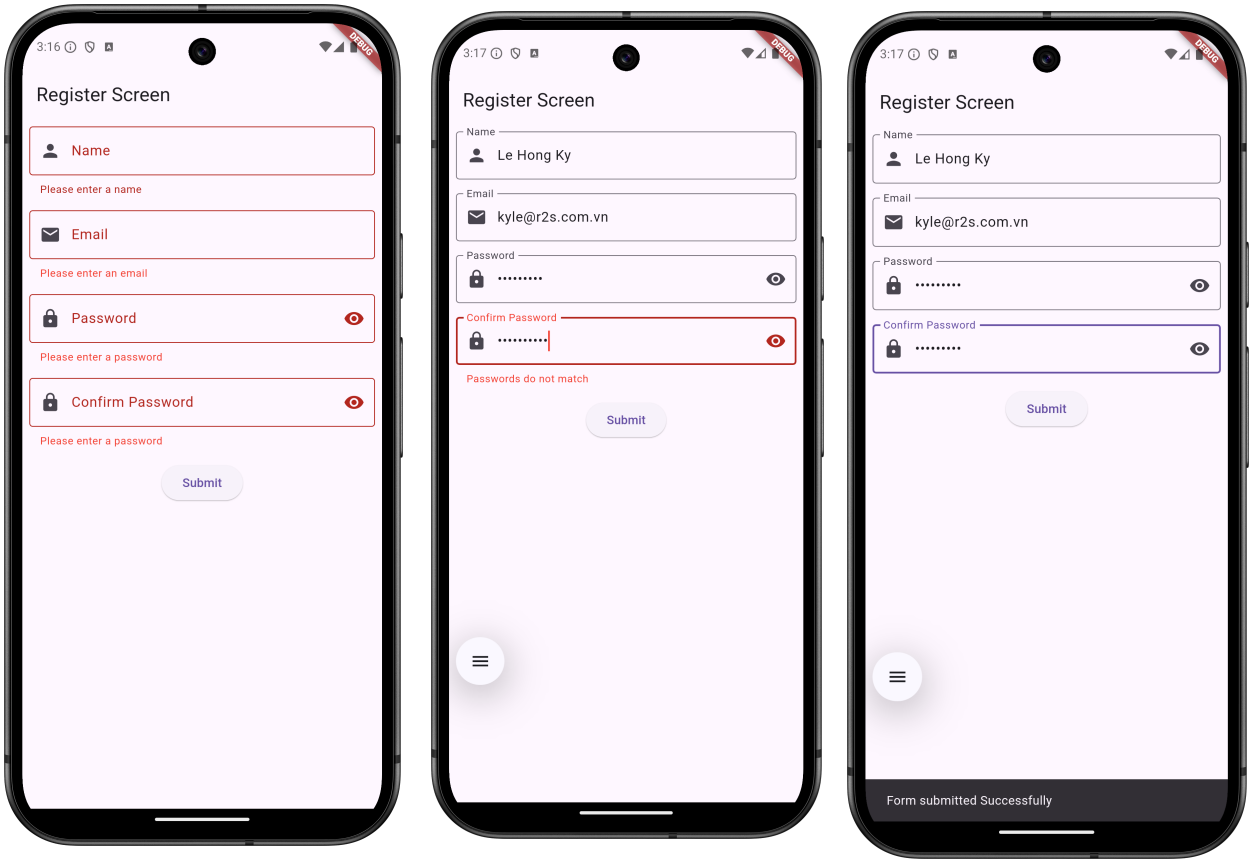
Forms

Overview

What you'll learn

- Build a form with validation.
- Create and style a text field.
- Retrieve the value of a text field.

Tasks



Code

register_screen.dart

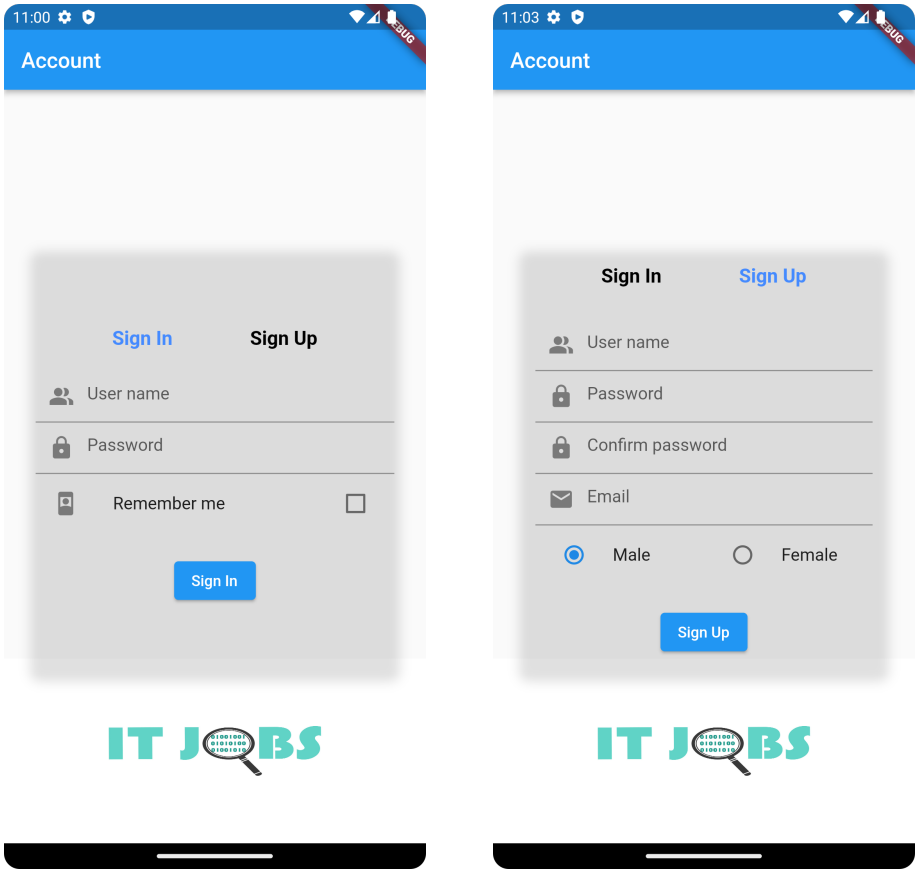
```
class RegisterScreen extends StatefulWidget {
 const RegisterScreen({super.key});

 @override
 State<RegisterScreen> createState() => _RegisterScreenState();
}

class _RegisterScreenState extends State<RegisterScreen> {
 final _formKey = GlobalKey<FormState>();
 final _nameController = TextEditingController();
 final _emailController = TextEditingController();
 final _passwordController = TextEditingController();
 final _confirmPasswordController = TextEditingController();
 bool _obscurePassword = true;
 var _obscureConfirmPassword = true;

 @override
 Widget build(BuildContext context) {
  return Scaffold(
   appBar: AppBar(
    title: const Text('Register Screen'),
   ),
   body: _buildForm(),
  );
 }

 String? _validateName(String? value) {
  if (value == null || value.isEmpty) {
   return 'Please enter a name';
  }

  return null;
 }

 String? _validateEmail(String? value) {
  if (value == null || value.isEmpty) {
   return 'Please enter an email';
  }
```

```
39.
40.   String pattern = r'\w+@\w+\.\w+';
41.   if (!RegExp(pattern).hasMatch(value)) {
42.     return 'Not a valid email';
43.   }
44.
45.   return null;
46. }
47.
48. String? _validatePassword(String? value) {
49.   if (value == null || value.isEmpty) {
50.     return 'Please enter a password';
51.   } else if (value.length < 8) {
52.     return 'Password too short';
53.   }
54.
55.   return null;
56. }
57.
58. String? _validateConfirmPassword(String? value) {
59.   if (value == null || value.isEmpty) {
60.     return 'Please enter a password';
61.   } else if (value.length < 8) {
62.     return 'Password too short';
63.   } else if (value != _passwordController.text) {
64.     return 'Passwords do not match';
65.   }
66.
67.   return null;
68. }
69.
70. Widget _buildForm() {
71.   return Padding(
72.     padding: const EdgeInsets.all(8.0),
73.     child: Form(
74.       key: _formKey,
75.       child: Column(
76.         children: [
77.           TextFormField(
78.             controller: _nameController,
```

```
79.               decoration: const InputDecoration(
80.                   labelText: 'Name',
81.                   border: OutlineInputBorder(),
82.                   prefixIcon: Icon(Icons.person)),
83.               validator: _validateName,
84.             ),
85.             const SizedBox(height: 16),
86.             TextFormField(
87.               controller: _emailController,
88.               decoration: const InputDecoration(
89.                   labelText: 'Email',
90.                   border: OutlineInputBorder(),
91.                   errorStyle: TextStyle(color: Colors.red),
92.                   prefixIcon: Icon(Icons.email)),
93.               validator: _validateEmail,
94.             ),
95.             const SizedBox(
96.               height: 16,
97.             ),
98.             TextFormField(
99.               controller: _passwordController,
100.              decoration: InputDecoration(
101.                  labelText: 'Password',
102.                  border: const OutlineInputBorder(),
103.                  errorStyle: const TextStyle(color: Colors.red),
104.                  prefixIcon: const Icon(Icons.lock),
105.                  suffixIcon: IconButton(
106.                      onPressed: () {
107.                        setState(() {
108.                          _obscurePassword = !_obscurePassword;
109.                        });
110.                      },
111.                      icon: _obscurePassword
112.                          ? const Icon(Icons.visibility)
113.                          : const Icon(Icons.visibility_off))),
114.              validator: _validatePassword,
115.              obscureText: _obscurePassword,
116.            ),
117.            const SizedBox(
118.              height: 16,
```

```
119.             ),
120.             TextFormField(
121.               controller: _confirmPasswordController,
122.               decoration: InputDecoration(
123.                   labelText: 'Confirm Password',
124.                   border: const OutlineInputBorder(),
125.                   errorStyle: const TextStyle(color: Colors.red),
126.                   prefixIcon: const Icon(Icons.lock),
127.                   suffixIcon: IconButton(
128.                       onPressed: () {
129.                         setState(() {
130.                           _obscureConfirmPassword = !_obscureConfirmPassword;
131.                         });
132.                       },
133.                       icon: _obscureConfirmPassword
134.                           ? const Icon(Icons.visibility)
135.                           : const Icon(Icons.visibility_off))),
136.               obscureText: _obscureConfirmPassword,
137.               validator: _validateConfirmPassword,
138.             ),
139.             const SizedBox(
140.               height: 16,
141.             ),
142.             // Submit button
143.             Center(
144.               child: ElevatedButton(
145.                 onPressed: () {
146.                   if (_formKey.currentState!.validate()) {
147.                     ScaffoldMessenger.of(context).showSnackBar(const SnackBar(
148.                         content: Text('Form submitted Successfully')));
149.                   }
150.                 },
151.                 child: const Text('Submit'),
152.               ),
153.             )
154.           ],
155.         )),
156.     );
157.   }
158. }
```

Extra tasks



Tips:

- Using Stack class: https://api.flutter.dev/flutter/widgets/Stack-class.html
- Using GestureDetector class
- Using Container class

```
child: Container(
  width: size.width * 0.85,
  height: size.height * 0.5,
  decoration: BoxDecoration(
      borderRadius: BorderRadius.circular(10),
      boxShadow: const [
        BoxShadow(color: Colors.black12, blurRadius: 8, spreadRadius: 5)
      ]),  // BoxDecoration
```

Persist data with SQLite

Overview

In this lab, we are going to build a small Flutter app that uses SQLite to persist data.

The app has a floating button that can be used to show a bottom sheet. That bottom sheet contains 2 text fields corresponding to “title” and “description”. These text fields are used to create a new “item” or update an existing “item”.

The saved “items” are fetched from the SQLite database and displayed with a list view. There are an update button and a delete button associated with each “item”.

Database Structure

We are going to create an SQLite database called **demo.db**. It has only a single table named **items**. Below is the structure of the table:

Column	Type	Description
id	INTEGER	The id of an item and auto increment
title	TEXT	The name of an item
description	TEXT	The detail of an item
createdAt	TIMESTAMP	The time that the item was created. It will be automatically added by SQLite

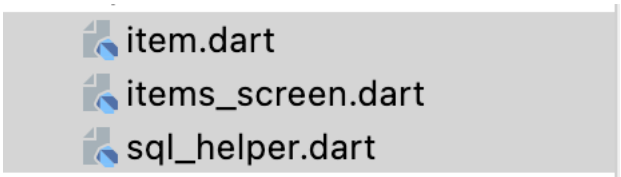
Tasks

1. In the **lib** folder

+ Add a new file named **sql_helper.dart** and **item.dart**

+ Add a new Flutter Widget file named **items_screen.dart**

+ The project structure:



+ Install the **sqflite** plugin (**sqflite**: ^2.1.0+1)

```
dependencies:
  flutter:
    sdk: flutter

  english_words: ^4.0.0

  sqflite: ^2.1.0+1
```

2. Full code in **item.dart**

```
class Item {
  final int? id;
  final String? title;
  final String? description;
  final String? createdAt;

  Item({this.id, this.title, this.description, this.createdAt});

  Map<String, dynamic> toMap() {
    return {
      'id': id,
      'title': title,
      'description': description,
    };
  }
}
```

3. Full code in **sql_helper.dart**

```
class SQLHelper {
  static const _databaseName = "MyDatabase.db";
  static const _databaseVersion = 1;
  static const _itemsTable = 'items';

  static const _columnId = 'id';
  static const _columnTitle = 'title';
  static const _columnDescription = 'description';
  static const _columnCreatedAt = 'createdAt';

  // id: the id of a item
  // title, description: name and description of your activity
  // created_at: the time that the item was created.
  // It will be automatically handled by SQLite
  static Future<void> createItemTable(Database database) async {
    try {
      await database.execute('''
        CREATE TABLE $_itemsTable (
          $_columnId INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
          $_columnTitle TEXT,
          $_columnDescription TEXT,
          $_columnCreatedAt TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
        )''');
    } catch (err) {
      debugPrint("createItemTable(): $err");
    }
  }

  static Future<Database> getDb() async {
    return openDatabase(
      _databaseName,
      version: _databaseVersion,
      onCreate: (Database database, int version) async {
        await createItemTable(database);
      },
    );
  }

  // Create new item
```

```
40.  static Future<int> createItem(Item item) async {
41.    int id = 0;
42.
43.    try {
44.      final db = await SQLHelper.getDb();
45.
46.      // Sometimes you want to insert an empty row, in that case ContentValues
47.      // have no content value, and you should use nullColumnHack.
48.      // For example, you want to insert an empty row into a table student(id, name),
49.      // which id is auto generated and name is null.
50.      id = await db.insert(_itemsTable, item.toMap(),
51.          conflictAlgorithm: ConflictAlgorithm.replace);
52.    } catch (err) {
53.      debugPrint("createItem(): $err");
54.    }
55.
56.    return id;
57.  }
58.
59.  // Read all items
60.  static Future<List<Map<String, dynamic>>> getItems() async {
61.    late Future<List<Map<String, dynamic>>> items;
62.    try {
63.      final db = await SQLHelper.getDb();
64.      items = db.query(_itemsTable, orderBy: _columnId);
65.    } catch (err) {
66.      debugPrint("getItems(): $err");
67.    }
68.
69.    return items;
70.  }
71.
72.  // Read a single item by id
73.  static Future<List<Map<String, dynamic>>> getItem(int id) async {
74.    late Future<List<Map<String, dynamic>>> item;
75.
76.    try {
77.      final db = await SQLHelper.getDb();
78.
79.      item = db.query(_itemsTable,
```

```
80.           where: "$_columnId = ?", whereArgs: [id], limit: 1);
81.     } catch (err) {
82.       debugPrint("getItem(): $err");
83.     }
84.
85.     return item;
86.   }
87.
88.   // Update an item by id
89.   static Future<int> updateItem(Item item) async {
90.     int result = 0;
91.     try {
92.       final db = await SQLHelper.getDb();
93.       result = await db.update(_itemsTable, item.toMap(),
94.           where: "$_columnId = ?", whereArgs: [item.id]);
95.     } catch (err) {
96.       debugPrint("updateItem(): $err");
97.     }
98.
99.     return result;
100.  }
101.
102.  // Delete a single item by id
103.  static Future<void> deleteItem(int id) async {
104.    try {
105.      final db = await SQLHelper.getDb();
106.      await db.delete(_itemsTable, where: "$_columnId = ?", whereArgs: [id]);
107.    } catch (err) {
108.      debugPrint("Something went wrong when deleting an item: $err");
109.    }
110.  }
111.}
```

4. Full code in **item_screen.dart**

```
class ItemsScreen extends StatelessWidget {
  const ItemsScreen({super.key});

  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      home: _HomePage(),
    );
  }
}

class _HomePage extends StatefulWidget {
  const _HomePage({super.key});

  @override
  State<_HomePage> createState() => _HomePageState();
}

class _HomePageState extends State<_HomePage> {
  // All items
  List<Map<String, dynamic>> _items = [];

  bool _isLoading = true;

  // This function is used to fetch all data from the database
  Future<void> _refreshItems() async {
    final data = await SQLHelper.getItems();

    setState(() {
      _items = data;
      _isLoading = false;
    });
  }

  @override
  void initState() {
    super.initState();
    _refreshItems(); // Loading the diary when the app starts
  }
```

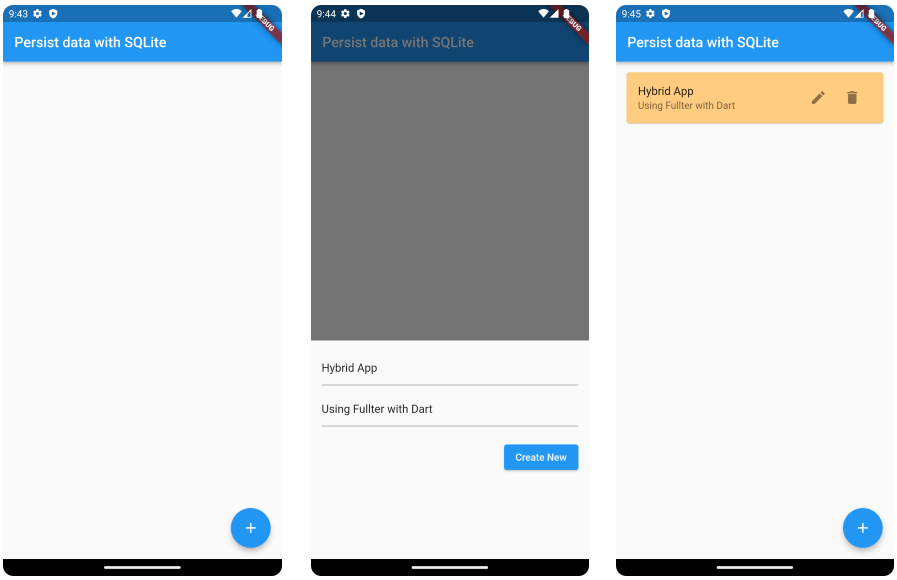
```
40.
41.  final TextEditingController _titleController = TextEditingController();
42.  final TextEditingController _descriptionController = TextEditingController();
43.
44.  // This function will be triggered when the floating button is pressed
45.  // It will also be triggered when you want to update an item
46.  void _showForm(int? id) async {
47.    if (id != null) {
48.      // id == null -> create new item
49.      // id != null -> update an existing item
50.      final existingItem = _items.firstWhere((element) => element['id'] == id);
51.      _titleController.text = existingItem['title'];
52.      _descriptionController.text = existingItem['description'];
53.    } else {
54.      _titleController.text = '';
55.      _descriptionController.text = '';
56.    }
57.
58.    showModalBottomSheet(
59.        context: context,
60.        elevation: 5,
61.        isScrollControlled: true, // fix pixel overflowed
62.        // builder: (_, counter, __) => Translations(counter.value),
63.        // means you have 3 parameters _, counter and __,
64.        // and only counter is what you are using,
65.        // so 1st and 3rd parameters are denoted with _ and __.
66.        // This is just cleaner way to write code.
67.        builder: (_) => Container(
68.              padding: EdgeInsets.only(
69.                top: 15,
70.                left: 15,
71.                right: 15,
72.                // this will prevent the soft keyboard from covering the text fields
73.                bottom: MediaQuery.of(context).viewInsets.bottom + 120,
74.              ),
75.              child: Column(
76.                mainAxisSize: MainAxisSize.min,
77.                crossAxisAlignment: CrossAxisAlignment.end,
78.                children: [
79.                  TextFormField(
```

```
80.                 controller: _titleController,
81.                 decoration: const InputDecoration(hintText: 'Title'),
82.               ),
83.               const SizedBox(
84.                 height: 10,
85.               ),
86.               TextFormField(
87.                 maxLines: 5,
88.                 keyboardType: TextInputType.multiline,
89.                 controller: _descriptionController,
90.                 decoration: const InputDecoration(hintText: 'Description'),
91.               ),
92.               const SizedBox(
93.                 height: 20,
94.               ),
95.               ElevatedButton(
96.                 onPressed: () async {
97.                   // Save new item
98.                   if (id == null) {
99.                     await _addItem();
100.                  } else {
101.                    await _updateItem(id);
102.                  }
103.
104.                  // Clear the text fields
105.                  _titleController.text = '';
106.                  _descriptionController.text = '';
107.
108.                  // Close the bottom sheet
109.                  // if you're not sure your widget is mounted.
110.                  // https://www.educative.io/answers/what-is-buildcontext-in-flutter
111.                  if (!mounted) return;
112.
113.                  Navigator.of(context).pop();
114.                },
115.                child: Text(id == null ? 'Create New' : 'Update'),
116.              )
117.            ],
118.          ),
119.        ));
```

```
120.  }
121.
122.  // Insert a new item to the database
123.  Future<void> _addItem() async {
124.    await SQLHelper.createItem(Item(
125.        title: _titleController.text,
126.        description: _descriptionController.text));
127.
128.    _refreshItems();
129.  }
130.
131.  // Update an existing item
132.  Future<void> _updateItem(int id) async {
133.    await SQLHelper.updateItem(Item(
134.        id: id,
135.        title: _titleController.text,
136.        description: _descriptionController.text));
137.
138.    _refreshItems();
139.  }
140.
141.  // Delete an item
142.  Future<void> _deleteItem(int id) async {
143.    await SQLHelper.deleteItem(id);
144.
145.    // if you're not sure your widget is mounted.
146.    if (!mounted) return;
147.
148.    ScaffoldMessenger.of(context).showSnackBar(const SnackBar(
149.      content: Text('Successfully deleted a item!'),
150.    ));
151.
152.    _refreshItems();
153.  }
154.
155.  @override
156.  Widget build(BuildContext context) {
157.    return Scaffold(
158.      appBar: AppBar(
159.        title: const Text('Persist data with SQLite'),
```

```
      ),
      body: _isLoading
          ? const Center(
              child: CircularProgressIndicator(),
            )
          : ListView.builder(
              itemCount: _items.length,
              itemBuilder: (context, index) => Card(
                color: Colors.orange[200],
                margin: const EdgeInsets.all(15),
                child: ListTile(
                    title: Text(_items[index]['title']),
                    subtitle: Text(_items[index]['description']),
                    trailing: SizedBox(
                      width: 100,
                      child: Row(
                        children: [
                          IconButton(
                            icon: const Icon(Icons.edit),
                            onPressed: () => _showForm(_items[index]['id']),
                          ),
                          IconButton(
                            icon: const Icon(Icons.delete),
                            onPressed: () => _deleteItem(_items[index]['id']),
                          ),
                        ],
                      ),
                    )),
              ),
            ),
      floatingActionButton: FloatingActionButton(
        child: const Icon(Icons.add),
        onPressed: () => _showForm(null),
      ),
    );
  }
}
```

Run the app



Extra tasks: Add registration and login functions. After successful login, redirect to the homepage.

Login Register Homepage

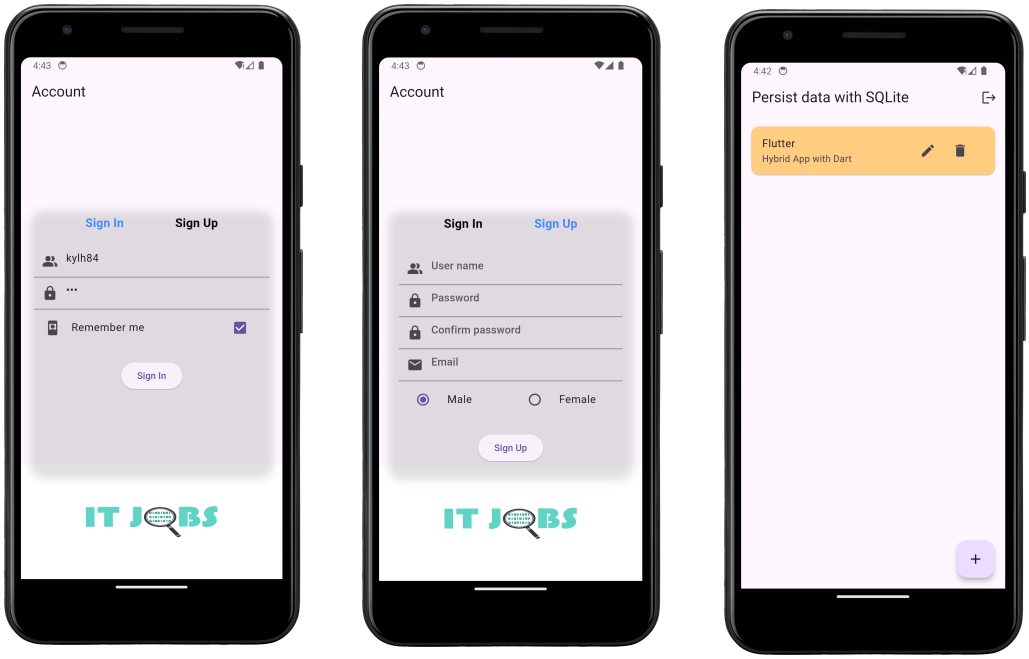


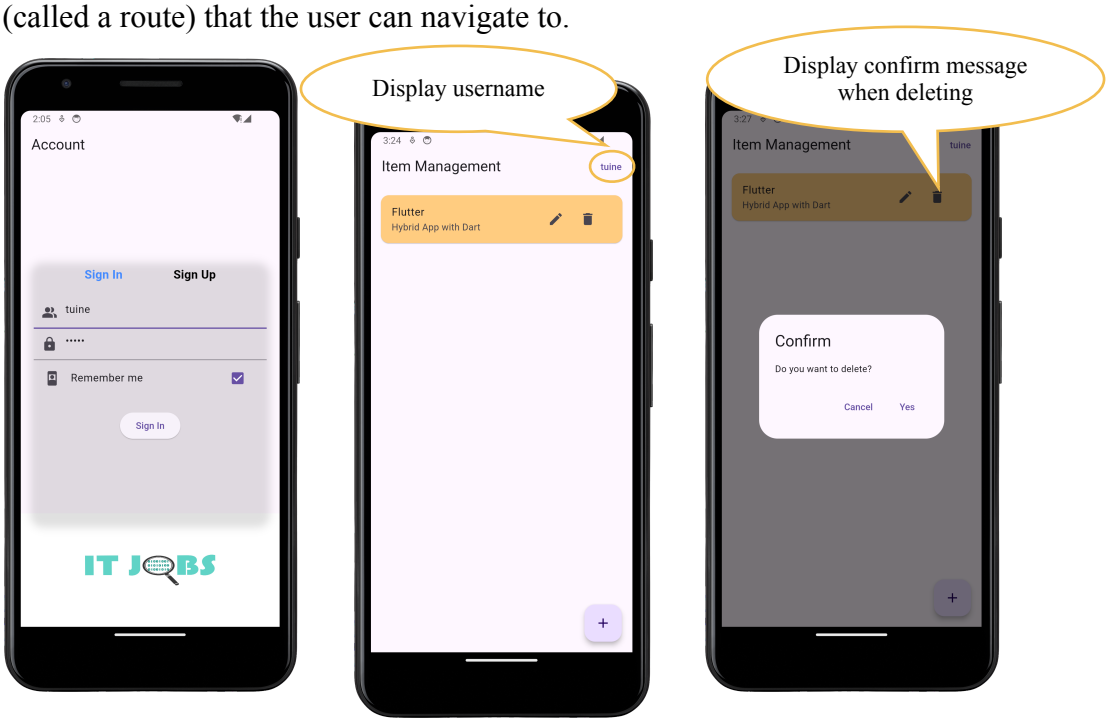
Table named **account**. Below is the structure of the table:

Column	Type	Description
username	TEXT	Primary key
password	TEXT	
email	TEXT	
gender	INTEGER	Boolean values are stored as integers 0 (female) and 1 (male)
createdAt	TIMESTAMP	The time that the item was created. It will be automatically added by SQLite

Using navigation and routing

Overview

In this practice, you'll extend a basic, mobile Flutter app to include interactivity. You'll also create a second page (called a route) that the user can navigate to.



Extra tasks (previous session): Add registration and login functions. After successful login, redirect to the homepage.

1. sql_helper.dart

```
1.  class SQLHelper {
2.    static const _databaseName = "MyDatabase.db";
3.    static const _databaseVersion = 1;
4.    static const _itemsTable = 'items';
5.    static const _accountTable = "accounts";
6.
7.    static const _columnId = 'id';
8.    static const _columnTitle = 'title';
9.    static const _columnDescription = 'description';
10.   static const _columnCreatedAt = 'createdAt';
11.
12.   // id: the id of a item
13.   // title, description: name and description of your activity
14.   // created_at: the time that the item was created.
15.   // It will be automatically handled by SQLite
16.   static Future<void> createItemTable(Database database) async {
17.     try {
18.       await database.execute('"
```

```
      CREATE TABLE $_itemsTable (
        $_columnId INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
        $_columnTitle TEXT,
        $_columnDescription TEXT,
        $_columnCreatedAt TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
      )''');
  } catch (err) {
    debugPrint("createItemTable(): $err");
  }
}

// id: the id of a item
// title, description: name and description of your activity
// created_at: the time that the item was created.
// It will be automatically handled by SQLite
static Future<void> createAccountTable(Database database) async {
  await database.execute('''CREATE TABLE $_accountTable(
      id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
      username TEXT,
      password TEXT,
      email TEXT,
      gender int,
      createdAt TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
    )''');
}

static Future<Database> getDb() async {
  return openDatabase(_databaseName, version: _databaseVersion,
      onCreate: (Database database, int version) async {
    await createItemTable(database);
    await createAccountTable(database);
  }, onUpgrade: (Database database, int oldVersion, int newVersion) async {});
}

// Create new item
static Future<int> createItem(Item item) async {
  int id = 0;

  try {
    final db = await SQLHelper.getDb();
```

```
59.
60.       // Sometimes you want to insert an empty row, in that case ContentValues
61.       // have no content value, and you should use nullColumnHack.
62.       // For example, you want to insert an empty row into a table student(id, name),
63.       // which id is auto generated and name is null.
64.       id = await db.insert(_itemsTable, item.toMap());
65.     } catch (err) {
66.       debugPrint("createItem(): $err");
67.     }
68.
69.     return id;
70.   }
71.
72.   // Read all items
73.   static Future<List<Map<String, dynamic>>> getItems() async {
74.     late Future<List<Map<String, dynamic>>> items;
75.     try {
76.       final db = await SQLHelper.getDb();
77.       items = db.query(_itemsTable, orderBy: _columnId);
78.     } catch (err) {
79.       debugPrint("getItems(): $err");
80.     }
81.
82.     return items;
83.   }
84.
85.   // Read a single item by id
86.   static Future<List<Map<String, dynamic>>> getItem(int id) async {
87.     late Future<List<Map<String, dynamic>>> item;
88.
89.     try {
90.       final db = await SQLHelper.getDb();
91.
92.       item = db.query(_itemsTable,
93.           where: "$_columnId = ?", whereArgs: [id], limit: 1);
94.     } catch (err) {
95.       debugPrint("getItem(): $err");
96.     }
97.
98.     return item;
```

```
99.   }
100.
101.  // Update an item by id
102.  static Future<int> updateItem(Item item) async {
103.    int result = 0;
104.    try {
105.      final db = await SQLHelper.getDb();
106.      result = await db.update(_itemsTable, item.toMap(),
107.          where: "$_columnId = ?", whereArgs: [item.id]);
108.    } catch (err) {
109.      debugPrint("updateItem(): $err");
110.    }
111.
112.    return result;
113.  }
114.
115.  // Delete a single item by id
116.  static Future<void> deleteItem(int id) async {
117.    try {
118.      final db = await SQLHelper.getDb();
119.      await db.delete(_itemsTable, where: "$_columnId = ?", whereArgs: [id]);
120.    } catch (err) {
121.      debugPrint("Something went wrong when deleting an item: $err");
122.    }
123.  }
124.
125.  // Create new an account
126.  static Future<int> createAccount(Account data) async {
127.    final db = await SQLHelper.getDb();
128.
129.    final id = await db.insert(_accountTable, data.toMap(),
130.        conflictAlgorithm: ConflictAlgorithm.replace);
131.
132.    return id;
133.  }
134.
135.  // Read all accounts
136.  static Future<List<Map<String, dynamic>>> getAccounts() async {
137.    final db = await SQLHelper.getDb();
138.
```

```
    return db.query(_accountTable, orderBy: "id");
  }

  // Read a single item by id
  // The app doesn't use this method but I put here in case you want to see it
  static Future<List<Map<String, dynamic>>> checkAccount(
      Account account) async {
    final db = await SQLHelper.getDb();

    return db.query(_accountTable,
        where: "username = ? and password = ?",
        whereArgs: [account.username, account.password],
        limit: 1);
  }
}
```

2. account_screen.dart

```
class AccountScreen extends StatelessWidget {
  const AccountScreen({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Account'),
        ),
        body: const _AccountPage(),
      ),
    );
  }
}

class _AccountPage extends StatefulWidget {
  const _AccountPage({super.key});

  @override
  State<_AccountPage> createState() => _AccountPageState();
}

```

```
enum GenderCharacter { male, female }

class _AccountPageState extends State<_AccountPage> {
  late bool isSignIn;
  bool isChecked = false;

  GenderCharacter _character = GenderCharacter.male;

  final _username = TextEditingController();
  final _password = TextEditingController();

  @override
  initState() {
    super.initState();
    isSignIn = true;
  }

  @override
  Widget build(BuildContext context) {
    Size size = MediaQuery.of(context).size;

    return Stack(children: [
      Column(
        mainAxisAlignment: MainAxisAlignment.end,
        children: [
          Container(
            height: size.height * 0.22,
            width: size.width,
            decoration: const BoxDecoration(
                image: DecorationImage(
                    image: AssetImage('images/it_job.png'), fit: BoxFit.fill)),
          ),
        ],
      ),
      Center(
        child: Container(
          width: size.width * 0.90,
          height: size.height * 0.5,
          decoration: BoxDecoration(
              borderRadius: BorderRadius.circular(20),
```

```
65.             boxShadow: const [
66.               BoxShadow(
67.                   color: Colors.black12, blurRadius: 10, spreadRadius: 5)
68.             ]),
69.           child: SingleChildScrollView(
70.             child: Column(
71.               mainAxisAlignment: MainAxisAlignment.center,
72.               children: [
73.                 Padding(
74.                   padding: const EdgeInsets.only(bottom: 20.0),
75.                   child: Row(
76.                     mainAxisAlignment: MainAxisAlignment.spaceEvenly,
77.                     children: [
78.                       GestureDetector(
79.                         onTap: () {
80.                           setState(() {
81.                             _username.clear();
82.                             _password.clear();
83.                             isSignIn = true;
84.                           });
85.                         },
86.                         child: Text(
87.                           'Sign In',
88.                           style: TextStyle(
89.                               fontSize: 18,
90.                               fontWeight: FontWeight.bold,
91.                               color:
92.                                   isSignIn ? Colors.blueAccent : Colors.black),
93.                         ),
94.                       ),
95.                       GestureDetector(
96.                         onTap: () {
97.                           setState(() {
98.                             _username.clear();
99.                             _password.clear();
100.                            isSignIn = false;
101.                          });
102.                        },
103.                        child: Text(
104.                          'Sign Up',
```

```
                    style: TextStyle(
                        fontSize: 18,
                        fontWeight: FontWeight.bold,
                        color:
                            !isSignIn ? Colors.blueAccent : Colors.black),
                  ),
                )
              ],
            ),
          ),
          isSignIn ? buildSignInPage() : buildSignUpPage(),
        ],
      ),
    ),
  ),
 )
]);
}

Widget buildSignInPage() {
 return Form(
   child: Column(
  children: [
   TextFormField(
     autofocus: true,
     controller: _username,
     decoration: const InputDecoration(
       hintText: 'User name', prefixIcon: Icon(Icons.people_alt)),
   ),
   TextFormField(
     obscureText: true,
     controller: _password,
     decoration: const InputDecoration(
       hintText: 'Password', prefixIcon: Icon(Icons.lock)),
   ),
   CheckboxListTile(
     title: const Text('Remember me'),
     value: isChecked,
     secondary: const Icon(Icons.remember_me),
     onChanged: (value) {
```

```
145.             setState(() {
146.               isChecked = value!;
147.             });
148.           }),
149.       Padding(
150.         padding: const EdgeInsets.only(top: 20.0),
151.         child: ElevatedButton(
152.             onPressed: () => _login(), child: const Text('Sign In')),
153.       )
154.     ],
155.   ));
156. }
157.
158. Widget buildSignUpPage() {
159.   final formKey = GlobalKey<FormState>();
160.   return Form(
161.       key: formKey,
162.       child: Container(
163.         padding: const EdgeInsets.all(10),
164.         child: Column(
165.           children: [
166.             TextFormField(
167.               autofocus: true,
168.               controller: _username,
169.               decoration: const InputDecoration(
170.                   hintText: 'User name', prefixIcon: Icon(Icons.people_alt)),
171.             ),
172.             TextFormField(
173.               obscureText: true,
174.               controller: _password,
175.               decoration: const InputDecoration(
176.                   hintText: 'Password', prefixIcon: Icon(Icons.lock)),
177.             ),
178.             TextFormField(
179.               obscureText: true,
180.               decoration: const InputDecoration(
181.                   hintText: 'Confirm password', prefixIcon: Icon(Icons.lock)),
182.             ),
183.             TextFormField(
184.               decoration: const InputDecoration(
```

```
185.                 hintText: 'Email', prefixIcon: Icon(Icons.email)),
186.             validator: (value) =>
187.                 isValidEmail(value!) ? null : "Invalid email format",
188.           ),
189.           Row(
190.             children: [
191.               Expanded(
192.                 child: RadioListTile(
193.                     title: const Text('Male'),
194.                     value: GenderCharacter.male,
195.                     groupValue: _character,
196.                     onChanged: (GenderCharacter? value) {
197.                       setState(() {
198.                         _character = value!;
199.                       });
200.                     }),
201.               ),
202.               Expanded(
203.                 child: RadioListTile(
204.                     title: const Text('Female'),
205.                     value: GenderCharacter.female,
206.                     groupValue: _character,
207.                     onChanged: (GenderCharacter? value) {
208.                       setState(() {
209.                         _character = value!;
210.                       });
211.                     }),
212.               ),
213.             ],
214.           ),
215.           Padding(
216.             padding: const EdgeInsets.only(top: 20.0),
217.             child: ElevatedButton(
218.                 onPressed: () => _register(formKey),
219.                 child: const Text('Sign Up')),
220.           )
221.         ],
222.       ),
223.     ));
224. }
```

```
225.
226.  Future<void> _login() async {
227.    var account = Account(username: _username.text, password: _password.text);
228.    // This function is used to fetch all data from the database
229.    final data = await SQLHelper.checkAccount(account);
230.
231.    if (!mounted) return;
232.
233.    if (data.isEmpty) {
234.      ScaffoldMessenger.of(context).showSnackBar(
235.          const SnackBar(content: Text('Invalid username or password')));
236.    } else {
237.      Navigator.push(
238.        context,
239.        MaterialPageRoute(builder: (context) => const ItemsScreen()),
240.      );
241.      setRemember(isChecked);
242.    }
243.  }
244.
245.  Future<void> _register(var formKey) async {
246.    if (formKey.currentState!.validate()) {
247.      var account = Account(username: _username.text, password: _password.text);
248.      final result = await SQLHelper.createAccount(account);
249.
250.      if (result > 0 && mounted) {
251.        ScaffoldMessenger.of(context)
252.            .showSnackBar(const SnackBar(content: Text('Successful')));
253.      }
254.    }
255.  }
256.
257.  bool isValidEmail(String email) {
258.    return RegExp(
259.            r"^[a-zA-Z0-9.a-zA-Z0-9.!#$%&'*+-/=?^_`{|}~]+@[a-zA-Z0-9]+\.[a-zA-Z]+")
260.        .hasMatch(email);
261.  }
262.}
```

Tasks

1. Modify **login** function in **_AccountPageState** class (attention to the **bold** lines)

```
Future<void> _login() async {
  var account = Account(username: _username.text, password: _password.text);
  // This function is used to fetch all data from the database
  final data = await SQLHelper.checkAccount(account);

  if (!mounted) return;

  if (data.isEmpty) {
    ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(content: Text('Invalid username or password')));
  } else {
    Navigator.push(
      context,
      MaterialPageRoute(
          builder: (context) => const ItemsScreen(),
          settings: RouteSettings(arguments: _username.text)),
    );
  }
}
```

2. Modify **ItemsScreen** class (attention to the **bold** lines)

```
class ItemsScreen extends StatelessWidget {
  const ItemsScreen({super.key});

  @override
  Widget build(BuildContext context) {
    String username = ModalRoute.of(context)?.settings.arguments as String;

    return MaterialApp(
      debugShowCheckedModeBanner: false,
      home: _HomePage(username: username),
    );
  }
}
```

3. Modify **_HomePage** class (attention to the **bold** lines)

```
class _HomePage extends StatefulWidget {
  const _HomePage({super.key, required this.username});

  final String username;

  @override
  State<_HomePage> createState() => _HomePageState();
}
```

4. Modify **_HomePageState** class (attention to the **bold** lines)

```
1.  Widget build(BuildContext context) {
2.    return Scaffold(
3.      appBar: AppBar(
4.        title: const Text('Item Management'),
5.        actions: [
6.          TextButton(
7.            onPressed: () => logout(),
8.            child: Text(widget.username),
9.          ),
10.       ],
11.     ),
12.     body: _isLoading
13.         ? const Center(
14.             child: CircularProgressIndicator(),
15.           )
16.         : ListView.builder(
17.             itemCount: _items.length,
18.             itemBuilder: (context, index) => Card(
19.               color: Colors.orange[200],
20.               margin: const EdgeInsets.all(15),
21.               child: ListTile(
22.                   title: Text(_items[index]['title']),
23.                   subtitle: Text(_items[index]['description']),
24.                   trailing: SizedBox(
25.                     width: 100,
26.                     child: Row(
27.                       children: [
28.                         IconButton(
29.                           icon: const Icon(Icons.edit),
30.                           onPressed: () => _showForm(_items[index]['id']),
31.                         ),
32.                         IconButton(
33.                           icon: const Icon(Icons.delete),
34.                           onPressed: () => _deleteItem(_items[index]['id']),
35.                         ),
36.                       ],
37.                     ),
38.                   )),
39.             ),
```

```
            ),
        floatingActionButton: FloatingActionButton(
          child: const Icon(Icons.add),
          onPressed: () => _showForm(null),
        ),
      );
    }
```

5. Create file named **confirmation_dialog.dart**

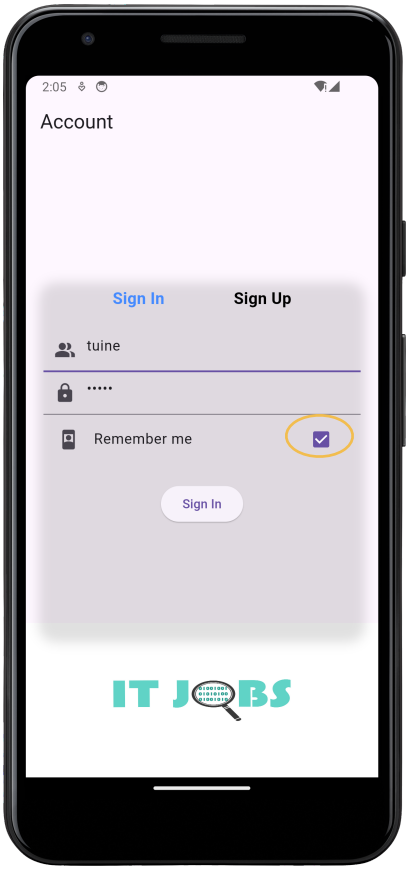
```
Future<void> showConfirmationDialog({
  required BuildContext context,
  required String title,
  required String content,
  required VoidCallback onConfirm,
}) {
  return showDialog<void>(
    context: context,
    barrierDismissible: false, // user must tap button
    builder: (BuildContext context) {
      return AlertDialog(
        title: Text(title),
        content: SingleChildScrollView(
          child: ListBody(
            children: <Widget>[
              Text(content),
            ],
          ),
        ),
        actions: <Widget>[
          TextButton(
            child: const Text('Cancel'),
            onPressed: () {
              Navigator.of(context).pop();
            },
          ),
          TextButton(
            child: const Text('Yes'),
            onPressed: () {
              Navigator.of(context).pop();
              onConfirm();
            },
          ),
        ],
      );
    },
  );
}
```

6. Modify **_deleteItem** function in _HomePageState class (attention to the **bold** lines)

```
Future<void> _deleteItem(int id) async {
  showConfirmationDialog(
    context: context,
    title: 'Confirm',
    content: 'Do you want to delete?',
    onConfirm: () async {
      await SQLHelper.deleteItem(id);
      // if you're not sure your widget is mounted.
      if (!mounted) return;

      ScaffoldMessenger.of(context).showSnackBar(const SnackBar(
        content: Text('Successfully deleted a item!'),
      ));

      _refreshItems();
    });
}
```

Extra tasks

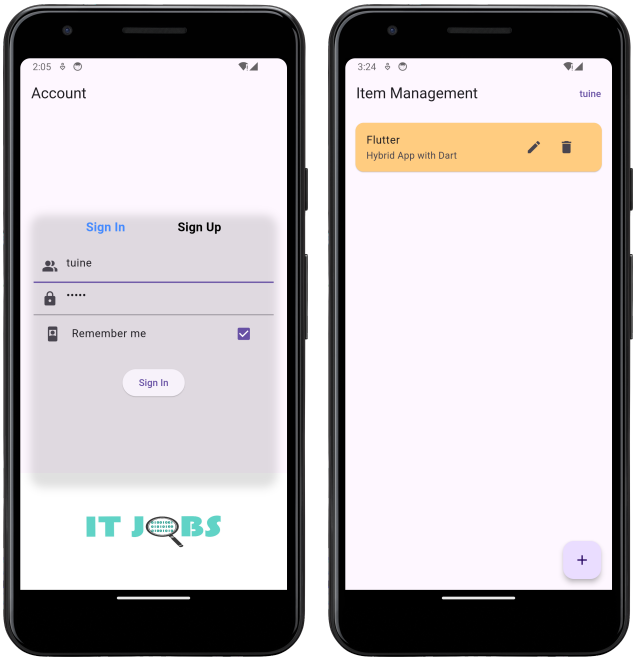
+ In this tasks, the "**Remember me**" option in the login interface of the mobile application is designed to provide users with a more convenient and seamless experience by allowing them to stay signed in on their device. When a user selects this option, the application remembers their login credentials, so they don't need to enter their username and password every time they open the app. This feature is particularly useful for users who frequently access the application, as it saves time and reduces the hassle of repeated logins.



Animations

Overview

In this updated code, createSlideTransition function is used to create SlideTransition to navigate to ItemsScreen after login



Tasks

+ Define the Slide Transition Function

Create a file named **transitions.dart**, and define the **createSlideTransition** function in it.

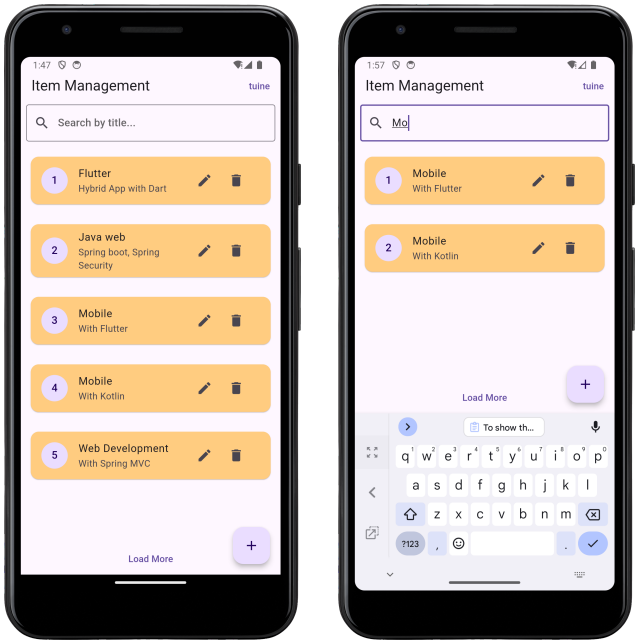
```
import 'package:flutter/material.dart';

PageRouteBuilder<dynamic> createSlideTransition(Widget newPage) {
 return PageRouteBuilder(
   pageBuilder: (context, animation, secondaryAnimation) => newPage,
   transitionsBuilder: (context, animation, secondaryAnimation, child) {
     const begin = Offset(1.0, 0.0); //The animation starts from the right side
     const end = Offset.zero; // The animation ends at the original position.
     const curve = Curves.easeInOut;

     var tween = Tween(begin: begin, end: end).chain(CurveTween(curve: curve));

     return SlideTransition(
       position: animation.drive(tween),
       child: child, // This is the newPage widget being transitioned
     );
   },
 );
}
```

+ Import and Use the Function

Modify **_login** function in account_page.dart (attention to the **bold** lines)

```
1.  Future<void> _login() async {
2.    var account = Account(username: _username.text, password: _password.text);
3.    // This function is used to fetch all data from the database
4.    final data = await SQLHelper.checkAccount(account);
5.
6.    if (!mounted) return;
7.
8.    if (data.isEmpty) {
9.      ScaffoldMessenger.of(context).showSnackBar(
10.         const SnackBar(content: Text('Invalid username or password')));
11.   } else {
12.     Navigator.push(
13.       context,
14.       createSlideTransition(const HomePage(), _username.text),
15.     );
16.     setRemember(isChecked);
17.   }
18. }
```

Extra tasks: Add search function based on title in ItemsScreen



Guideline:

To show the search bar within the AppBar of the ItemsScreen, you can incorporate a TextField inside the AppBar using a PreferredSize widget. Here's how you can do it:

1. Add a TextEditingController to handle the search input.
2. Add the _searchItems method to accept a search query and filter the items accordingly.
3. Add a search TextField inside the AppBar.

--THE END--