#4 Setting up the Flutter Project and Connect to the Supabase API

Introduction

In this section, we will cover the steps to set up a Flutter project for the educative.io clone app. This includes adding dependencies and assets, configuring environment variables, and connecting to the Supabase API using the `supabase_flutter` package. Additionally, we will set up the Riverpod state management.

Once these steps are complete, we will be ready to start building our educative.io clone app. Let's get started! 🚀

Project structure

```
educative_clone_app/
├── ...
├── assets/
│   ├── lessons/
│   └── ...
├── lib/
│   ├── controllers/
│   ├── data/
│   ├── models/
│   ├── pages/
│   ├── themes/
│   ├── widgets/
```

```
│   └── main.dart
└── ...
```

The project structure for our Flutter application is shown above.

- The `lib` directory contains the source code for our application, organized into different directories based on the purpose of the code.
- The `pages` directory contains the UI pages of our application.
- The `widgets` directory contains reusable widgets that can be used on multiple pages.
- The `themes` directory contains the theme data for our application like typography and colors.
- The `models` directory contains data models used throughout the application.
- The `controllers` directory contains business logic for our application.
- The `data` directory contains mock or sample data that is used for testing or demonstration purposes.
- The `assets` directory contains the assets used in our application, such as images and data files.
- The `assets/lessons` directory contains the lesson files in markdown format. To save a lesson file in markdown format, we can create a new file with a `.md` extension and save it in the `assets/lessons` directory. We can then load this file in our Flutter application using the `flutter_markdown` package.
- The `main.dart` file is the entry point for our Flutter application.

To set up the project folder, you can follow this video:

> https://www.loom.com/share/ff5dfd8fe85e4b819a43922b5bb6d9c9

Adding Dependencies

To add dependencies to our Flutter project, we need to update the `pubspec.yaml` file that is located in the root directory of our project. This file is used to list the dependencies that our project needs to run. To add a dependency, we need to specify the package name and the version number.

Here are some commonly used dependencies that we can add to our project:

- `any_link_preview: ^3.0.0`: This package provides an easy way to extract metadata from a URL, such as title, description, and image.
- `cupertino_icons: ^1.0.2`: This package provides the Cupertino icons that are used in iOS-style Flutter applications.
- `flutter: sdk: flutter`: This is the SDK for Flutter.
- `flutter_dotenv: ^5.0.2`: This package provides a way to load environment variables from a `.env` file.
- `flutter_markdown: ^0.6.14`: This package provides a way to display Markdown text in our Flutter application.
- `flutter_riverpod: ^2.3.2`: This package provides a simple way to manage state in our Flutter application.
- `flutter_svg: ^2.0.4`: This package provides a way to display SVG images in our Flutter application.
- `flutter_syntax_view: ^4.0.0`: This package provides a way to display code syntax highlighting in our Flutter application.
- `flutter_widget_from_html: ^0.10.0`: This package provides a way to display HTML content in our Flutter application.
- `google_fonts: ^4.0.3`: This package provides a way to use custom fonts in our Flutter application.
- `readmore: ^2.2.0`: This package provides a way to display a "Read More" link for long text.
- `shimmer: ^2.0.0`: This package provides a way to display a shimmering effect in our Flutter application.
- `supabase_flutter: ^1.5.0`: This package provides a way to use the Supabase API in our Flutter application.

- `url_launcher: ^6.1.10`: This package provides a way to launch URLs in our Flutter application.

We can add these dependencies to our project by specifying the package name and version number in the `dependencies` section of the `pubspec.yaml` file. For example, to add the `flutter_markdown` package, we can add the following line to the `dependencies` section:

```
flutter_markdown: ^0.6.14
```

Once we have added the dependencies to the `pubspec.yaml` file, we can run the `flutter packages get` command in our terminal or in the IDE terminal to download the packages and update our project.

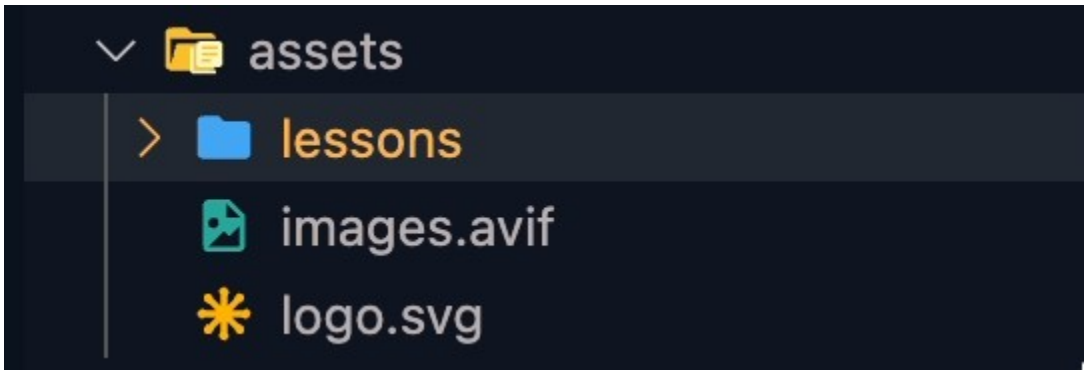
Adding assets

To download the assets for this Flutter project, please follow the URL below:

```
https://github.com/sahibul-nf/educative_clone_guide/raw/main/assets/assets.zip
```

This URL will take you to a page where you can download a `.zip` file containing all the assets for this project. Once you have downloaded the file, you can extract the contents to the `assets` directory in the root directory of your Flutter project.

After downloading and extracting the assets, the `assets` directory in the root directory of your Flutter project should contain a `lessons` subdirectory, which contains the lesson files in markdown format, and any other asset files needed for the project.



The `assets` directory is where we store all the assets for our Flutter application, such as images and data files. It is important to declare the assets in the `pubspec.yaml` file, so that Flutter knows to include them in the application. To do this, we can add the following lines to the `pubspec.yaml` file:

```
flutter:
  assets:
    - assets/
    - assets/lessons/
```

These lines tell Flutter to include all files in the `assets/` and `assets/lessons/` directories as assets in our application. Once we have declared the assets in the `pubspec.yaml` file, we can load the assets in our Flutter application using the `rootBundle` object provided by the `flutter/src/services/asset_bundle.dart` package, or using the `NetworkAssetBundle` object to load assets from a remote URL. The `rootBundle` object is used to load assets that are bundled with our application, while the `NetworkAssetBundle` object is used to load assets from a remote server.

Setup .env file

To store sensitive information like Supabase URL and Supabase Anon Key, we will create a `.env` file in the root directory of our project. This file will contain key-value pairs for the environment variables that we want to use in our Flutter application.

To create the `.env` file, we can use a text editor to create a new file in the root directory of our project and name it `.env`.

In the `.env` file, we will define the following environment variables:

```
SUPABASE_URL=<your_supabase_url>
SUPABASE_ANON_KEY=<your_supabase_anon_key>
```

Replace `<your_supabase_url>` with the URL of your Supabase project, and `<your_supabase_anon_key>` with your Supabase Anon Key.

Get Supabase URL and Supabase Anon Key

To get the Supabase URL and Supabase Anon Key, we need to go to our Supabase project dashboard and click on the `Project Settings` tab. In the `API` section, we can find our Supabase URL and Anon Key, which we can copy and paste into our `.env` file. It is important to keep these values private and not share them with anyone, as they provide access to our Supabase project.

https://www.loom.com/share/15e44ffe8c7f4550b9d3d8a38c6cc23e

Configure .env file in pubspec.yaml

Make sure that the `flutter_dotenv` package is included in the `pubspec.yaml` file, so that we can use the environment variables defined in the `.env` file in our Flutter application. The `pubspec.yaml` file should include the following lines:

```
dependencies:
  flutter_dotenv: ^5.0.2  # Add this line

flutter:
  uses-material-design: true

  # Assets
  assets:
    - .env  # Add this line
    - assets/
    - assets/lessons/
```

These lines tell Flutter to include the `flutter_dotenv` package as a dependency in our project and to include the `.env` file as an asset so that the environment variables can be accessed.

Configure .env file in .gitignore (Optional)

To prevent the sensitive information in the `.env` file from being pushed to the repository, we need to add the `.env` file to the `.gitignore` file.

Add the following line to the `.gitignore` file:

```
.env
```

This line tells Git to ignore the `.env` file when committing changes.

Configure Flutter code to connect with Supabase

To use the `supabase_flutter` package in your Flutter project, ensure that it is included as a dependency in your `pubspec.yaml` file. To do this, add the following line to the `dependencies` section:

```
supabase_flutter: ^1.5.0
```

This will enable you to use the package in your project.

Once we have added the `supabase_flutter` package to our project, we can import the `supabase_flutter.dart` file in our `main.dart` file using the `import` statement:

```
import 'package:supabase_flutter/supabase_flutter.dart';
```

After importing the `supabase_flutter.dart` file, we can initialize the `Supabase` object with the Supabase URL and Anon Key environment variables defined in the `.env` file. To do this, we can use the `Supabase.initialize` method:

```
import 'package:flutter_dotenv/flutter_dotenv.dart';
import 'package:supabase_flutter/supabase_flutter.dart';

void main() async {
  await dotenv.load(fileName: ".env");

  final supabaseUrl = dotenv.get('SUPABASE_URL');
  final supabaseAnonKey = dotenv.get('SUPABASE_ANON_KEY');

  WidgetsFlutterBinding.ensureInitialized();

  await Supabase.initialize(
    url: supabaseUrl,
    anonKey: supabaseAnonKey,
  );

  runApp(const MyApp());
}
```

In the code above, we first ensure that the Flutter bindings are initialized by calling `WidgetsFlutterBinding.ensureInitialized()`. We then load the environment variables from the `.env` file using the `dotenv.load(fileName: ".env")` method. Next, we get the Supabase URL and Anon Key environment variables from the `dotenv.get('<key>')` and store them in two variables, `supabaseUrl` and `supabaseAnonKey`. Finally, we initialize the `Supabase` object using the `Supabase.initialize` method with the Supabase URL and Anon Key environment variables, and then we run our Flutter application.

After setting up the `Supabase` object, we can use it to interact with the Supabase API in our Flutter application.

Setup Flutter Riverpod Package

To use the `flutter_riverpod` package, we need to wrap our `MaterialApp()` widget with the `ProviderScope()` widget to enable state management with `flutter_riverpod`. This allows us to use the `Provider()` and `Consumer()` widgets to manage and access the state throughout our application.

The `main()` function in the `main.dart` file will look something like this:

```
import 'package:flutter_riverpod/flutter_riverpod.dart';   // import this

void main() async {
  // ...

  runApp(
    ProviderScope(
      child: const MyApp(),
    ),
  );
}
```

With `flutter_riverpod` set up, we can start using it to manage the state in our Flutter application.

Conclusion

Congratulations on successfully setting up our Flutter project, adding dependencies and assets, configuring environment variables using a `.env` file, and using the `supabase_flutter` package to connect with the Supabase API. We have also set up the Riverpod state management.

Now, we are ready to start building our educative.io clone app! Happy coding! 🚀

Full `main.dart` code:

```
import 'package:flutter/material.dart';
import 'package:flutter_dotenv/flutter_dotenv.dart';
import 'package:flutter_riverpod/flutter_riverpod.dart';
import 'package:supabase_flutter/supabase_flutter.dart';

import 'pages/home_page.dart';

void main() async {
  await dotenv.load(fileName: ".env");

  final supabaseUrl = dotenv.get('SUPABASE_URL');
  final supabaseAnonKey = dotenv.get('SUPABASE_ANON_KEY');

  WidgetsFlutterBinding.ensureInitialized();

  await Supabase.initialize(
    url: supabaseUrl,
    anonKey: supabaseAnonKey,
  );

  runApp(
    ProviderScope(
      child: const MyApp(),
    ),
  );
}

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

  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'Educative',
    );
  }
}
```

Full `pubspec.yaml` code:

```
name: educative_app_clone
description: A new Flutter project.

publish_to: "none" # Remove this line if you wish to publish to pub.dev

version: 1.0.0+1

environment:
  sdk: ">=2.19.2 <3.0.0"

dependencies:
  any_link_preview: ^3.0.0
  cupertino_icons: ^1.0.2
  flutter:
    sdk: flutter
  flutter_dotenv: ^5.0.2
  flutter_markdown: ^0.6.14
  flutter_riverpod: ^2.3.2
  flutter_svg: ^2.0.4
  flutter_syntax_view: ^4.0.0
  flutter_widget_from_html: ^0.10.0
  google_fonts: ^4.0.3
  readmore: ^2.2.0
  shimmer: ^2.0.0
  supabase_flutter: ^1.5.0
  url_launcher: ^6.1.10

dev_dependencies:
  flutter_lints: ^2.0.0
  flutter_test:
    sdk: flutter

flutter:
  uses-material-design: true

  assets:
    - .env
    - assets/
    - assets/lessons/
```