**Quiz Manager**

**Technical Specification**

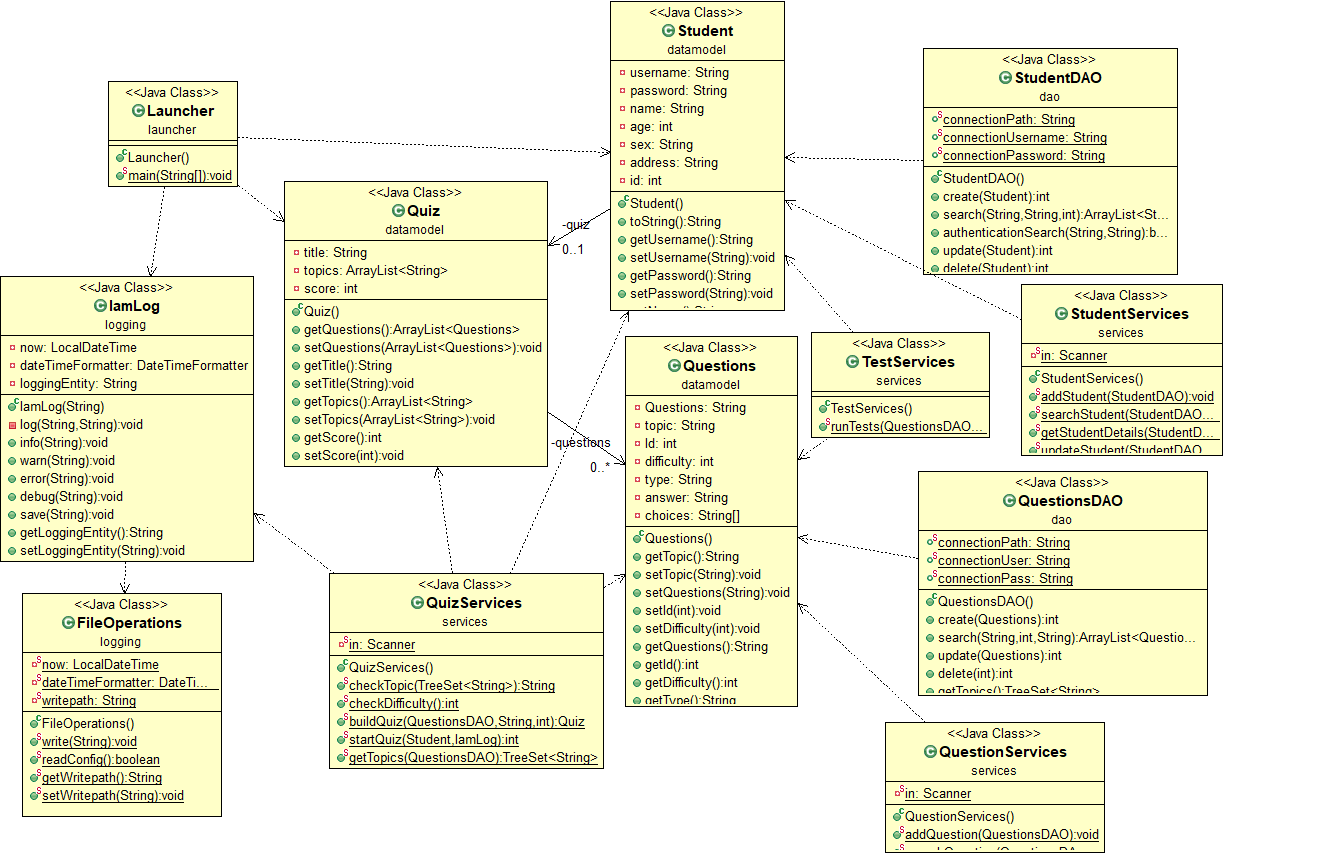
**Author**

**Subaandh Sambharathan VELLAKINAR KRISHNAKUMAR**

**Overview :**

The Quiz Manger application is developed using JAVA it runs both as a console-based application and on a GUI. The Database used is PostgreSQL. JavaFx is used for GUI. The users are the admin and the student the admin can perform operations like Create/Read/Update/Delete on all available tables in the database. The student can login to attend Quiz based on the topic and difficulty selected.

**Schema Diagram:**



**Design and API’s :**

The program runs both on console and GUI. Once the user runs the launcher all the initialization part is done. To add/search/update/delete values the DAO’s are used which acts as an abstraction layer between the user and the database. All Query’s are constructed and handled in this function. There are special function in the services which read and process the input data from the user and pass it on to the DAO function which perform CRUD operations.

**Launchers :** There are two launchers in this program one for the console based working (Launcher.java)and other launcher for the GUI(InitializeGUI.java)

**Quiz:** The quiz class holds all the values needed to generate a quiz it is stored inside student and the values are filled based on the input given by the students.

**Student:** The student class fetches the student details from the database once logged in. And the values are maintained till the quiz ends or when the type of user changes. To login the student’s username and password is also saved in the database.

**Questions:** The question class is filled based on the requirements it is maintained as a list inside the Quiz object. The question list in the quiz object is altered whenever a building a new quiz. There are different types of questions like open and MCQ.

**API’s:**

|  |  |
| --- | --- |
| **API** | **Description** |
| **AuthenticationServices: authenticate()** | This takes three arguments and it returns true or false. Where true means the authentication is successful and false when it fails. |
| **QuestionServices:**  **addQuestion()** | Acts as an intermediate layer between DAO and Launcher. It gets input from the user and processes the values before adding a question. |
| **QuestionServices:**  **deleteQuestion()** | Acts as an intermediate layer between DAO and Launcher. It displays the available question and once the ID is specified the value question corresponding to it is deleted. |
| **QuestionServices:**  **searchQuestion()** | There are multiple search parameters by which it can filter so the user’s preferred search value is read as input and the output is returned from the Database. |
| **QuestionServices:**  **updateQuestion()** | It displays the available question and once the ID is specified the values that the user prefers to change is read and its updated in the database. |
| **StudentServices:**  **addStudent()** | Acts as an intermediate layer between DAO and Launcher. It gets input from the user and processes the values before adding a new Student. |
| **StudentServices:**  **searchStudent()** | There are multiple search parameters by which it can filter so the user’s preferred search value is read as input and the output is returned from the Database. |
| **StudentServices:**  **updateStudent()** | The student details is shown and it is updated based on user provided input. |
| **StudentServices:**  **deleteStudent()** | The student details is shown and it is deleted based on user provided input. |
| **QuizServies:**  **buildQuiz()** | The build quiz is a internal API once the input from the student is given about the topic and difficulty. The Quiz object is created, and Questions are added to it. |
| **QuizServies:**  **startQuiz()** | After building the quiz is started the questions are displayed to the user and once the answer is entered it is verified and the score variable is appended and at the end the result is displayed. |
| **FileOperations:**  **readConfig():**  **write():** | The readConfig() is to read database configuration parameters from the config.properties file.  The write function is to write the console output of the quiz results to the user directory. |
| **QuestionsDAO:**  **create()**  **search()**  **update()**  **delete()** | To create a question in the database this is called from question services after the input is processed. This add a question in the databased with a unique id to it.  To search a question from the database the search query is built based on the processed data from the previous call. It returns value based on the search parameters.  To update the details in the databased based on ID from the input provided by the user.  To delete the details in the databased based on ID from the input provided by the user. |
| **StudentsDAO:**  **create()**  **search()**  **update()**  **delete()** | To create a student entry in the database this is called from student services after the input is processed. This will add a new student in the databased with a unique id to it.  To search a student from the database the search query is built based on the processed data from the previous call. It returns value based on the search parameters.  To update the details in the databased based on ID from the input provided by the user.  To delete the details in the databased based on ID from the input provided by the user. |
| **InitializeGUI:**  **start()**  **switchscenes()** | This launches a GUI interface powered by javafx.  The different pages/scenes which is built using javafx is controlled in this function. Once this function is called with the request to switch the page it changes here. |
| **LoginGUI:**  **login()** | This is to build a page which gets the login credentials of the student and validates whether it true or false by calling AuthenticateServices.authenticate() |
| **QuizGUI:**  **difficulty()**  **topics()**  **buildQuiz()**  **displayResult()** | This builds the page/scene to get the difficulty from the student.  This builds the page/scene so that the student can select from the available topics.  This generates the quiz page/scene based on the topic and difficulty selected by the user this uses the API’s in the JAVA program to fetch values from the database.  To build a page which display the final score to the user once finish test is clicked. |