Design Document for GeoQuiz

Group IP_106

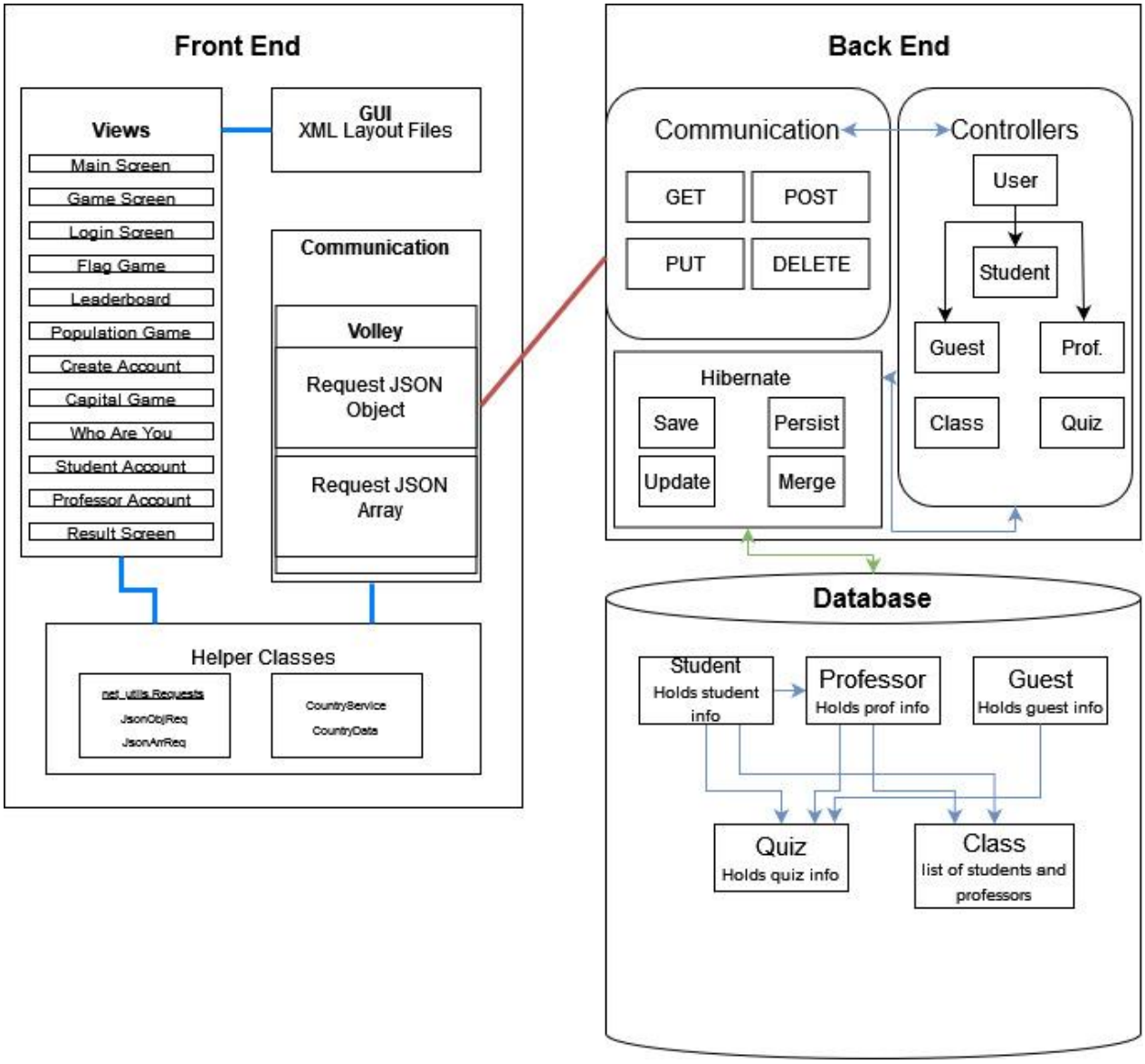
Member1 Name: Hankel Haldin Backend square of block diagram, SQL tables, Backend portion of pg 3

Wyatt Rayl: FrontEnd Block Diagram(Views & Helper Classes), FrontEnd of pg 3(Create Account, Account home)

Brayden Lamb: FrontEnd Block Diagram(Gui & Communication & Connector Lines), FrontEnd of pg 3(View Scores)

Member4 Name: % contribution

Block Diagram



Use this third page to describe complex parts of your design.

Frontend

-Create Account(Student,Professor)

Create account has the following text boxes:

- First & Last name for identification
- Email for Login
- Password
- Classcode

Once this information has been filled out it is sent to the backend to be made into a new account that begins storing the data of that student like quiz scores.

-Account Home(Student,Professor)

Account home differs depending on if you have a professor or student account

For Students:

- Button:SelectGame
- Button:ViewPastScores

For Professors:

- Button:AssignGame
- Button:ViewStudentScores

Depending on which type of account is being used the purpose of the account home will be different and allow them to access different functionalities.

-View Scores(Student,Professor)

- DropDownMenu: Scores over the past week, month, and year
  - TextTable: QuizName, Score, %Correct, Quiz takers name.

View Scores allows a user to see a history of what games they have played and how well they did in each game, like a play history.

-Games (Guest, Student)

A user can choose from the games below to play

- Capital Game
  - Multiple choice questions
- Flag Game
  - Multiple choice questions
- Continents Game
  - Multiple choice questions
- Coat of Arms Game
  - Multiple choice questions
- Higher or Lower Population Game
  - Higher or Lower buttons

The user will then be given a score that is stored in the Database

-Leaderboard (Guest, Student, Professor)

A user can go to the Leaderboard screen to see the top scores of each game

Backend

Communication:

GeoQuiz uses a RESTful API to communicate with the Android Studio frontend as well as an external API that hosts geography data. This means that under some circumstances, a user request in the frontend will generate API calls to the GeoQuiz backend, which in turn make calls to an external API. These RESTful operations are encoded by their URL and are represented by the following operations:

- GET: Retrieves information the GeoQuiz database stores on user accounts (i.e, students, professors, and guests) as well as JSON data from an external API.
- POST: This operation sends data collected from the UI to the backend to be processed by the server and ultimately persisted in the database.
- PUT: In some instances, existing data needs to be edited or updated. The PUT method allows the application to refer to an existing object in the GeoQuiz database and alter its value.

- DELETE: Not all information in the database needs to be persisted indefinitely. This method allows the application to remove irrelevant or stale data from the system. An example might be removing a student from a class when they are no longer enrolled.

Server:

Our backend uses Springboot with Apache TomCat to implement our server. The server consists of three parts: models, controllers, and repositories.

Models:

Our models are Java objects that represent real-world entities. GeoQuiz collects and organizes a student's quiz performance and parses them into the relevant classes. For ease of use, it also aggregates these simpler objects into larger collections like classes.

Controllers:

A controller handles the interaction between requests from the client and what data is stored in the database. In other words, these are the interfaces specified on the web server that define how the client interacts with deeper layers of the GeoQuiz application. Each controller defines the RESTful operations for a specific model.

Repositories:

Each model has its associated repository, which is a collection of items that use the same model. For ease of development, GeoQuiz uses Hibernate to map Plain Old Java Objects to SQL tables on the backend. This way, the development team is able to quickly achieve the desired design and behavior and iterate to better refined versions of the application.

Database schema

Note: SQL workbench had difficulty representing our professor class in the following screenshot. The class name that is difficult to read that forms a many-to-one relationship with the student table is the professor table.