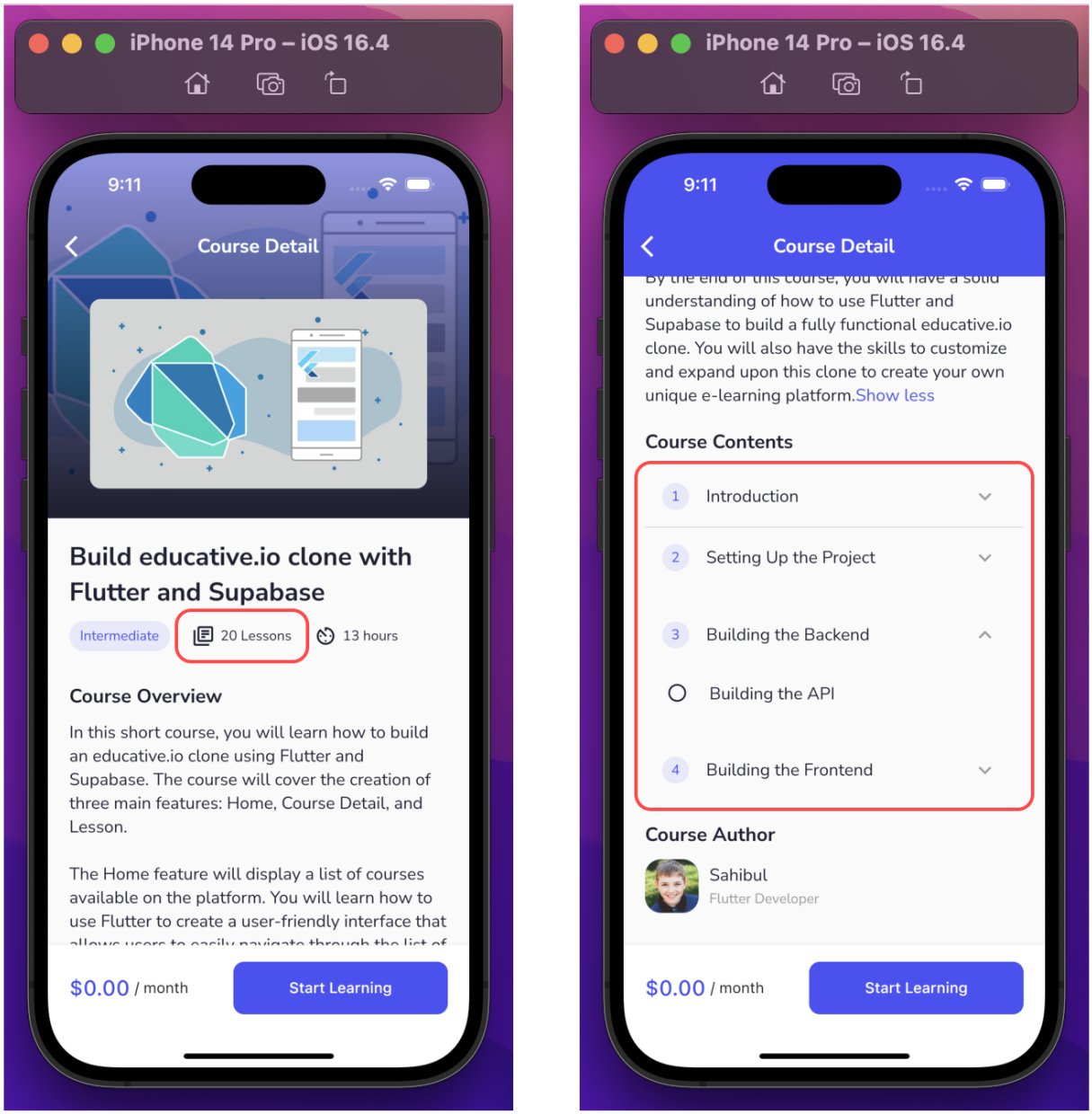
#11 Fetch Data from Supabase to Display in Course Detail Page

Introduction

In this section, we will discuss how to fetch lesson data from Supabase to display the number of lessons and the course content on the Course Detail Page of our app. The way probably will be the same as the previous section. So, let’s do it!



Implementation - Step-by-step guide

Same as the previous section, before we define the function to retrieve lesson data from Supabase, firstly we need to insert data into the Supabase database that we already created before. Specifically, we will insert data for the `lesson` and the `lesson_child` table in this step.

Step 1: Insert data into the `lesson` and the `lesson_child` table

Please follow the link below to download the lesson and the lesson_child data in a CSV file.

lesson_rows.csv

lesson_child_rows.csv

After downloading the file, we need to insert the lesson data into the `lesson` table in the Supabase Database. To see how to do this, please watch the video below on inserting data from a CSV file to the `lesson` table in the Supabase Database.

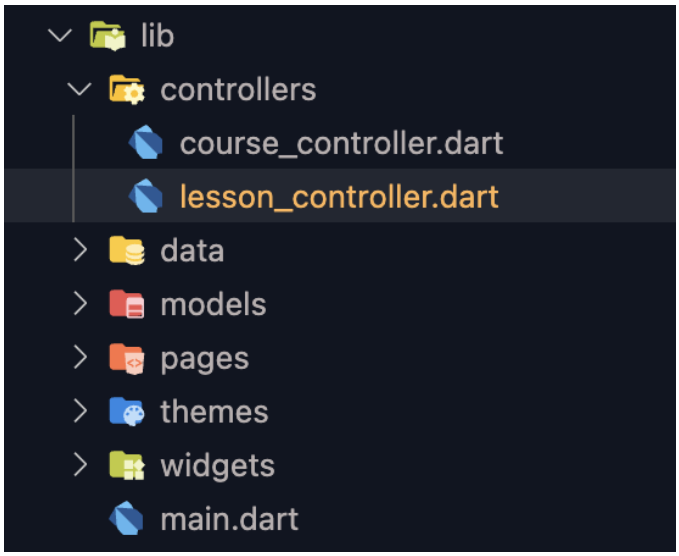
https://www.loom.com/share/7e1142cee3c84401a552010d3274cda8?sid=95985311-2a2b-44ef-acf2-b64eab4ccb0e

After inserting the lesson data into the `lesson` table, do the same to insert the lesson_child data into the `lesson_child` table. You can follow the same way as in the video above.

Okay, next we will define a function that retrieves lesson data from the Database. To achieve this follow the following steps.

Step 2: Create a `lesson_controller.dart` file

Inside the `lib/controller` folder create a new file called `lesson_controller.dart`.



Step 3.1: Retrieve Lesson data from the Database

Create a class called `LessonController` inside the `lesson_controller.dart` file that we created before the step. And inside the `LessonController` class, we define a function that handles retrieving lesson data from the Supabase database.

Add the following code snippet below into the `lesson_controller.dart` file:

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

import '../models/lesson.dart';

class LessonController {
  final supabase = Supabase.instance.client;

  Future<List<Lesson>> getLessonsByCourseId(String courseId) async {
    final response = await supabase
        .from('lesson')
        .select('*, lesson_child!inner(*)')
        .eq('course_id', courseId);

    print("response: $response");

    final lessons = response
        .map((lesson) => Lesson.fromJson(lesson))
        .toList()
        .cast<Lesson>();

    print("lessons: ${lessons.length}");

    return lessons;
```

```
  }
}
```

The code snippet shows a function defined in a class named `LessonController`. This class contains a method called `getLessonsByCourseId` which takes a string `courseId` as input and returns a list of `Lesson` objects.

Inside the function, the Supabase client is used to query the `lesson` table in the database, filtering the results based on the `course_id` field matching the input `courseId`. The `select` method retrieves all fields from the `lesson` table and all related records from the `lesson_child` table using the `inner` modifier. The response is then mapped to a list of `Lesson` objects using the `fromJson` method and returned.

After we define the function to retrieve data from the Supabase database, we next need to define a provider that will use in the UI to show the course contents.

Here's the code snippet of the provider, add the following code snippet below into the `lesson_controller.dart` file:

```
final lessonProvider = FutureProvider.autoDispose.family<List<Lesson>, String>(
  (ref, courseId) async {
    final lessonController = LessonController();
    final lessons = await lessonController.getLessonsByCourseId(courseId);

    return lessons;
  },
);
```

The code snippet above defines a `FutureProvider` named `lessonProvider`, which is a family provider that takes a `String` input `courseId` and returns a `List` of `Lesson` objects.

The provider uses the `autoDispose` modifier, which automatically disposes of the provider when it is no longer needed. Inside the provider definition, an instance of `LessonController` is created, and the `getLessonsByCourseId` method is called to retrieve lessons from the Supabase database based on the input `courseId`.

The retrieved lessons are then returned as the value of the provider. This provider can be used in the UI to display the course contents for a specific course in `CourseDetailPage`.

And don't forget to import the `FutureProvider` widget from the `flutter_riverpod` package, it should be imported at the top of the file along with the other necessary imports.

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

The populate the `lesson_controller.dart` file will look something like this:

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

import '../models/lesson.dart';

final lessonProvider = FutureProvider.autoDispose.family<List<Lesson>, String>(
  (ref, courseId) async {
    final lessonController = LessonController();
    final lessons = await lessonController.getLessonsByCourseId(courseId);

    return lessons;
  },
);

class LessonController {
  final supabase = Supabase.instance.client;

  Future<List<Lesson>> getLessonsByCourseId(String courseId) async {
    final response = await supabase
        .from('lesson')
        .select('*, lesson_child!inner(*)')
        .eq('course_id', courseId);

    print("response: $response");

    final lessons = response
        .map((lesson) => Lesson.fromJson(lesson))
        .toList()
        .cast<Lesson>();

    print("lessons: ${lessons.length}");

    return lessons;
  }
}
```

Step 3.2: Display Course Contents Data on the UI

In this step, we will display the course content data on the UI/widget of the `CourseDetailPage`. To achieve this, we must first define a variable that will listen for changes to `lessonProvider`.

Inside the `course_detail_page.dart` file, add the following code snippet to the `build` method of the `CourseDetailPage` widget, specifically before returning the widget.

```
final lessonState = ref.watch(lessonProvider(widget.course.id));
```

And don't forget to import `lessonProvider` at the top of the file along with the other necessary imports.

```
import '../controllers/lesson_controller.dart';
```

Afterward, we can use the `lessonState` variable that we defined earlier to build a widget. Update the `buildCourseContents` method of the `CourseDetailPage` widget with the code snippet provided below:

```
Widget buildCourseContents(AsyncValue<List<Lesson>> lessonState) {
    return lessonState.when(
      data: (lessons) {
        return ExpansionPanelList(
          elevation: 0,
          expandedHeaderPadding: EdgeInsets.zero,
          expansionCallback: (panelIndex, isExpanded) {
            setState(() {
              lessons[panelIndex].showDetail = !isExpanded;
            });
          },
          children: lessons.map<ExpansionPanel>((Lesson lesson) {
            return ExpansionPanel(
              isExpanded: lesson.showDetail,
              canTapOnHeader: true,
              backgroundColor: Colors.transparent,
              headerBuilder: (context, isExpanded) {
                int index = lessons.indexOf(lesson) + 1;

                return ClipRRect(
                  child: ListTile(
                    leading: ClipRRect(
                      borderRadius: BorderRadius.circular(15),
                      child: Container(
                        color: MyColors.primary.withOpacity(0.1),
                        height: 25,
                        width: 25,
                        alignment: Alignment.center,
                        child: Text(
                          index.toString(),
```

```
                        style: MyTypography.bodySmall.copyWith(
                          color: MyColors.primary,
                        ),
                      ),
                    ),
                  ),
                  minLeadingWidth: 0,
                  title: Text(
                    lesson.title,
                    style: MyTypography.body,
                  ),
                ),
              );
            },
            body: ListView.separated(
              padding: const EdgeInsets.symmetric(
                horizontal: 20,
                vertical: 16,
              ),
              shrinkWrap: true,
              physics: const NeverScrollableScrollPhysics(),
              itemCount: lesson.lessons.length,
              separatorBuilder: (context, index) {
                return const Divider();
              },
              itemBuilder: (context, index) {
                return Row(
                  crossAxisAlignment: CrossAxisAlignment.start,
                  children: [
                    if (lesson.lessons[index].isCompleted == true)
                      const Icon(
                        Icons.check_circle,
                        size: 20,
                        color: Colors.green,
                      )
                    else
                      Icon(
                        Icons.radio_button_unchecked,
                        size: 20,
                        color: MyColors.black,
                      ),
                    const SizedBox(width: 20),
                    Expanded(
                      child: Text(
                        lesson.lessons[index].title,
                        style: MyTypography.body,
                      ),
                    ),
                  ],
                );
              },
            ),
          );
        }).toList(),
```

```
        );
      },
      loading: () => Shimmer.fromColors(
        baseColor: Colors.grey[300]!,
        highlightColor: Colors.grey[100]!,
        child: Container(
          height: 30,
          width: double.infinity,
          decoration: BoxDecoration(
            color: Colors.white,
            borderRadius: BorderRadius.circular(10),
          ),
        ),
      ),
      error: (error, stackTrace) => const Center(
        child: Text('Error'),
      ),
    );
  }
```

And don't forget to import the `shimmer` package into the `course_detail_page.dart` file:

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

Additionally, we should update a bit of code inside the `build` method of the `CourseDetailPage`. Specifically, that's in the **Course Content** section. So, the updated code will look something like this:

```
          // Course Content
          SliverToBoxAdapter(
            child: Padding(
              padding: const EdgeInsets.symmetric(
                horizontal: 20,
                vertical: 10,
              ),
              child: Column(
                crossAxisAlignment: CrossAxisAlignment.start,
                children: [
                  Text(
                    'Course Contents',
                    style: MyTypography.titleSmall,
                  ),
                  const SizedBox(height: 10),
                  buildCourseContents(lessonState),  // <-- Updated code
                ],
              ),
```

```
            ),
          ),
```

Okay, if you successfully follow the steps so far, the result will look like the following video:

> https://www.loom.com/share/051434f524194832892d9413ddf67998?sid=1a73f0c5-6467-463f-a294-e2b5f9a1a6f8

Step 4.1: Retrieve Lesson Child data from the Database

To retrieve data for the `lesson_child` table, we will create a new function inside the `LessonController` class that will specifically retrieve the data from the `lesson_child` table. We'll name this function `getLessonChildByLessonId`.

Here's the code for the new function:

```
// Get all lesson child by course id
Future<List<LessonChild>> getLessonChildByLessonId(String courseId) async {
  final response = await supabase
      .from('lesson')
      .select('*, lesson_child!inner(*)')
      .eq('course_id', courseId)
      .order('created_at', ascending: true);

  // debugPrint("response: $response");

  final lessons = response
      .map((lesson) => Lesson.fromJson(lesson))
      .toList()
      .cast<Lesson>();

  final lessonChilds = <LessonChild>[];

  for (var lesson in lessons) {
    lessonChilds.addAll(lesson.lessons);
  }

  print("lessonChild: ${lessonChilds.length}");

  return lessonChilds;
}
```

The function uses the Supabase client to query the `lesson_child` table in the database, filtering the results based on the `lesson_id` field matching the input `lessonId`. The `select` method is used to retrieve all fields from the `lesson_child` table. The response is then mapped to a list of `LessonChild` objects using the `fromJson` method and returned.

After we define the function to retrieve data from the Supabase database, we next need to define a provider that will use in the UI to show the number of lessons.

Here's the code snippet of the provider, add the following code snippet below into the `lesson_controller.dart` file:

```
final lessonChildProvider =
    FutureProvider.autoDispose.family<List<LessonChild>, String>(
  (ref, lessonId) async {
    final lessonController = LessonController();
    final lessonChild =
        await lessonController.getLessonChildByLessonId(lessonId);

    return lessonChild;
  },
);
```

This code defines a `FutureProvider` named `lessonChildProvider` which takes a `String` input `lessonId` and returns a `List` of `LessonChild` objects. The provider uses the `autoDispose` modifier, which automatically disposes of the provider when it is no longer needed.

Inside the provider definition, an instance of `LessonController` is created, and the `getLessonChildByLessonId` method is called to retrieve data from the Supabase database based on the input `lessonId`. The retrieved data is then returned as the value of the provider.

We can then use this provider to update the UI as needed.

The populate the `lesson_controller.dart` file, so far will look something like this:

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

import '../models/lesson.dart';

final lessonProvider = FutureProvider.autoDispose.family<List<Lesson>, String>(
  (ref, courseId) async {
    final lessonController = LessonController();
```

```
    final lessons = await lessonController.getLessonsByCourseId(courseId);

    return lessons;
  },
);

final lessonChildProvider =
    FutureProvider.autoDispose.family<List<LessonChild>, String>(
  (ref, lessonId) async {
    final lessonController = LessonController();
    final lessonChild =
        await lessonController.getLessonChildByLessonId(lessonId);

    return lessonChild;
  },
);

class LessonController {
  final supabase = Supabase.instance.client;

  Future<List<Lesson>> getLessonsByCourseId(String courseId) async {
    final response = await supabase
        .from('lesson')
        .select('*, lesson_child!inner(*)')
        .eq('course_id', courseId);

    print("response: $response");

    final lessons = response
        .map((lesson) => Lesson.fromJson(lesson))
        .toList()
        .cast<Lesson>();

    print("lessons: ${lessons.length}");

    return lessons;
  }

    // Get all lesson child by course id
  Future<List<LessonChild>> getLessonChildByLessonId(String courseId) async {
    final response = await supabase
        .from('lesson')
        .select('*, lesson_child!inner(*)')
        .eq('course_id', courseId)
        .order('created_at', ascending: true);

    // debugPrint("response: $response");

    final lessons = response
        .map((lesson) => Lesson.fromJson(lesson))
        .toList()
        .cast<Lesson>();

    final lessonChilds = <LessonChild>[];
```

```
    for (var lesson in lessons) {
      lessonChilds.addAll(lesson.lessons);
    }

    print("lessonChild: ${lessonChilds.length}");

    return lessonChilds;
  }
}
```

Step 4.2: Display the Number of Lesson Data on the UI

In this step, we will display the number of lessons in the course on the UI/widget of the `CourseDetailPage`. To achieve this, we must first define a variable that will listen for changes to `lessonChildProvider`.

Inside the `course_detail_page.dart` file, add the following code snippet to the `build` method of the `CourseDetailPage` widget, specifically before returning the widget.

```
final lessonChildState = ref.watch(lessonChildProvider(widget.course.id));
```

Afterward, we can use the `lessonChildState` variable that we defined earlier to build a widget.

Update code inside the `build` method of the `CourseDetailPage`. Specifically, that's in the **Course Info** section. So, the updated code will look something like this:

```
          // Course Info
          SliverToBoxAdapter(
            child: Padding(
              padding: const EdgeInsets.symmetric(
                horizontal: 20,
                vertical: 20,
              ),
              child: Column(
                children: [
                  Text(
                    widget.course.title,
                    style: MyTypography.title,
                  ),
                  const SizedBox(height: 10),
                  Row(
                    children: [
                      Wrap(
```

```
                        crossAxisAlignment: WrapCrossAlignment.center,
                        direction: Axis.horizontal,
                        spacing: 20,
                        runSpacing: 10,
                        children: [
                          ClipRRect(
                            borderRadius: BorderRadius.circular(15),
                            child: Container(
                              color: MyColors.primary.withOpacity(0.1),
                              padding: const EdgeInsets.symmetric(
                                horizontal: 10,
                                vertical: 5,
                              ),
                              child: Text(
                                widget.course.level,
                                style: MyTypography.bodySmall.copyWith(
                                  color: MyColors.primary,
                                ),
                              ),
                            ),
                          ),
                          lessonChildState.when(
                            data: (lessonChild) {
                              return buildCourseInfo(
                                Icons.library_books_outlined,
                                "${lessonChild.length} Lessons",
                              );
                            },
                            error: (error, stackTrace) => const SizedBox(),
                            loading: () => Shimmer.fromColors(
                              baseColor: Colors.grey[300]!,
                              highlightColor: Colors.grey[100]!,
                              child: Container(
                                height: 25,
                                width: 100,
                                decoration: BoxDecoration(
                                  color: Colors.white,
                                  borderRadius: BorderRadius.circular(10),
                                ),
                              ),
                            ),
                          ),
                          buildCourseInfo(
                            Icons.av_timer,
                            widget.course.duration,
                          ),
                        ],
                      ),
                    ],
                  ),
                ],
              ),
            ],
          ),
        ),
      ),
```

Okay, if you successfully follow the steps so far, the result will look like the following video:

https://www.loom.com/share/8a5ad1bc14114804a594e17e44a97618?sid=d3b3ba09-724c-4886-83e8-821997520bfa

Conclusion

In this section, we learned how to fetch data from a Supabase database and display it in our educative.io clone app. We created a `LessonController` class to handle CRUD operations for the `lesson` and `lesson_child` tables in the database. We also used the `flutter_riverpod` package to manage the state and created a `FutureProvider` to retrieve data from the database.

Happy coding!