Quizzy- App Report

Logic App Development Coursework

Name: ***Akshay Kumar Venkatesha Narla***
Matriculation Number: ***3571510***

Introduction: The developed application, known as "Quizzy", provides a quiz based on propositional logic using the Flutter/Dart framework. The objective of the app is to run as a standalone app on a mobile device with the quiz in offline (when server is not running) as well as when server is online mode. The quiz provided consists of first-order propositional logic in offline mode and logical equivalence logic questions when the server is running/online. Additionally, the server is also developed from scratch using golang and MySQL. The format of quiz in the app is as follows:

- A quiz session contains 10 tasks.
    - Offline/Guest mode: 10 propositional logic equivalence question
    - Server mode: 10 tasks fetched from backend --> Boolean logic equivalence questions
        - Here, all questions are assumed to be of 4 variables- P, Q, R and S
        - Truth table of 4 variables must be equivalent for the correct choice and the question
- After the session ends, user can choose to review the quiz.
    - User answer and the correct answer would be displayed along with the session score.

Although, the licenses are not mentioned in the software itself, MIT License applies to the entire developed product (i.e, frontend, serverside, generator, SQL Schema). Therefore, any person or entity can use this product without any limitations.

Overview: This section shall provide an overview of the app functionalities. The detailed implementation information for each component of the developed software is provided against the specification defined at the beginning of the application.

Main features of the app include:

- UI can operate in offline mode as well. No user data is stored or persisted if the user wants to login in offline mode-- Guest mode or if the user is already logged in and the server isn't running.
- When connected to server, complete features of app are available:
    - New quiz everytime.
    - 2 different modes - Timed and Normal.
    - Persisted stats.
    - Multiple sessions for a user possible (different session tokens - although not tested in 2 devices).
    - User persistence -> can close the app and come back to see all data persisted. But if the user closes the app in between a quiz session, then current session data is lost. In case the user is logged in and the server goes offline then, the data is persisted in app database and will be sent to server (as long as session is valid and the user completes a quiz session when server is online, else data lost).

- Session expiry verified while app testing.

Front-end: The main functionality of the UI is to display tasks, manage the quiz and show users various data. It is a rather simple and easy-to-use user interface. The goal was to provide maximal functionality with minimal logical complexity. A clean architecture with minimal providers is provided. A singleton instance concept is used for database and API to handle the app lifecycle coherently.

Mostly, the concepts used during weekly tasks like riverpod state management, routing management with go_router and sqflite database for persistence have been reused for implementing the UI features. The quiz state is managed by a State Notifier that holds all necessary states of variables needed for managing a quiz session and is provided to the all widgets via riverpod. Nested Navigation with Stateful Shell Routing helps in efficiently providing a navigation bar/rail in the UI.

Widget binding is used at the start of the app to provide the riverpod data at app start/restart. Quiz API is efficiently managed with HttpService class to avoid code repetition. Comments are also provided in code for better understanding. The frontend architecture is inspired from several apps as well as internet resources which are mentioned in the references section. The code from these sources are modified to match the specifications of the current app.

Specification	Implemented Features in Software
“Flutter framework shall be used to develop the user interface(UI).”	The frontend software has been implemented using Flutter/Dart framework. The UI has been developed with Flutter > 3.1.1.
“The UI shall contain a “login” screen which allows users to log onto the quizzy app home screen.”	User login is now possible through the developed `quiz_api.dart` or a default guest mode if server is offline.
“The home screen shall provide users with the option to start a new quiz session, look at the user statistics and app user guide.”	Implemented with Flutter as a simple column widget with buttons to navigate to different screens and as the sample images provided in the specification document.
“Upon “New quiz session” selection, a new screen with quiz shall be displayed. An individual session would provide 10 quiz questions/tasks”	Quiz session selection feature has been implemented with an added step now between mode selection and quiz screen. A "Tap to start quiz" button allows the app to fetch tasks from backend (in case server is connected) and effectively acts as a quiz prep screen for the app database.
“The app shall run on 2 modes: timed and non-timed quiz. The mode selection shall be possible after the new quiz session selection.”	2 modes provided with the selection screen after the "New Session" option.

“The quiz screen shall provide the questions in a multiple choice format and shall be automatically generated from the back-end.”	The quiz is indeed a multiple choice quiz with 10 question and 3 choices each. The user has to select the expression that produces the equivalent truth table of 4 variables to the provided question.
“A countdown timer shall start when a new question is displayed, if timed quiz mode is selected. If the question is not answered before timed out, a new question shall be generated”	The only new package used in the project, \`timer_count_down\` is used for countdown of the timer. This is an easy to use package allowing user actions at the end of the countdown and robust restart or pause options. As soon as the timer counts down, the quiz automatically moves to the next question or next screen.
“Once an answer is provided, the correct answer shall be highlighted in green while the incorrect options in red.”	This is implemented in a slightly different manner. Instead of displaying the correct answer after the answer is provided, the quiz review is available at the end of the quiz where a user can review their session.
“Upon “User statistics” selection, quiz statistics of the user from previous sessions shall be displayed. This data shall be persistent.”	Upon stats button press, stats of the current user will be displayed. This data is persistent and can be verified by restarting the app. Provided the session is valid, this persisted data can be sent to the server upon completion of a online quiz session. Else, data is lost if the session expires.
“The app user guide shall provide users instructions for app usage.”	Although the developed UI is intuitive and easy to understand, upon guide button press, instructions for app usage is pictorially provided via PageView builder rather than a new package. The user can swipe through the guide to understand app operation.
“The app shall provide a different layout for portrait and landscape mode.”	Different layouts for landscape and portrait operations has been implemented and tested successfully. Although layouts has been tested only on the emulator mentioned above, it should work fine on other devices as well.
“To run the quiz offline, a set of 10 defualt quiz questions shall be used and built-in to the UI.”	Provided 10 first order propositional logic offline tasks which shall be fetched when the server is offline. The only drawback is that they will be displayed in the same order and the tasks will be same every time.

“The developed app shall run both online as well as offline”	From the context of the developed app, online refers to when the server is running. Offline mode, also known as guest mode here, is provided but there are certain pre-conditions for running this. In offline/guest mode, the 10 fixed questions would be displayed always.
“The developed app shall be tested on android emulator”	The developed app functionalities tested on an Android Emulator (Pixel4a with API 34 (latest or atleast 2023 version)).
“The validity of the generated quiz shall be ensured in the server-side.”	Handled in backend, particularly in the generator code.
“The app shall depend on “flutter_riverpod” for state management and “go_router” for navigation between screens”	Implemented as planned. Please look into \`pubspec.yaml\` for more details.

What's not in the Frontend?

- Flexibility to change countdown timer in timer mode → possible only via code
- Setting difficulty level of the questions.
- Check user session validity at app restart. (This is done only with get or post requests from the app)
- Leaderboard or any other user's data.
- No data saved without registering. i.e, guest mode is predominantly for offline cases.
- In case, admin changes the user passcode, that is not reflected if that user is logged in i.e., doesn't logout automatically → Reflected only on next login.
- In case of session expired, no automatic logout → User has to signout and login again.

Back-end: The main functionality of the developed serverside is to provide tasks to the Quizzy Flutter/Dart frontend. Additionally, it also handles the user management (login, register, signout handling), admin functionalities (via a HTML page - reset user passcode, remove user add or remove tasks, anonymous data tracking) and data storage (in the form of MySQL database for user data and stats storage).

Main features of the server:

- JSON REST based API endpoints, mainly for providing tasks to frontend.
- Unique usernames only allowed (i.e, no duplicate usernames possible) --> also handled in server.
- User management included.
- Admin functionalities included --> can view top users, anonymous tracking of who is logged in at any given moment.
- MySQL used for data persistence and all necessary database handling included. (ofcourse, database has to be setup before this)

In the following sections, details about the implementation of the server features are provided. The backend is written in Golang with MySQL database for storage. Mostly, the concepts used during weekly tasks have been reused. New concepts include - session cookie management, database management and json handling. Only the new concpets have some new package requirements else, mostly golang base packages are used. Comments are also provided in code for better understanding. The server has been developed with golang 1.21.5 and tested on a Windows local machine.

Specification	Implemented Features in Software
“The server shall provide the quiz questions or tasks to the UI using a Golang API server.”	The golang server provides the generated questions from the generator program to the Quizzy UI using the defined API endpoints. The tasks from the JSON file is fed into the MySQL database and will be available in the \`tasks\` table.
“JSON-based REST API shall be used for communication between UI and the server.”	Server sends data by encoding the tasks fetched from the MySQL database in JSON format before sending it to the UI upon \`GET\` requests. The received \`POST\` requests are also in JSON format and allows for easy data structure handling. Predominantly, user login data, user stats and tasks are communicated between the UI and the server.
“The server shall also collect information from UI and provide it to the admin via a HTML interface.”	In the developed server, the admin HTML provides user session tracking data (with session token info and the expiry time) and top users for the admin. This data is collected anonymously in the backend and is not reflected in the UI.
“The server shall provide an interface to the admin to remove or add tasks/quiz questions without stopping the server.”	The HTML provides separate sections for the admin to add or remove tasks directly from the database without having to stop the server. This can be later verified by checking the MySQL database.
“A local database based on sqflite package (SQLite) shall be used for storing and persisting user data.”	For data storage, a MySQL database is used. Although SQLite database exists, this was written initially with Flutter in mind. Provided the resources available for MySQL, it was easier to use that as database. Hence, MySQL is used here for this specification.

User management:

Specification	Implemented Features in Software
“A new server backend based on REST API shall be implemented for login and registering users using	User management is implemented with the help of MySQL. Login is handled by comparing stored hashed passcodes for a given username and

Golang for synchronizing across devices."	access allowed. For synchronization across devices and to manage same data for multiple logins, a session token is generated at login for the user, which allows user to get stats and tasks from the server as long as the session is valid. Since username is the primary key here, all data remains consistent as username remains same even when logging in from multiple devices. Therefore, data can be fetched in a consistent manner.
"The admin shall be able to delete users and reset passwords."	Through the admin HTML that is accessible via localhost:8080/admin when the server is running, the admin can delete users or reset user passcodes directly.

What's not in the server?

- Automatic task generation is a seperate program and has to be run separately to obtain the JSON file. It has to be placed in the correct location with the same name `tasks_backend.json` for server to run without any user intervention.
- Creation of MySQL database is not handled in the server. It has to be done separately by the user.
- The time.Now() package has some issues as it is inconsistent in the time output provided. Sometimes it provides time with UTC 0000, sometimes with UTC+0100. Hence, a session time is 80 minutes provided here, so that user will have atleast 20 minutes session.

Database: The database is created with MySQL. The setting up of database was as per this installation guide (https://allthings.how/how-to-install-mysql-on-windows-11/). Following this, the database and tables were setup according to this reference (https://www.mysqltutorial.org/mysql-basics/mysql-commands/). Mostly, the default configurations are retained. A new user other than the root was setup using the MySQL Workbench configuration and used to connect the server to database. But the visualization can be done via the root user as well.

- Database name: quizzy_server
  - Tables created: with column details and PK is primary key.
    - user: username (PK), passcode -> in hashed form
    - user_stats: username (PK) - composite with user 'PK', mode(PK), score, sessions
    - tasks: task_id (PK), Question, Choice1, Choice2, Choice3, AnswerIndex
    - sessions: username, token (PK), expiresAt --> unique token allows same users to login from multiple devices with different session tokens

The database schemas is also provided with the gitlab repository for reference.

Automatic Task/Quiz generation: The main functionality of the developed generator is to automatically generate tasks that will be used in the server to provide for the Quizzy frontend. The equivalent expression evaluation has to be always done as P, Q, R and S truth table and not just the values written in the question.

For example, if the expression is (P || S), then the equivalence is decided based on the P, Q, R, S or 4-variable truth table itself and not on just P-S truth table/2-variable truth table. Therefore, the quiz might be a bit time-consuming and not so straightforward.

The following section provides necessary information about the implementation of the automatic task generation for the quizzy app. The generator provides boolean equivalence tasks where the user has to select the equivalent choice for the generated logic's truth table with 4 variables – P, Q, R and S. The generator is written in golang 1.21.5 using basic logic and tested in a Windows local machine. Mostly, the concepts used during weekly tasks have been reused. Comments are also provided in code for better understanding.

Specification	Implemented Features in Software
"The propositional logic quiz questions shall be generated automatically using Golang."	A random variable/s selection is done and then they are combined with the && or || operators. 8 variables are defined with 2 operators which gives us 112 possible 2 variable combinations. Additionally, random negation of the complete expressions is also possible => 224 possible unique expressions now. The 224 unique expressions can be combined to generate more expressions. Then to generate propositions, the program combines the expressions generated based on a random number generated (1 or 2). Therefore, this program can generate a logic with 4 variables at max. This can be changed by increasing the proposition length.
"The multiple choices shall also be automatically generated along with the questions."	For generating the choices, first a truth table for the generated proposition is created. Since, only 4 variables(P, Q, R and S and its negation make up 4 more) are used, a truth table for 4 variable(i.e, TTTT, TTTF, TTFT, TTFF ....) is created. The generated expression would then be evaluated with this truth table. i.e. in case (P || Q) is to be evaluated, it will be given the truth table for evaluation and then the corresponding truth table for the expression is obtained. After this another proposition is to be picked which produces the same truth table => logically equivalent, which will be the correct choice. Programming is done to avoid same expressions being picked and in case the truth table is not equivalent, the program will keep generating new expression until an equivalent expression is found. Similar logic is applied for generating not equivalent choices as well. Shuffling of the choices while keeping track of answerIndex is done for randomness.

“The generated tasks shall be fetched from the server to the UI using JSON-based APIs.”	The generated tasks and choices are structured onto correct data structure and provided as a JSON file. A JSON file will be generated and is ready to be used with the server.

What's not in the generator?

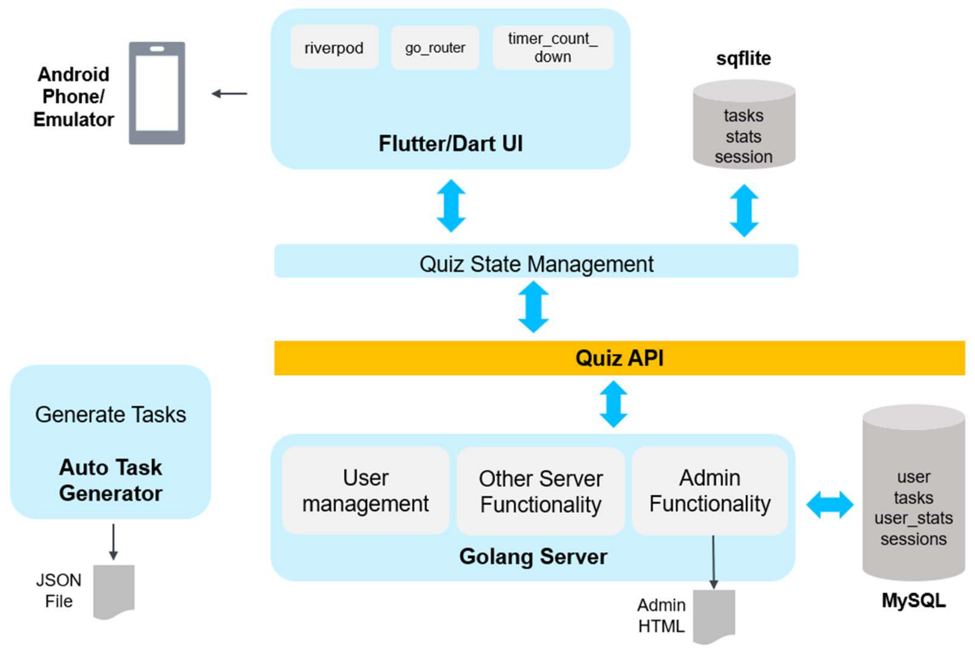
- An executable file. This can be generated by the user by running `go build`. This is not provided since cross platform support needs changing environment setup (i.e, exe file created in Windows might not be suitable for Linux environment). Therefore, in case the user wants to run the program, he/she can run it by `go run main.go`.

Project Setup: The project was developed in a VS Code environment with installations of Flutter, Dart and Golang in a Windows machine. The app frontend can be run by running ‘flutter run’ command from the command line for the unmodified code available in the git repository. Additionally, the debug apk or compiled version of the frontend files should be available in the uploaded git repository which can be directly installed onto an emulator.

Provided the integrity of the serverside files are retained and the database is setup correctly, the server can be run from the cmd folder by running `go run main.go`. Comments are retained on the golang program since it will be easy to debug in case of any errors in the backend. Running the generator is fairly straight forward. Just navigate to the cmd folder and run `go run main.go` from the terminal (provided golang is correctly configured).

For a successful running of the app with all its features, the frontend apk should be installed correctly onto the device/emulator and the serverside program should be running. Generator is to be executed seperately to obtain the tasks in json file, which can then be used with serverside program.

Notes on System Architecture:



Focus of the overall software development has been towards simplicity, while satisfying the specifications. The above figure represents the overall system architecture of the developed software. Navigation within the app is provided via go_router using a nested navigation with statefull shell branching. A clear separation between the frontend component and Quiz API is provided via the quiz status state notifier and other providers using riverpod. The countdown timer was initially tried as a state notifier provider. But this architecture functioned incorrectly and never counted down to 0. Additionally, the need of a different logic to automatically move to next question was complicated initially. Therefore, a new easy-to-use countdown timer package was used which functioned efficiently and provided the flexibility to easily configure next action upon counting down to 0.

The session data and other data collected by the providers is stored into the local sqflite database via CRUD operations. The local sqflite database stores the user session data, tasks for a session and the stats of the logged in user for persistence during app restarts. The quiz API provided is robust and avoids any redundant code and provides a JSON REST based communication end point to the serverside.

The serverside code written in golang is also structured in an easy-to-understand manner and with focus on simplicity. No complex logic is used here as well and mostly basic golang functionalities is used with CRUD operations for database operations. The generator logic for generating tasks automatically is naïve and fairly straightforward as explained in the previous sections.

Tests conducted: To test the overall software, mostly manual basic functional test has been conducted and verified. Initially, individual components are tested to verify their functionalities.

Frontend:

- Firstly, individual screens are tested for their functionality. Routing to correct screens upon button press and any user input is verified.
- Use of 'debugPrint' helped in identifying major bugs and fixing it.
- Quiz screen is tested by verifying if the choices are correctly selected upon user click.
- At the end of the quiz, the session score calculation using riverpod state notifier is verified along with the quiz review screen. Correct answers and the user selection of the answer displayed is cross-verified manually.
- User stats update is also verified manually by verifying if the accuracy is updated after each session.
- Automatic moving to next question is also verified by manually running down the countdown timer.

Serverside:

- Serverside responses are initially verified with curl and command line, followed by responses for requests directly in the browser.
- Additionally, use of print statements verified the functionality of the individual functions of the server.
- HTML page functionality is also verified manually by doing the corresponding operations in the page and cross-verifying it in the server database.
- Database operations are verified visually using MySQL Workbench.

Generator:

- Generated questions and the truth table equivalence of the generated choices are verified manually.

Overall, system testing is also done manually by verifying if the tasks, user stats and login details are correctly fetched between the frontend and the serverside. API tests is manually done by checking the logout, login, register, reset passcode for the user from the frontend and verifying the change in the MySQL server database. Session token expiration testing is done manually by ensuring that the app is running for 80 mins, followed by verifying the session expiry notice while trying to run a quiz session.

Packages used: This section provides a small overview of the packages used in the software other than the base packages that comes with the programming language.

- *riverpod*: Flutter package for reactive caching using declarative programming. This package provides efficient, easy state and data management within the app, allowing for robust features such as quiz session data, user stats and countdown controller to be accessed anywhere within the app.
- *go_router*: Flutter package for declarative routing that provides a convenient, url-based API for navigating between different screens. In the context of current software, this package is used for moving between the various screens of quizzy in a nested navigation manner.
- *timer_count_down*: Flutter package for simple timer countdown. This package is highly customizable with built-in modifiable timer and onFinished callback options. This is used for timer in timer mode of the quiz.
- *sqflite*: Flutter package for managing SQLite database for local app database. This package is used to create and manage database for storage and data persistence. It consists of all necessary helper functions for CRUD operations in the created app database.
- *http*: Flutter library for managing HTTP requests. This service is mainly used with the created API to post data into and get data from the server.
- *uuid*: Golang package for generating session tokens that is used to authenticate user communicating with the server.
- *goval*: Golang package for logic expression evaluation in golang. This is used for evaluating logical expessions to generate boolean logic and also the multiple choices with one correct/equivalent choice in generator software developed in golang.
- *mysql-driver*: Golang package for setting up the driver of the database/sql package in golang that allows to connect the created database in MySQL to the golang server program.

Conclusion: As described in the previous sections, a working quiz app, quizzy, is developed which provides propositional logic quiz tasks. The entire software architecture as well as the software for UI and server has been developed from scratch using Flutter/Dart framework and Golang. This full stack development project has provided a complete understanding of the functional aspects of a working app. The project provided an opportunity to learn concepts such as state management, navigation within the app, database management for an app, JSON REST based API endpoints, server handlers, admin functionalities, user management, MySQL database for a small project and clean architecture practices. The project would help in providing a base for further app development techniques.

Sources and References: References are also mentioned within the code itself. There might be some missing references in the following list that might have been mentioned in the code. Additionally, this list does not contain the official documentation of the flutter native packages and the coding language used itself.

- Weekly Flutter assignments
- Logic Quiz App reference: (https://github.com/zjhnb11/logic_quiz_App)
- Trivioso App: (https://github.com/zg0ul/Trivioso)
- The MIT License: (https://opensource.org/license/mit)
- Nested Navigation/Stateful Shell Navigation: (https://codewithandrea.com/articles/flutter-bottom-navigation-bar-nested-routes-gorouter/)
- Riverpod Documentation: (https://riverpod.dev/docs/introduction/why_riverpod)
- Riverpod: Code with Andrea: (https://codewithandrea.com/articles/flutter-state-management-riverpod/)
- Singleton: sqflite reference: (https://nyonggodwill11.medium.com/flutter-sqflite-323c035dcffe)
- HttpService as a Singleton class: (https://www.digitalocean.com/community/tutorials/flutter-flutter-http)
- timer_count_down package: (https://pub.dev/packages/timer_count_down)
- Golang official documentation: (https://pkg.go.dev/)
- Project layout of serverside inspired from article "Getting Started with Go Project": (https://medium.com/evendyne/getting-started-with-go-project-structure-ab8814ded9c3)
- Markdown documentation: (https://www.markdownguide.org/)
- Go and databases: (https://go.dev/doc/tutorial/database-access)
- MySQL installation: (https://allthings.how/how-to-install-mysql-on-windows-11/)
- MySQL Basics: (https://www.mysqltutorial.org/mysql-basics/mysql-commands/)
- MySQL with VSCode: (https://www.geeksforgeeks.org/how-to-connect-to-mysql-server-using-vs-code-and-fix-errors/)
- MySQL Composite Primary Key: (https://hevodata.com/learn/mysql-composite-primary-key/)
- Exception Handling in Golang: (https://www.honeybadger.io/blog/go-exception-handling/)
- Authenticating multiple logins from same user: (https://dev.to/theghostmac/understanding-and-building-authentication-sessions-in-golang-1c9k)
- Session Handling: Multiple login from same user: (https://www.sohamkamani.com/golang/session-cookie-authentication)
- Hashing of passwords: (https://gobyexample.com/sha256-hashes)
- Google UUID package: (https://pkg.go.dev/github.com/google/uuid)
- MySQL driver in golang: (https://github.com/go-sql-driver/mysql)
- GoVal package: (https://github.com/maja42/goval)