Name: Rui Hu
Student id number: 3623329

Introduction

A Flutter-based Quiz App focused on propositional logic, designed to simplify learning for users. This application offers a range of quizzes on different topics in propositional logic. Users can choose options to answer questions, and for each question, there is an explanation available for reference.

The app includes a statistics page that provides a snapshot of the user's daily progress, accuracy, and volume of completed quizzes, represented both in linear charts and heatmaps. It allows users to visually track their learning progress over days and weeks easily.

Users can upload questions/statistics to the server or download new questions. All server data can be viewed via HTML, and questions can be edited, added, or deleted from the HTML web page.

Licenses: MIT Licenses, Apache License 2.0.

Overview

	Feature	Description	Mandatory
Frontend	Questions	provide propositional questions to users	•
	Options	Allows users to select from multiple answer options for each question.	•
	Explanation	Presents a detailed explanation of the correct answer after a response is	•
	Timer Progress Indicator	Visually displays the amount of time elapsed while answering a question.	
	Dark mode	Enables users to switch the app theme to a darker color palette, reducing eye strain in	
	Statistics	Analyzes and visualizes user performance data through interactive charts and graphs.	•
	User guide	At the first time launching the app, highlight the widgt step by step to guide	•
Backend	Html: show statistics	Show all stastistics/database via html	•
	html: add/delete/edit questions	could add/delete/edit questions in html	•
	Rest API	Communication between front-end and back-end	•
	Persist all data	Use MySQL to persist all data	
Generator	Generator	Automatically create tasks for the user.	

Setting up the project

1. **Install Flutter and Dart Plugins:**
   Download and install Flutter and Dart. Run `flutter doctor` to check environment.

2. **Configure Flutter SDK:**
   Set up Flutter SDK in Android Studio by navigating to Settings > Languages & Frameworks > Flutter and pointing to the right directory.
3. **Open Flutter Project:**
   Open quiz_app project from the Welcome screen of Android Studio.
4. **Get Dependencies:**
   Run the `flutter pub get` command in the terminal inside Android Studio to fetch all the required dependencies specified in the `pubspec.yaml` file.
5. **Set Up and an Emulator:**
   Go to Tools >Device manager and create a new Virtual Device or choose an existing one, configured for Android as needed. The device i used is `Pixel 3a API 33 x86 64` . Then launch the emulator by clicking the play button under actions column in device manager.
6. **Run/ Compile the App:**
   Once the emulator is running, click the ‘Run’ / 'Debug' button in Android Studio to compile and launch the app on the emulator.

Brief Documentation

Frontend

Question Card

Fetching questions from local

After the first time build questions card widget, it will get the uncompleted questions' id from database using `questionIdProvider` .

```
final questionsIdProvider = FutureProvider<List<int>?>((ref) async {
  return await ref.watch(dataBaseProvider).getUnansweredQuestions();
});

final AsyncValue<List<int>?> asyncValue = ref.watch(questionsIdProvider);
```

Show current question number / total questions

To construct a widget showing "current question number / total questions," use a `FutureBuilder` . Use `questionIndexProvider` for the current question index and

`getAmount()` from `databaseProvider` for the total number of unfinished questions.

Timer Progress Indicator

Utilize `LinearProgressIndicator` to construct a timer progress indicator and implement any timer-related functions through the `TimeClock` class. Employ `timerMaximumProvider` to present the maximum value of the timer, ensuring that the timer pauses when it reaches this maximum value. In `SettingsPage`, incorporate a `TextField` to allow users to define the maximum value of the timer and pass the input to `timerMaximumProvider`.

Question Content

To construct a widget that displays question content and options, use a `FutureBuilder`. Leverage `future: ref.watch(dataBaseProvider).getQuestionById(questionId),` to retrieve questions based on their corresponding `questionId`. Implement the questions card using `Card` and utilize `TexText` from the third-party package `tex_text` to render the question content in LaTeX.
After every query, if a question has already been answered, the state of `ref.read(selectedIndexProvider.notifier).state` is set to modify the style of the options.

Question Options

Display options using `ListTile` and query whether the question has already been completed. If completed, modify `selectedIndex` based on the completion status; options turn green when the selection is correct and red when incorrect.

With each response to a question, the database is updated in real-time using `ref.read(dataBaseProvider).updateQuestionInDatabase(newQuestion);`.

Next and Last Button

Utilize `TextButton` to implement the Last and Next buttons.
For the Last button, decrement the question index and query the previous question in `questionIdList`. Pass the `questionId` to retrieve the question. If there is no previous question, display an `AlertDialog()` to inform the user.

The implementation of the Next button is similar to the Last button, but it necessitates an assessment to determine whether all the questions have been completed, by comparing `questionIndex` with `amount`.

Information Button

Utilize `IconButton` to implement the information button. Each time this button is pressed, an explanation of the question will be displayed using a `bottomSheet`.

Statistics

Utilize `computeDailyAccuracy()`, `computeDailyCompletedQuestions()`, and `getTodayTotalTime()` from `DatabaseHelper` to calculate the corresponding data, and implement asynchronous operations using `FutureBuilder`.

In `WeeklyCompletedWidget`, employ the third-party package `fl_chart` to depict the daily completion quantities for the recent week.

In `HeatMapCalendarWidget`, leverage `flutter_heatmap_calendar` to represent the completion quantities for all dates for the user.

Dark mode

Utilize `darkModeProvider` to determine whether dark mode is currently adopted, and modify the overall app's light/dark appearance by altering `themeMode` in `ThemeData` within `MyApp()`.

User Guide

Use the third-party package `showcase` to sequentially highlight widgets, serving the purpose of guiding the user. To highlight certain widgets, it is necessary to wrap the corresponding widget with `showcase()` and set `GlobalKey()` to display different widgets in sequence.

Back end

`databaseHelper.go` encapsulates all functions related to database interactions, including initializing the database and performing CRUD operations—create, read, update, and delete.

`main.go` is responsible for handling data requests from the front end, using a JSON-based REST API structure to process and respond to HTTP requests. It serves as the central hub for routing and processing client-side interactions.

For example, when a request to `update_question` is sent from the front end, the questions are extracted from the database and converted into JSON format before being sent to the back end. The `updateQuestionHandler()` on the server side manages this request, decoding the JSON formatted data into the `Question` struct. Subsequently, it invokes the `updateQuestion()` function from `databaseHelper` to update the questions in the database. With this, the functionality to synchronize data from the front end to the back end is effectively realized.

Generator

1. **Base Case:** When the `depth` of recursion reaches `maxDepth`, the function returns either 'A' or 'B' randomly, serving as the base case to halt further recursion. (If you want more variables, could modified code to add more variables)
2. **Operator Selection:** At each level of recursion, one operator is randomly selected from the list `operators` containing 'AND', 'OR', and 'NOT'. The list is shuffled before selection to ensure randomness.
3. **Recursive Calls:**
    - If the selected operator is 'NOT', the function makes a single recursive call, incrementing the `depth` by 1, and wraps the result in 'NOT(...)'.
    - If the selected operator is 'AND' or 'OR', the function makes two recursive calls, incrementing the `depth` by 1 for each call, and combines the results with the selected operator between them.

Tests

Due to a lack of familiarity with the `flutter_test` package during the development phase, I did not perform automated testing on this project. However, I conducted thorough manual testing to ensure the stability and reliability of the application. I performed extensive boundary testing on various features and components of the app and continuously debugged throughout the development process to identify and resolve any potential issues and vulnerabilities. I plan to learn and explore how to utilize Flutter's testing packages more efficiently in future development to further enhance the quality and stability of my applications.

Packages

1. **settings_ui:**
   Settings_ui is leveraged to efficiently construct elegant and user-friendly settings screen UIs in Flutter applications.
2. **sqflite:**
   Sqflite serves as a SQLite plugin for Flutter, facilitating the efficient management of local, structured data within the application.
3. **path:**
   Path provides a comprehensive, cross-platform interface for managing file and directory paths, ensuring efficient path manipulation and retrieval.
4. **flutter_riverpod:**
   Flutter Riverpod is employed for state management, optimizing the application's state in a streamlined and scalable manner to ensure smooth user experience and performance.
5. **tex_text:**
   Tex_text is essential for rendering LaTeX in questions, ensuring the accurate and aesthetic representation of mathematical symbols and formulas.
6. **http:**
   The http package enables the execution of HTTP requests, allowing seamless communication between the app and web servers.
7. **fl_chart:**
   Fl_chart is utilized for delivering linear chart visualizations of data, representing user progress and statistics in an appealing and coherent manner.
8. **flutter_heatmap_calendar:**
   Flutter Heatmap Calendar offers heatmap calendar visualizations, presenting users' activity data intuitively over different time periods.
9. **go_router:**
   Go_router is incorporated for effective routing and navigation within the app, facilitating smooth transitions between different screens and views.
10. **showcaseview:**
    Showcaseview is a Flutter package allows you to Showcase/Highlight your widgets step by step. It's deployed to provide user guides, enhancing user understanding and interaction with the app's features.
11. **shared_preferences:**
    Shared Preferences is used for locally storing user preferences and app settings, ensuring a consistent and personalized user experience across sessions.

Sources

1. Parts of the code in class `HeatMapCalendar` are adapted from examples found in the `flutter_heatmap_calendar` package, available at links
2. Code used to construct option widget are adapted from ChatGPT.

```
ListTile(
    // Option Widget
    leading: Icon(
      selectedIndex == null
          ? Icons.circle
          : selectedIndex == i
              ? i == currentQuestion.correctIndex
                  ? Icons.check_circle
                  : Icons.cancel
              : i == currentQuestion.correctIndex
                  ? Icons.check_circle
                  : Icons.circle,
      color: selectedIndex == null
          ? Colors.grey
          : selectedIndex == i
              ? i == currentQuestion.correctIndex
                  ? Colors.green
                  : Colors.red
              : i == currentQuestion.correctIndex
                  ? Colors.green
                  : Colors.grey,
```

3. The structure of handler functions in the backend is inspired by ChatGPT.
4. The script that makes cells editable in html is inspired by ChatGPT.

Conclusions

Through this Fachpraktikum, I've gained a lot experience in application development. In parts of front-end development, I've learned how to build apps using the Flutter framework, mastering the use of various widgets, navigation between pages, and local data storage and transmission.

For the backend, I've learned how to use Go to transmit data with the front-end based on REST API and how to represent database information in HTML format. What

impressed me the most were the state management in the front-end using Riverpod and various asynchronous operations, which often led to unexpected bugs.