



Object Oriented Analysis and Design
with Java
UE20CS352

QUIZZINATOR

NAME	SRN
ARYAN KAPOOR	PES2UG20CS069
AVIJIT PANDA	PES2UG20CS078
BHARATH D H	PES2UG20CS086
DOREDLA TIRUMALA RAVI TEJA	PES2UG20CS115

Abstract

We aim to build a quizzing platform for a wide assortment of quizzes which cater to the needs of both quizmaster and participant alike.

The project: "Quizzinator" is a collection of a number of different types of quizzes covering various topics. A user can access/play all of the quiz and can attempt any of the one.

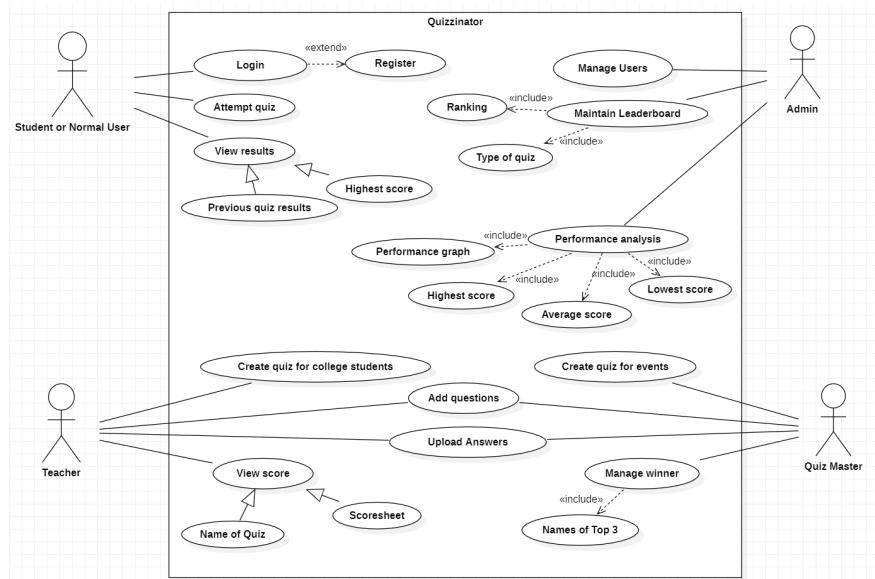
Quizzinator is a user-friendly quiz application which contains : Various quizzes , answers to every question, different formats of quizzes(Rapid fire, hints,Classic,Lifeline) Uploading of question and answer by admin/quizmaster, and to improve the knowledge level of users. Using this application the user and the quizmaster can analyze the scores as well as check the rankings and the highest,average scores.

QUIZZINATOR : A ONE STOP solution for Quizzing.

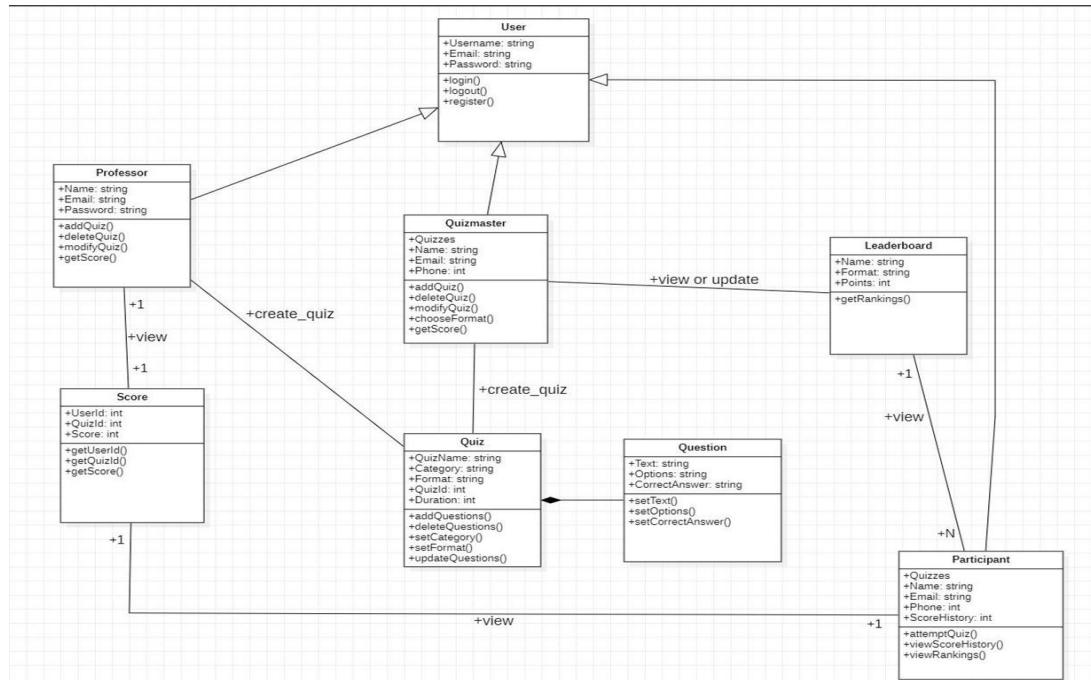
Tools Used:

1. **Spring Boot Web Application**
2. **MySQL Database**

USE CASE DIAGRAM:



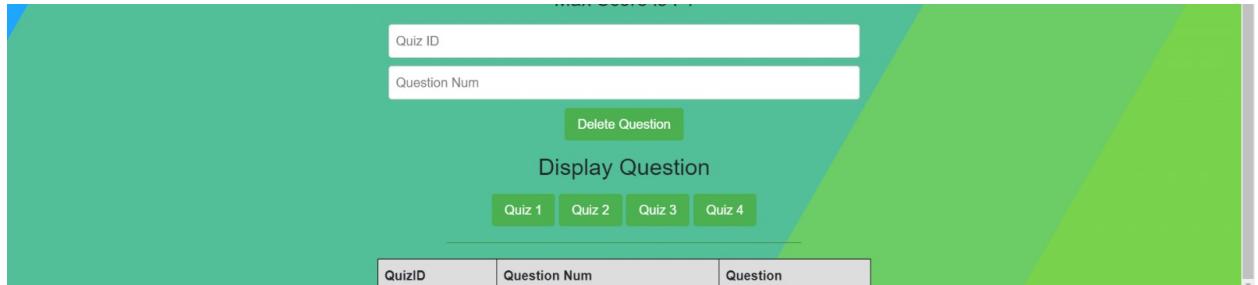
CLASS DIAGRAM:



Architecture:

Utilised GRASP Principles:

1. Information Expert: Assign responsibility to the object that has the most information required to fulfill it.



Controller: Assign responsibility to the object that manages and coordinates the flow of information and requests between objects.

QuizID	Username	Score
i2	Bharath	4
i2	Bharath	2

2. High Cohesion: Assign responsibility to the object that has the most related functions to perform.

The screenshot shows a web browser window with the URL `localhost:8080/profscore?profid=8&quizid=i2`. The page title is "DASHBOARD". At the top, there is a navigation bar with links for "Home", "Logout", "Contact", and "About". Below the navigation bar is a green button labeled "Add Question". A blue button labeled "Display Score" is also present. Below these buttons, there are four small green buttons labeled "Quiz 1", "Quiz 2", "Quiz 3", and "Quiz 4". A table below the buttons displays student scores:

QuizID	Username	Score
i2	Bharath	4
i2	Bharath	2
i2	Bharath	0
i2	Bharath	0
i2	Bharath	0

Below the table, two pieces of text are displayed: "Avg Score is : 1.5" and "Max Score is : 4".

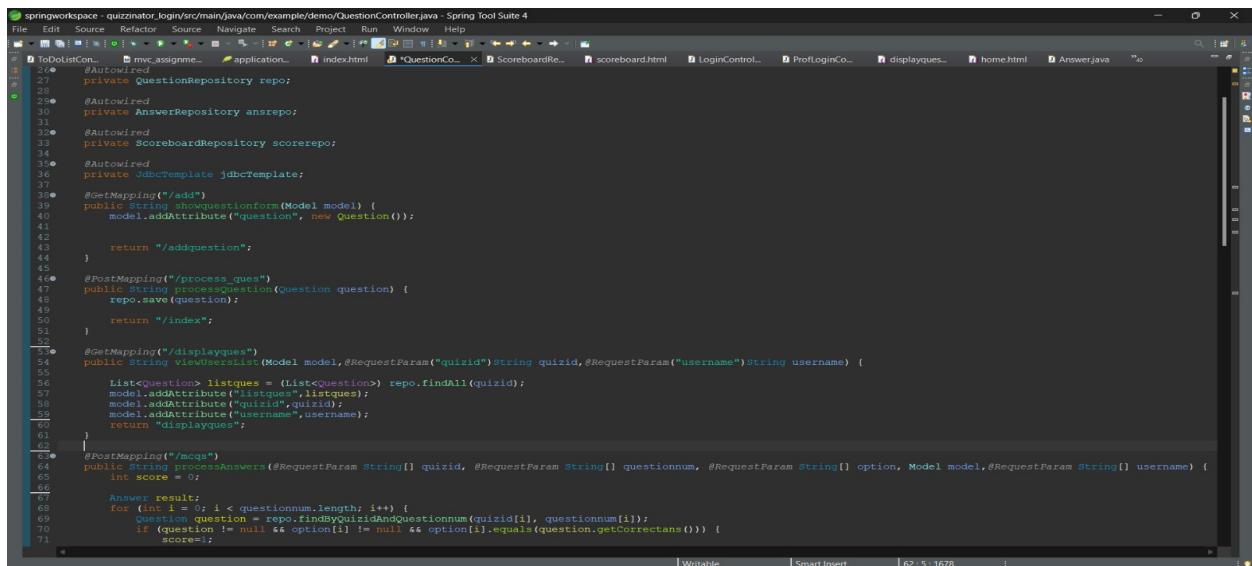
3. Low Coupling: Assign responsibility to the object that has the least amount of dependency on other objects.

The screenshot shows a web browser window with the title "Display Questions". On the left side, there is a sidebar with a purple header containing the text "Display Questions". Below the header, there are four green buttons labeled "Quiz 1", "Quiz 2", "Quiz 3", and "Quiz 4". To the right of the sidebar, there is a table displaying quiz details:

QuizID	Question Num	Question
i1	q1	er
i1	q2	ww

The MVC architecture can be implemented in the following way:

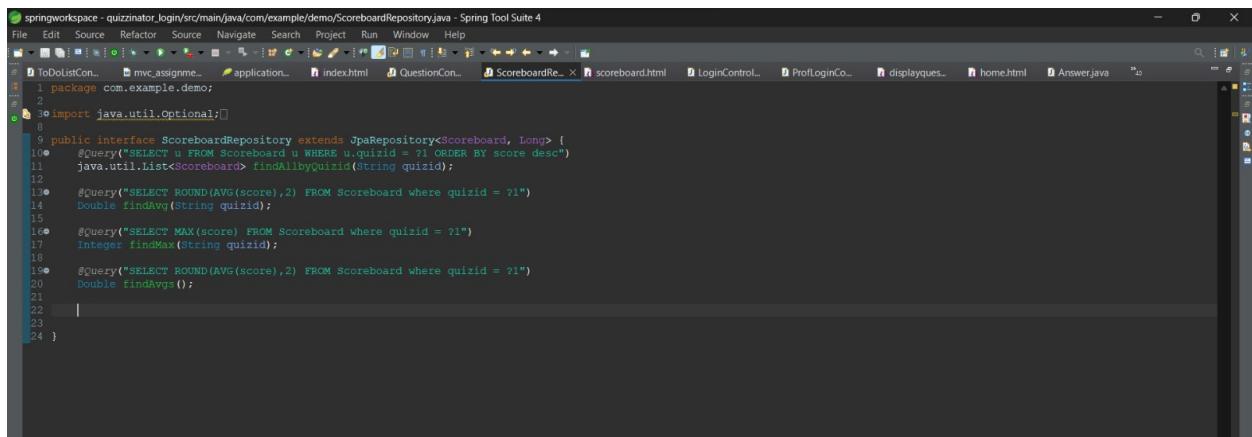
Model: The model represents the data and business logic of the quiz app. In this case, the model would include the quiz questions, answer options, correct answers, and other related data. The model would also contain the logic for generating random questions, checking answers, and calculating the score.



```

springworkspace - quizzinator_login/src/main/java/com/example/demo/QuestionController.java - Spring Tool Suite 4
File Edit Source Refactor Source Navigate Project Run Window Help
ToDoListCon... mvc_assignme... application... index.html QuestionCon... ScoreboardRe... scoreboard.html LoginControl... ProfLoginCo... displayques... home.html Answer.java ...
1 package com.example.demo;
2
3 import org.springframework.stereotype.Controller;
4 import org.springframework.ui.Model;
5 import org.springframework.web.bind.annotation.GetMapping;
6 import org.springframework.web.bind.annotation.PostMapping;
7 import org.springframework.web.bind.annotation.RequestParam;
8 import org.springframework.web.bind.annotation.ResponseBody;
9
10 import com.example.demo.model.Question;
11 import com.example.demo.model.Scoreboard;
12 import com.example.demo.repository.QuestionRepository;
13 import com.example.demo.repository.AnswerRepository;
14 import com.example.demo.repository.ScoreboardRepository;
15 import com.example.demo.util.JdbcTemplate;
16
17 @Controller
18 @RequestMapping("/addquestion")
19 public String showquestionform(Model model) {
20     model.addAttribute("question", new Question());
21
22     return "/addquestion";
23 }
24
25 @PostMapping("/process ques")
26 public String processQuestion(Question question) {
27     repo.save(question);
28
29     return "/index";
30 }
31
32 @GetMapping("/displayques")
33 public String viewdisplaylist(Model model,@RequestParam("quizid")String quizid,@RequestParam("username")String username) {
34
35     List<Question> listques = (List<Question>) repo.findAll(quizid);
36     model.addAttribute("listques",listques);
37     model.addAttribute("quizid",quizid);
38     model.addAttribute("username",username);
39
40     return "displayques";
41 }
42
43 @PostMapping("/mcqs")
44 public String processAnswers(@RequestParam String[] quizid, @RequestParam String[] questionnum, @RequestParam String[] option, Model model,@RequestParam String[] username) {
45
46     Answer result;
47     for (int i = 0; i < questionnum.length; i++) {
48         Question question = repo.findByQuizidAndQuestionnum(quizid[i], questionnum[i]);
49         if (question != null && option[i] != null && option[i].equals(question.getCorrectans())) {
50             score++;
51         }
52     }
53
54     model.addAttribute("score", score);
55
56     return "displayques";
57 }
58
59 }
60
61
62
63
64
65
66
67
68
69
70
71

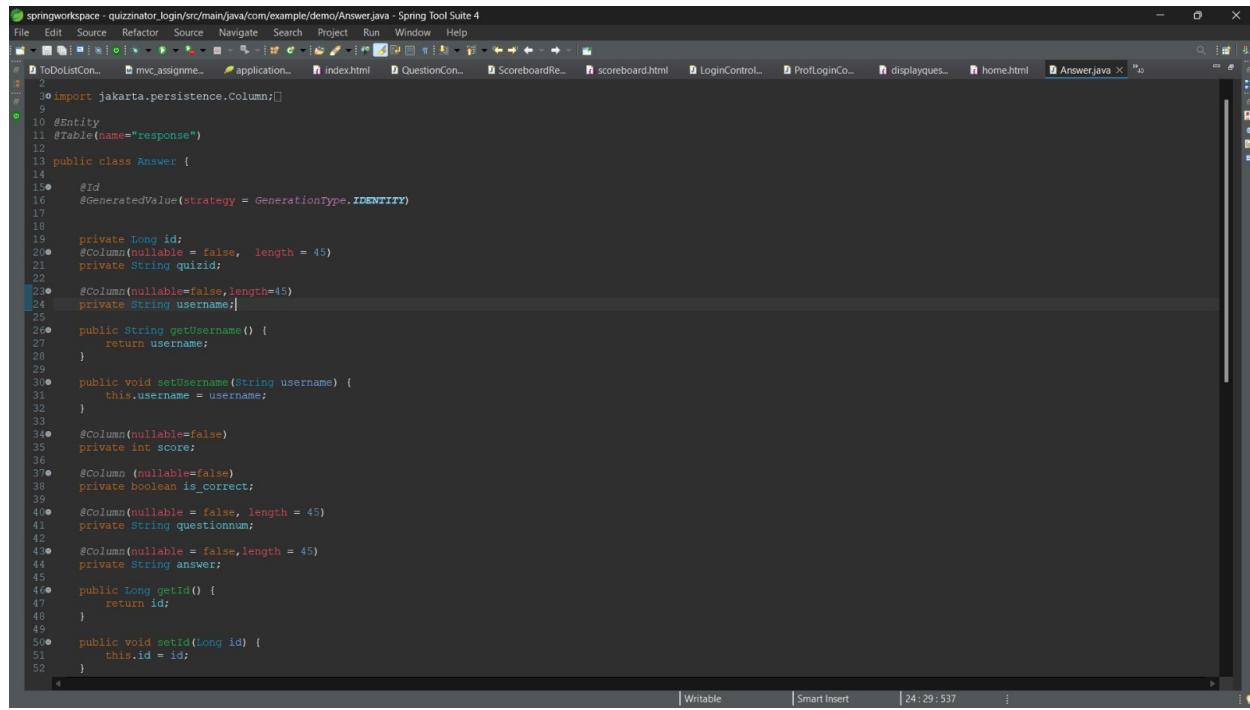
```



```

springworkspace - quizzinator_login/src/main/java/com/example/demo/ScoreboardRepository.java - Spring Tool Suite 4
File Edit Source Refactor Source Navigate Project Run Window Help
ToDoListCon... mvc_assignme... application... index.html QuestionCon... ScoreboardRe... scoreboard.html LoginControl... ProfLoginCo... displayques... home.html Answer.java ...
1 package com.example.demo;
2
3 import java.util.Optional;
4
5 public interface ScoreboardRepository extends JpaRepository<Scoreboard, Long> {
6
7     @Query("SELECT u FROM Scoreboard u WHERE u.quizid = ?1 ORDER BY score desc")
8     java.util.List<Scoreboard> findAllbyQuizid(String quizid);
9
10    @Query("SELECT ROUND(AVG(score),2) FROM Scoreboard where quizid = ?1")
11    Double findAvg(String quizid);
12
13    @Query("SELECT MAX(score) FROM Scoreboard where quizid = ?1")
14    Integer findMax(String quizid);
15
16    @Query("SELECT ROUND(AVG(score),2) FROM Scoreboard where quizid = ?1")
17    Double findAvgs();
18
19
20
21
22
23
24

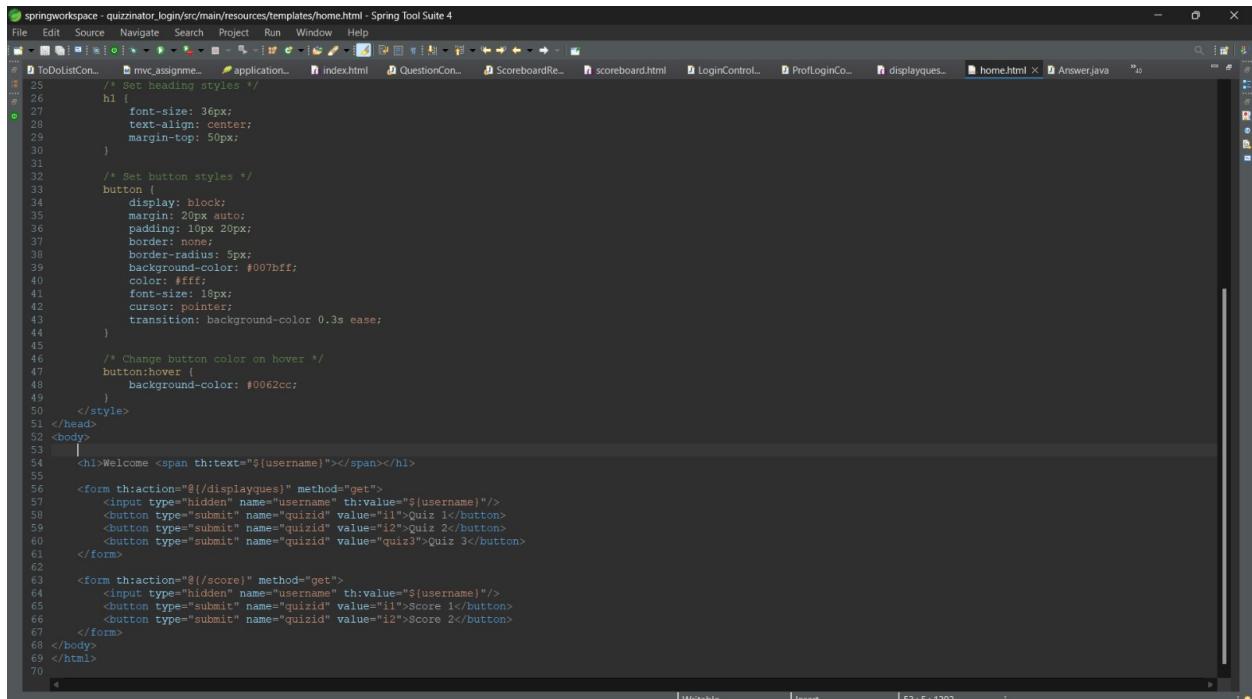
```



The screenshot shows the Spring Tool Suite 4 interface with the file 'Answer.java' open in the editor. The code is annotated with JPA annotations such as @Entity, @Table, @Id, @GeneratedValue, @Column, and @Transient. The code defines a class 'Answer' with fields for id, quizid, username, score, is_correct, questionnum, and answer, along with their respective getters and setters.

```
springworkspace - quizzinator_login/src/main/java/com/example/demo/Answer.java - Spring Tool Suite 4
File Edit Source Refactor Source Navigate Search Project Run Window Help
... ToDoListCon... mvc_assignme... application... index.html QuestionCon... ScoreboardRe... scoreboardhtml LoginControl... ProfLoginCo... displayques... home.html Answer.java X ...
1 import jakarta.persistence.Column;[]
2
3 import jakarta.persistence.Entity;
4 import jakarta.persistence.GeneratedValue(strategy = GenerationType.IDENTITY);
5
6 import jakarta.persistence.Id;
7 import jakarta.persistence.Table;
8
9 private Long id;
10 @Column(nullable = false, length = 45)
11 private String quizid;
12
13 @Column(nullable=false,length=15)
14 private String username;
15
16 public String getUsername() {
17     return username;
18 }
19
20 public void setUsername(String username) {
21     this.username = username;
22 }
23
24 @Column(nullable=false)
25 private int score;
26
27 @Column(nullable=false)
28 private boolean is_correct;
29
30 @Column(nullable = false, length = 45)
31 private String questionnum;
32
33 @Column(nullable = false, length = 45)
34 private String answer;
35
36 public Long getId() {
37     return id;
38 }
39
40 public void setId(Long id) {
41     this.id = id;
42 }
```

View: The view represents the user interface of the quiz app. In this case, the view would include the screens and UI elements used to display the quiz questions and answer options to the user. The view would also be responsible for handling user input, such as selecting an answer option and submitting the answer.



```

25     /* Set heading styles */
26     h1 {
27         font-size: 36px;
28         text-align: center;
29         margin-top: 50px;
30     }
31
32     /* Set button styles */
33     button {
34         display: block;
35         margin: 20px auto;
36         padding: 10px 20px;
37         border: none;
38         border-radius: 5px;
39         background-color: #007bff;
40         color: #fff;
41         font-size: 18px;
42         cursor: pointer;
43         transition: background-color 0.3s ease;
44     }
45
46     /* Change button color on hover */
47     button:hover {
48         background-color: #0062cc;
49     }
50 
```

</style>

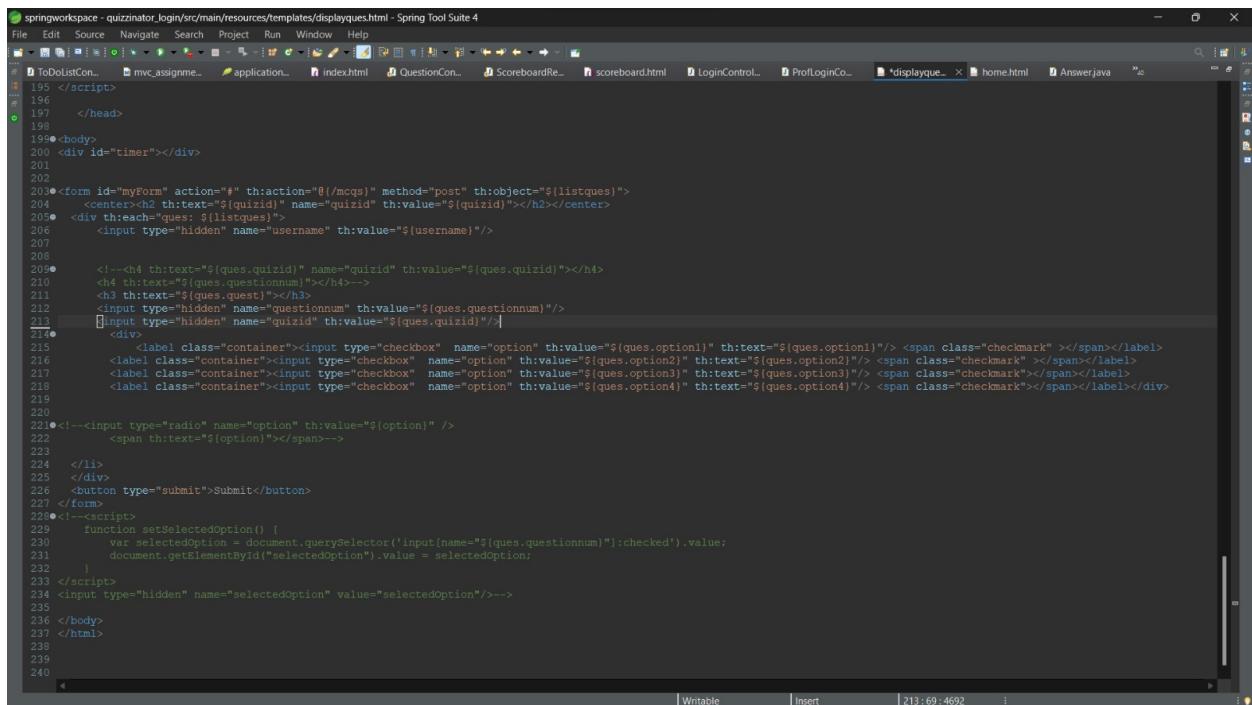
```

51 </head>
52 <body>
53     <h1>Welcome <span th:text="${username}"></span></h1>
54
55     <form th:action="@{/displayques}" method="get">
56         <input type="hidden" name="username" th:value="${username}" />
57         <button type="submit" name="quizid" value="1">Quiz 1</button>
58         <button type="submit" name="quizid" value="12">Quiz 2</button>
59         <button type="submit" name="quizid" value="quiz3">Quiz 3</button>
60     </form>
61
62     <form th:action="@{/score}" method="get">
63         <input type="hidden" name="username" th:value="${username}" />
64         <button type="submit" name="quizid" value="i1">Score 1</button>
65         <button type="submit" name="quizid" value="i2">Score 2</button>
66     </form>
67 
```

</body>

```

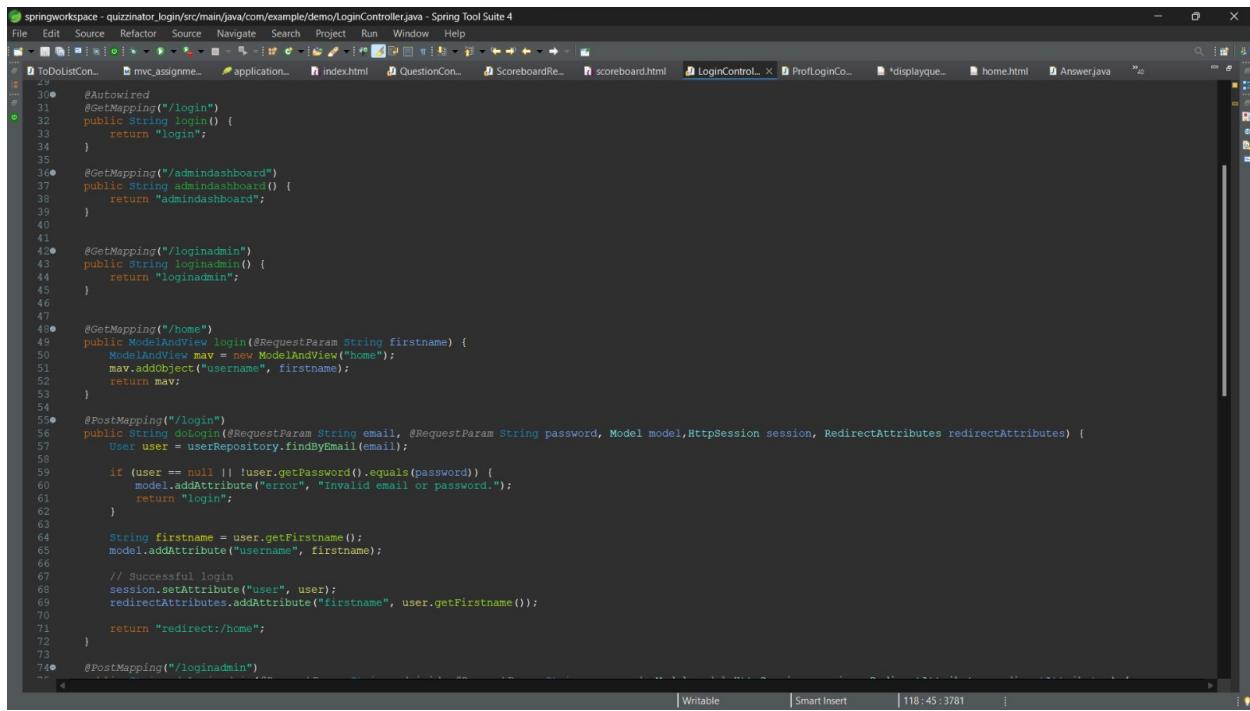
68 </html>
69
70 
```



```

185     </script>
186
187     </head>
188
189     <body>
190         <div id="timer"></div>
191
192
193     <form id="myForm" action="#" th:action="@{/mcqs}" method="post" th:object="${listques}">
194         <center><h2>${ques.quizid} name="quizid" th:value="${ques.quizid}"></h2></center>
195         <div th:each="ques: ${listques}">
196             <input type="hidden" name="username" th:value="${username}" />
197
198             <!--<h4>${ques.quizid}</h4> -->
199             <h4 th:text="${ques.questionnum}"></h4> -->
200             <h3 th:text="${ques.quest}"></h3>
201             <input type="hidden" name="questionnum" th:value="${ques.questionnum}" />
202             <input type="hidden" name="quizid" th:value="${ques.quizid}" />
203
204             <div>
205                 <label class="container"><input type="checkbox" name="option" th:value="${ques.option1}" th:text="${ques.option1}"> <span class="checkmark" ></span></label>
206                 <label class="container"><input type="checkbox" name="option" th:value="${ques.option2}" th:text="${ques.option2}"> <span class="checkmark" ></span></label>
207                 <label class="container"><input type="checkbox" name="option" th:value="${ques.option3}" th:text="${ques.option3}"> <span class="checkmark" ></span></label>
208                 <label class="container"><input type="checkbox" name="option" th:value="${ques.option4}" th:text="${ques.option4}"> <span class="checkmark" ></span></label>
209             </div>
210
211             <!--<input type="radio" name="option" th:value="${option}" />
212             <span th:text="${option}"></span>-->
213
214         </div>
215         <button type="submit">Submit</button>
216     </form>
217
218     <!--<script>
219         function setSelectedOption() {
220             var selectedOption = document.querySelector('input[name="${ques.questionnum}"]:checked').value;
221             document.getElementById("selectedOption").value = selectedOption;
222         }
223     </script>
224     <input type="hidden" name="selectedOption" value="selectedOption"/>-->
225
226     </body>
227     </html>
228
229
230
231
232
233
234
235
236
237
238
239
240 
```

Controller: The controller acts as an intermediary between the model and view, handling user input and updating the model and view as needed. In this case, the controller would receive user input from the view, validate the input, and update the model accordingly. It would also update the view with the current question and answer options, as well as the user's score.

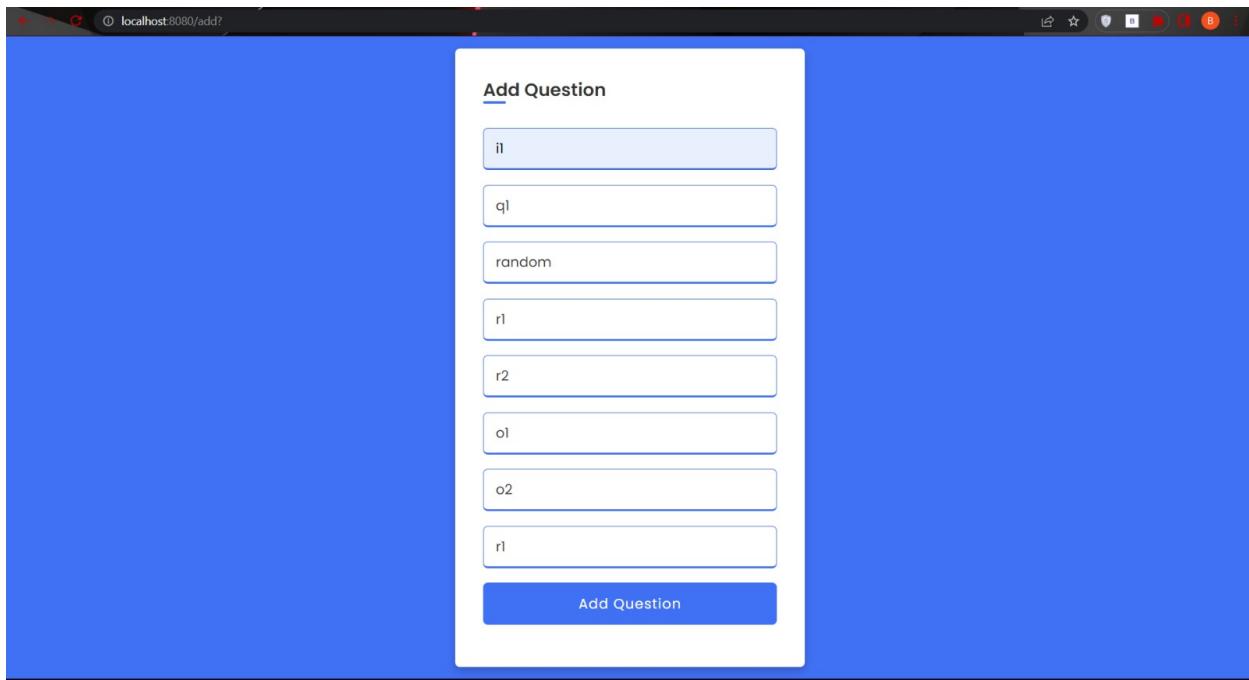


The screenshot shows the Spring Tool Suite 4 interface with the file 'LoginController.java' open. The code implements a Spring MVC controller with various methods for handling login requests. The code includes annotations like @Controller, @RequestMapping, and @ModelAttribute, along with logic for validating user inputs and setting session attributes.

```

springworkspace - quizzinator_login/src/main/java/com/example/demo/LoginController.java - Spring Tool Suite 4
File Edit Source Refactor Source Navigate Search Project Run Window Help
... ToDoListCon... mvc_assignment_ application_ index.html QuestionCon... ScoreboardRe... scoreboard.html LoginControl... ProfLoginCo... *displayque... home.html Answerjava ...
29
30● @Autowired
31 @GetMapping("/login")
32 public String login() {
33     return "login";
34 }
35
36● @GetMapping("/admindashboard")
37 public String admindashboard() {
38     return "admindashboard";
39 }
40
41
42● @GetMapping("/loginadmin")
43 public String loginadmin() {
44     return "loginadmin";
45 }
46
47
48● @GetMapping("/home")
49 public ModelAndView login(@RequestParam String firstname) {
50     ModelAndView mav = new ModelAndView("home");
51     mav.addObject("username", firstname);
52     return mav;
53 }
54
55● @PostMapping("/login")
56 public String doLogin(@RequestParam String email, @RequestParam String password, Model model, HttpSession session, RedirectAttributes redirectAttributes) {
57     User user = userRepository.findByEmail(email);
58
59     if (user == null || !user.getPassword().equals(password)) {
60         model.addAttribute("error", "Invalid email or password.");
61         return "login";
62     }
63
64     String firstname = user.getFirstname();
65     model.addAttribute("username", firstname);
66
67     // Successful login
68     session.setAttribute("user", user);
69     redirectAttributes.addAttribute("firstname", user.getFirstname());
70
71     return "redirect:/home";
72 }
73
74● @PostMapping("/loginadmin")

```



DESIGN PATTERNS USED:

1. Factory Method Pattern: The Question class is an abstract class that defines the **getQuestion()** method. This method is used as a factory method to create different types of questions, such as MCQQuestion and TrueFalseQuestion. The Quiz class uses this method to create the questions for the quiz.
2. Strategy Pattern: The Answer class is an abstract class that defines the **evaluate()** method. This method is implemented by the concrete MCQAnswer and TrueFalseAnswer classes, which represent different strategies for evaluating answers to multiple-choice and true/false questions, respectively.
3. Singleton Pattern: The Quiz class uses a private constructor and a static instance variable to implement the Singleton pattern, ensuring that only one instance of the Quiz class can exist at a time.
4. Template Method Pattern: The Quiz class defines the **run()** method, which serves as a template method that defines the overall structure of the quiz. The MCQQuestion and TrueFalseQuestion classes override the **display()** method, which is a hook method that allows the subclasses to customize the behavior of the template method.

These are the main design patterns we have used in our project QUIZZINATOR

INDIVIDUAL CONTRIBUTIONS:

Aryan Kapoor - Handled the Student/User Class and Frontend

Avijit Panda - Handled the Professor Class and Frontend

Bharath D H - Handled the Admin and Scoreboard Class

Ravi Teja - Handled the Question Class

Github link : https://github.com/Aryan-11-Kapoor/QuizApp_OOADJ