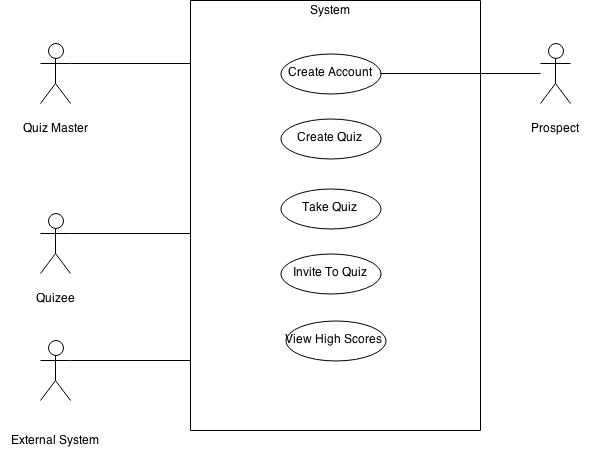
John Reagan

SE 554

Quiz Master

**Overview:**

A simple quiz administration application allowing registered users to create quizzes, take quizzes, view high scores and invite others to take a quiz.

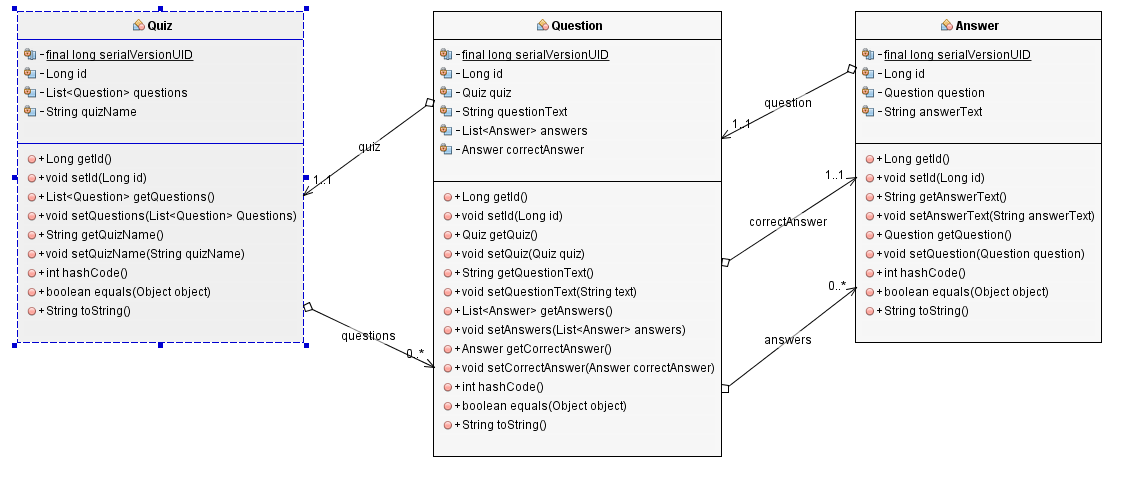


**Description of problem:**

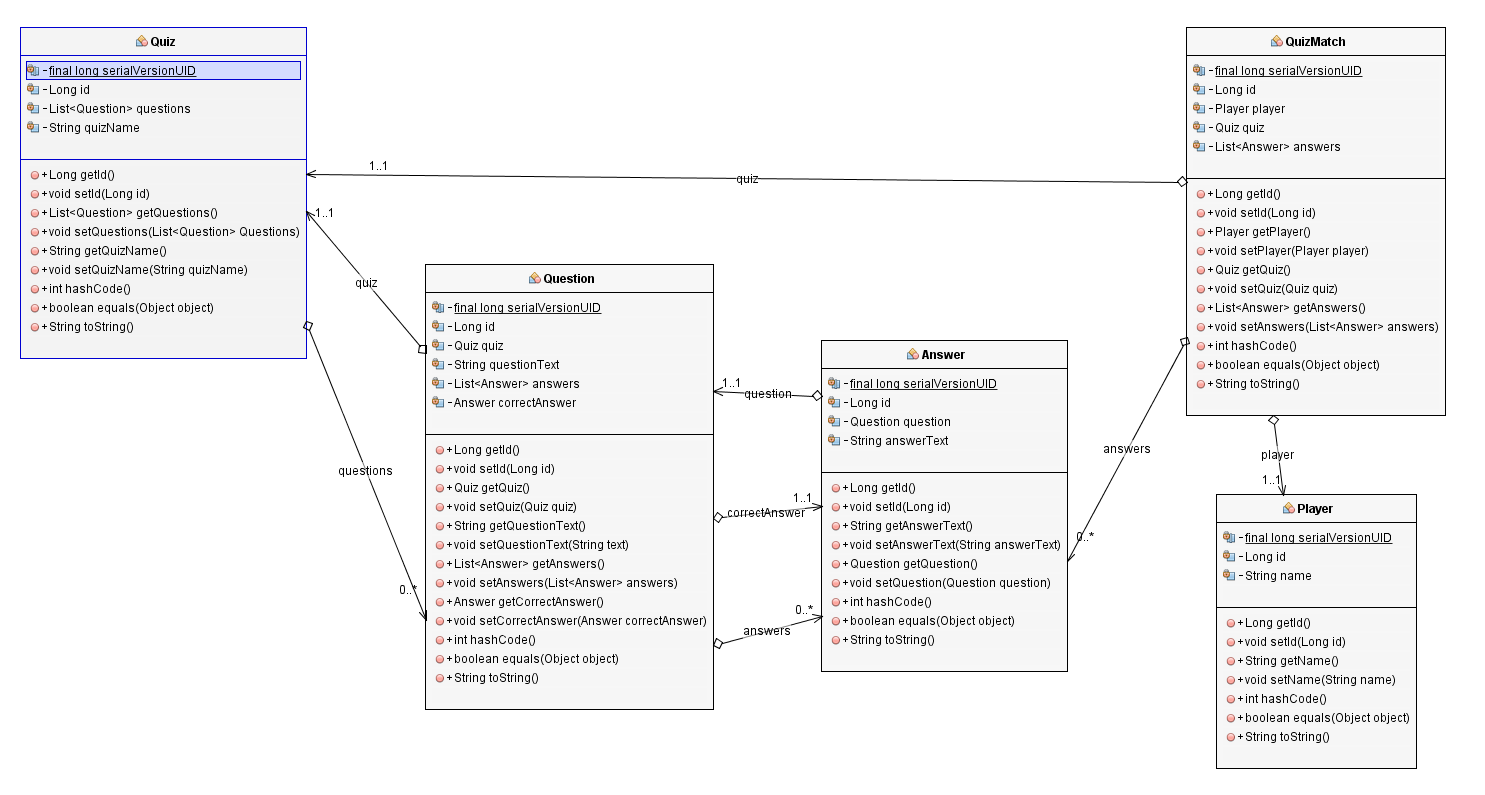
Basic quiz administration functionality for quiz masters, quizees, prospects and an external system. The application currently supports multiple choice quiz formats, but expanding the scope to include text, numerical and picture answers is planned.

**UML Diagrams**:

Quiz-Question-Answer Diagram:

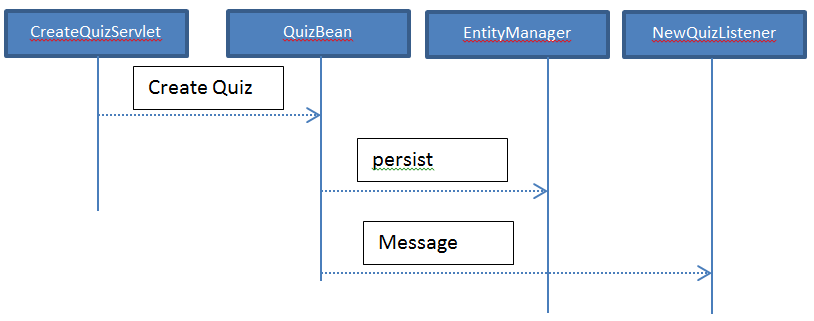


Complete Diagram showing relationships with player & quizmatch:

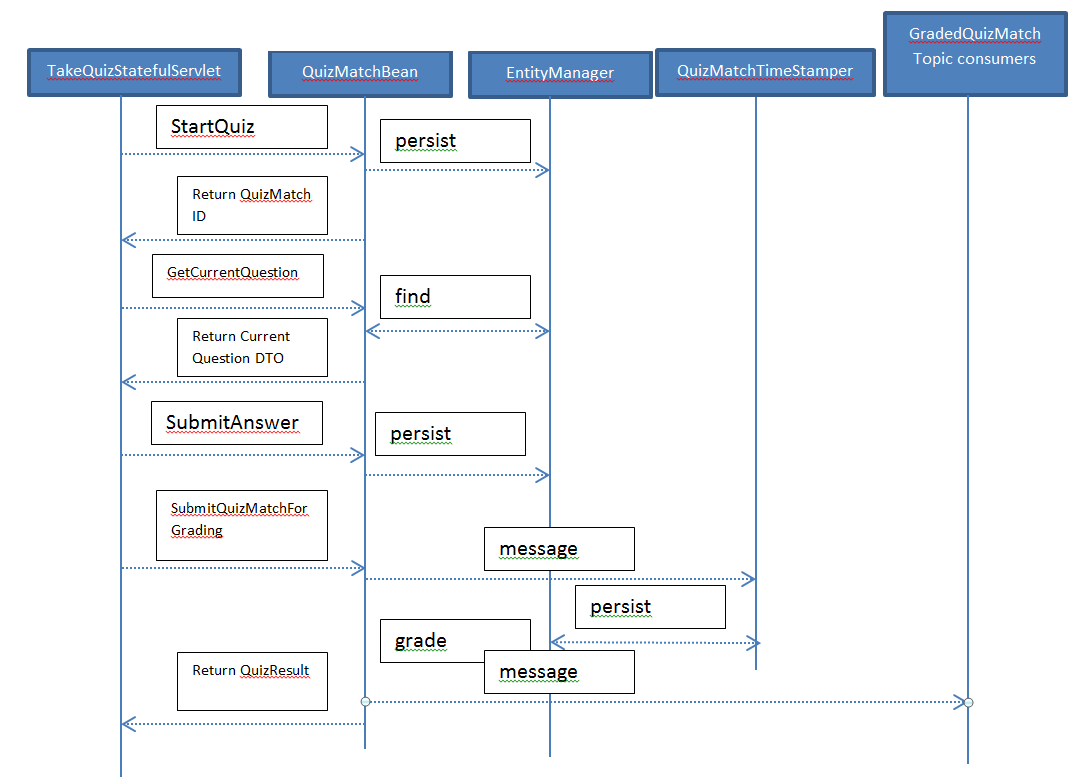


**Sequence Diagram:**

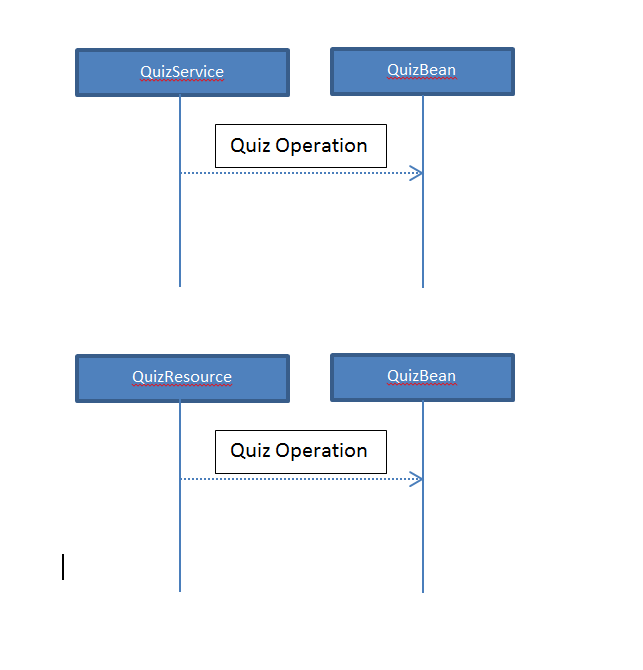
**Creating a new Quiz via web UI:**



**Taking a quiz via the web UI:**



**Interaction with quiz bean via SOAP or REST(proxy pattern):**



**Table Structure:**

|  |  |  |  |
| --- | --- | --- | --- |
| |  | | --- | | QUIZ | | ID  QUIZ\_TYPE  QUIZ\_NAME | | A quiz is created by a user and contains a list of questions which in turn hold answers.  There are two types of quizzes, scored quizzes which provide a score based on the number of correct answers and survey quizzes which analyze the score and provide a result. |
| |  | | --- | | SURVEYRESULTRANGE | | ID  QUIZ\_ID  CELING  FLOOR  MESSAGE | | SurveyQuizzes(or personality quizzes) use result ranges to analyze a quiz result. If the result is between the ceiling and floor provided, the quiz returns a message. |
| |  | | --- | | QUESTION | | ID  QUESTION\_TEXT  QUIZ\_ID  CORRECTANSWER\_ID | | A question is a question in a quiz. It holds a reference to the quiz which holds it as well as the correct answer for the question.  Ex: What color is the sky? |
| |  | | --- | | ANSWER | | ID  ANSWER\_TEXT  QUESTION\_ID | | Stores an answer associated with a quiz. May or may not be the correct answer for the question.  Ex:  Blue  Brown |
| |  | | --- | | QUIZMATCH | | ID  PLAYER\_ID  QUIZ\_ID | | A quizmatch represents an instance of a quiz being taken. Accordingly, it holds a reference to the quiz being taken as well as the player taking it. |
| |  | | --- | | PLAYER | | ID  PLAYER\_NAME | | A player, or quizee, is a user of the application who is taking a quiz. |
| |  | | --- | | QUIZMATCH\_ANSWER | | QUIZMATCH \_ID  ANSWER \_ID | | Table used to maintain the maintain the many to many relationship between the quizmatch and answer tables. Essentially, this table stores the supplied answers for a quiz by a player. |
| |  | | --- | | SEQUENCE | | SEQ\_NAME  SEQ\_COUNT | |  |

**Project Analysis:**

The quiz master project provided an interesting exercise and valuable experience in enterprise architecture. My employment experience is focused around desktop and stand-alone web applications and if nothing else, this project demystified many of the aspects around enterprise development and architecture.

One of the main strengths of the QuizMaster project is that it is architecturally sound. The web/UI and business layer are completely decoupled from one another via the QuizMaster-remote interface. In order to accomplish this without relying on a jar containing the entity classes, I chose to implement a sort of canonical model(since none was available for quiz administration) and defined simple descriptor objects that could be created and populated with data by the EJB module and consumed by the web/UI module. The downside of this decision is that there is additional work in implementing and testing these data transfer objects. As the saying goes, one degree of indirection gives one degree of flexibility. The other main strength of the project is its robustness, the application includes a full slate of services in both REST and SOAP formats, message transfer with JMS, declarative security and a working web interface.

The main weakness of the project emerged as a result of the integration testing in the embedded glassfish container. Despite providing the setup resources and using every bit of documentation from oracle that I could find, I found it very difficult to administer container resources on the embedded glassfish instance. Each time the application introduced a new resource that needed to be created in the container, a new set of issues were introduced. Each time a new problem emerged, I gritted(and sometimes worked around) the issue and continued using the embedded glassfish. By the time I was implementing container testing for message driven beans, I sincerely regretted by decision not to opt for a container testing framework such as Arquillian. The separate MDB module, the AdminTestRunner session are both examples of concessions I made in order to provide an adequate level of integration testing.

Ultimately, this was a great experience and provided valuable insights for me into the world of enterprise development. I regret spending so much time working around the embedded container, but only because it prevented me from expanding upon my project further.

**Feature Specific Notes:**

**Stateless Session Beans:**

QuizBean – used to create new quizzes.

QuizMatchBean – used to start and administer quiz matches.

**Stateful Session Beans:**

StatefulQuizBean – used to create a new QuizMatch instance and cycle through the questions.

**Message Driven Beans:**

QuizMatchTimeStamperBean – queue listener which provides a central resource for time stamping completed quizzes.

NewQuizMessageListener – Simple topic listener which receives messages regarding new quizzes being created.

**Singleton Beans:**

HighScoreSingletonBean -

**Remote Interface implementation:**

QuizMaster-remote: contains remote interfaces for beans. I also opted to include Data transfer objects due to difficulties of distributing and exposing quizzes without some sort of transfer object. DTOs seemed preferable to exposing entities with annotations and business logic to clients.

**Transactions:**

Both the QuizBean and QuizMatchBean require a new transaction to all of their methods. These beans do the bulk of the work of the operation and ensuring integrity at these critical points seemed necessary.

**Security:**

Included declarative security around quiz creation by restricting it to users belonging to the admin role. First, the CreateQuiz servlet has access restricted to it via web.xml configuration. The site uses basic authentication to verify if the user attempts to access the servlet. Second, QuizBean is annotated with an @AllowRoles and restricts access again to users in the admin role.

**SOAP:**

QuizService - A SOAP service utilizing reliable messaging

**REST:**

QuizResource – A REST service used to access public information in the QuizBean

**Github repository:**

<https://github.com/johnkreagan/QuizMaster/>

**Decision Log:**

|  |  |  |
| --- | --- | --- |
| **Date** | **Topic** | **Notes** |
| 4/14/2015 | IDE Environment | Chose Netbeans over eclipse for ease of use primarily. |
| 4/16/2015 | Persistence Provider | Chose EclipseLink as the JPA provider due to it being the default choice in the Netbeans IDE. Hibernate was also considered. |
| 4/16/2015 | Project Setup | Chose the maven project setup over a standard netbeans project. Future IDE flexibility and a personal desire to gain more experience with Maven were the driving factors. |
| 5/1/2015 | Presentation Layer | Chose to use standard JSP for the presentation layer over JSF. I found the facelets combination of controller and view awkward and opted for the more manual JSP setup. |
| 5/15/2015 | Testing | Chose to setup two sets of test, unit test for the primary entity objects and integration test for the EJB modules. I found container testing to be awkward and time consuming and felt that separate unit tests for entity POJOs would provide flexibility moving forward. |
| 5/16/2015 | Integration Test Container | Opted to use an embedded glassfish container for integration tests over an integration test framework such as Arquillian. A desire for simplicity and initial success with the embedded container drove this decision, though I would choose to use Arquillian if given the choice today. |
| 5/26/2015 | Project Structure | Chose a decoupled module approach leveraging a remote interface which is implemented by the EJB module and consumed by the UI(Web) module. |
| 5/27/2015 | Data Transfer Objects | Implementing the remote interface created a problem with the web/UI layer not having access to business objects. Rather than creating a JAR and making a dependency on the EJB module, I opted to create simple objects to be created by the EJB module and consumed by the web/UI layer. The downside to this decision was a duplication of entity objects. |
| 5/30/2015 | Web Services | Decided to implement both SOAP and REST services. Testing will be done with SOAPUI. Other testing is unnecessary because the services are essentially proxies to EJBs. |