Fachpraktikum / Practical Course

Logic App Development: Coursework

PD Dr. Manfred Kufleitner

This coursework represents 100% of the total assessment of the module. You will build your own quiz app for formal logic using Flutter/Dart and Go (aka Golang). This includes the specification, implementation and documentation of your app. The app consists of three parts: the front-end (written in Flutter/Dart), the server-side (written in Go), and the automatic generation of tasks for the user (written in either Go or Dart).

Propose and Specify an App [mandatory]

Draft a specification (no more than two pages) of what type of formal logic your app is about (either propositional or first-order logic, or both) and what kind of quiz you create. Give short descriptions of the features of your app and your server-side. Explain how you are planning to automatically generate the tasks for the user. Create mock screens for all user-facing screens (front-end and admin screen); these screens do not count towards the page limit. Hand-drawn sketches in a digital format are OK.

Front-end [mandatory, 50%]

Develop the front-end in Flutter/Dart and make sure it implements the functionalities that you have specified. If the app uses a login mechanism (which is not mandatory), then there needs to exist a test user with username "`asdfgh`" and password "`asdfgh`". Your app needs to have the following mandatory screens:

- A screen for the quiz. After the user completed a task, they must be given some information about the correct solution (e.g., which answers where correct and which were wrong). This information could be provided on the same screen (either immediately after completing a task or after a round which could consist of multiple tasks).
- A screen for statistical information on the success of the user in completing the tasks. This information needs to be persistent.
- A screen for a user guide which explains the features of the front-end, containing images and texts as necessary.

Further screens are possible. Your front-end also needs to work off-line in which case there should be at least 10 different tasks for the user. The mandatory requirements can give you at most 40% of the marks. In addition to the mandatory core functionality (quiz, statistical information, user guide), you can choose to implement some of the following optional requirements:

- Implement some learning technique (e.g., spaced repetition).
- Use a timer either as a stopwatch or for a countdown and provide corresponding statistical information.
- Implement both a light mode and a dark mode (persistent).
- Use different layouts for landscape and portrait on all main screens.
- Introduce various difficulty levels as well as awards for the user (persistent).
- Allow different types of tasks (e.g., with and without a countdown, or questions about both propositional and first-order logic).

Each optional requirement gives up to 5% of the marks. It is possible to compensate for missing marks at the mandatory requirements with the implementation of more optional requirements. The maximal marks for the front-end is 50%.

Server-side [optional, 30%]

The main purpose for the server-side is to provide tasks for the front-end. The server-side needs to have at least 10 additional tasks (different from those built-in into the front-end). For the communication between the front-end and the server-side, you can either use a JSON-based REST API or GraphQL. The front-end should persist tasks downloaded from the server-side. The server-side also needs to collect anonymous statistical information from the front-end and provide this information via an HTML interface to an admin (e.g., show the easiest and most difficult tasks, etc.). And as a third requirement, it should be possible for an admin to add and remove tasks (without having to stop and restart the server). These three properties (API, admin interface, add and remove tasks) account for 25% of the marks. If you use a local database (the default choices would be PostgreSQL, MySQL or MongoDB) for persisting all data (tasks, statistical information, and possibly user data), you can achieve additional 5% of the marks.

User Management [bonus, 20%]

As a bonus, you can implement some user management (users can register and sign in) to synchronize all front-end data with the server-side such that users can login from different devices and have the same state of the front-end, and provide some high-score tables of the various users to the front-end. The admin should be able to delete users and reset passwords, but the passwords should not be stored as plain-text. You can achieve additional 20% of the marks which can be used to compensate for missing marks in both the front-end and the server-side. Note that this opportunity for bonus marks relies on implementing these features on the front-end as well as the server-side. You are only eligible for the bonus marks, if you implemented all core functionality of the front-end and the server-side. If you use some third-party libraries (i.e., third-party source code, not services like firebase) for implementing the user management (both on the front-end and the server-side), you are still eligible for the full 20% of the marks. The **maximal marks** for the front-end plus the server-side is **80%.**

Task Generation [optional, 20%]

Automatically create tasks for the user. You need to create at least 200 tasks; this can include the 10 built-in tasks of the front-end. You need to ensure that all your tasks are different and correct (i.e., that if answers are assumed to be true, then they are indeed true; and if they are assumed to be false, they are indeed false). You need to submit the program as well as the generated tasks themselves.

Report [mandatory]

Your report (no more than 12 pages) should have at least the following components:

- Your name and student id number.
- Introduction: What does your app do? Which licenses apply to the components?
- Overview: Which of the above features did you implement? Which additional features did you implement? Which parts are only partially functional?
- Setting up the project: Give all necessary steps for compiling and running your project. It should be possible to run your front-end on an emulator (iOS or Android).
- Brief documentation at a programming level: how did you implement the features, which architectural decision did you make and why, etc. (no more than 2 pages).
- Tests: Which kind of automatic tests did you do? What is your test coverage? (Do not include a complete test protocol!)
- Packages: Which packages and libraries did you use and for which purpose?
- Sources: Give a complete list of external sources other than the original package and programming language documentations (example apps always need to be cited even if you used only some very specific parts thereof and even if the examples are part of the packages).
- Conclusion: Summary of your experience.

You can use bullet points and tables where appropriate.

Presentation [mandatory]

Presentations are approximately 15 minutes each and can be given in either English or German. The presentation should include a short introduction, an overview of the project's implementation, and a live demo. The live demo needs to rely on the submitted project (later changes to the source code for the live demo are not allowed). You can use your own computer for the presentation; if you need a computer for the presentation, please let me know a few days in advance.

Sources

Whenever you use code or parts of some code written by somebody else, you need to give the source as a comment in your program as well as a citation in your report. The only exceptions are the original documentations of Dart, Flutter, Go, and the packages you use; here, you can copy and modify short examples given on their websites (but you still have to name the sources of larger amounts of code, e.g., copied from an example app).

Marking

In addition to the correct functionality, marks depend on how well you implement your own specification, on code style, on the structure of the project, on the originality, on the readability, and on the quality and smoothness of the user interface. If you use third-party packages or third-party code for directly implementing some of the features, the marks only depend on your work (e.g. on the effort for using the package in your particular situation). You are not allowed to use the Dart/Flutter library `getx` (sometimes also known as `get`). Moreover, you are responsible for any problems of your app caused by third-party contributions. Plagiarism detection software can be used during the marking process.

While the specification, the report, and the presentation do not directly contribute to the marks, they are also part of the marking process for all your other components.

Dates and Deadlines

Specification

You need to submit your specification by **23rd June 2023** at **11am** as a single pdf file in a folder `coursework/specification` in your git repository. The folder can contain additional files which you used to generate the pdf, but only the pdf will be relevant for marking.

Program for your Components

You need to submit the code for all your components by **22nd September 2023** at **11am** in your git repository in the folders `coursework/frontend` for the front-end, `coursework/serverside` for the server-side, and `coursework/generator` for the task generation. The submission for the front-end should also contain an apk file (basically, a compiled version of your front-end for Android). Moreover, all intermediate steps during the development need to be committed and pushed to the repository (not just the final version).

Report

You need to submit your report as a single pdf file by **25th September 2023** at **11am** in a folder `coursework/report` in your git repository. The folder can contain additional files which you used to generate the pdf, but only the pdf will be relevant for marking.

Presentation

The presentation of your app will be scheduled between **26th** and **29th September 2023**.