

QUIZ MANAGER

Final Advanced Java Project

ABSTRACT-USERGUIDE & TECHNICAL SPECIFICATION



Sabari Nilash KURAPATI
[M.sc (SE)]

Table of Contents

1. Subject Description

2. Subject Analysis

2.1 Major Features

2.2 Application feasibility

2.3 Expected Results

2.4 Scope of the application

2.5 UML Scope

2.6 Technology

2.7 Objectives and concentrations

2.8 User Interface

3. Conception

3.1 Home Page

3.2 Admin Login

3.3 Register Questions

3.4 Student Registration

3.5 Student Login

3.6 Quiz Test

3.7 Results

4 Future Scopes

5 Bibliography

1. Subject Description:

This project is made to provide an evaluation of the advanced java course the goal of this project is to develop a program (API oriented, Web-based) that helps in dealing with quiz assessments.

The usual problem while preparing an evaluation is to:

- Constitute an appropriate evaluation regarding of the required level
- Reuse former questions
- Organize sample evaluations
- Correct automatically the MCQ questions.

2. Subject Analysis:

2.1 Major Features

- Authentication: A user should have a valid login in-order to go into the application.
- Admin: Can be able to create the quiz questions and review the scores.
- Student: Can be able to take the test and view scores.

2.2 Application Feasibility

- The application is developed with Java, Springs and hibernate. The SQL calls are dynamically called by using H2 Data Base connection for database.

- We have used Java, Spring MVC, Boot, ORM and Hibernate in the backend and JSP pages in the front-end.

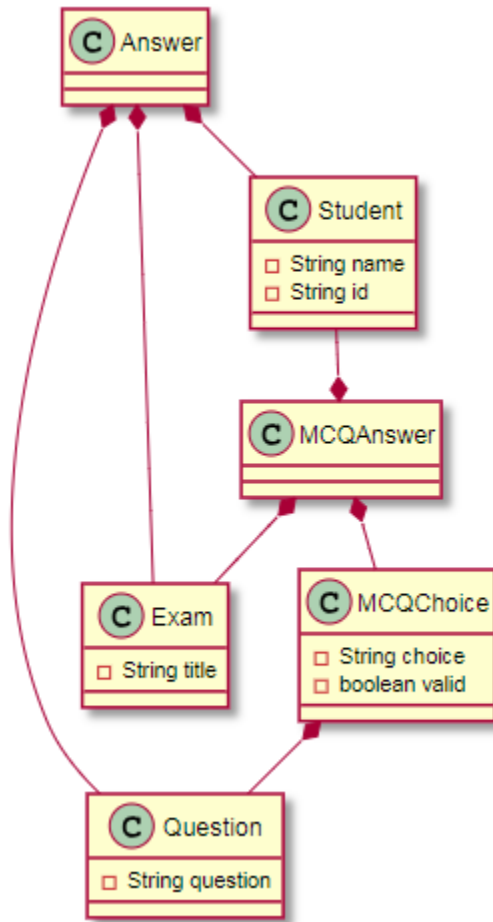
2.3 Expected Results

- The expected results of the project are making a quiz Manager Application. There would be two flows of the application. Admin and Student, Both will be having different logins. Based on the login either Admin or Student.
- Admin: Admin will be able to create the questions
- Student: Will be able to take the test and get the score of his after the test.

2.4 Scope of the Application

- The application management is restricted to the authenticated users. It manages both the students and the users. This is helpful for the dynamic management.

2.5 UML Scope



2.6 Technology

The implementation of this project has been done by using Frontend (HTML, CSS, JSP, JavaScript), Backend (Java 8), H2 (Data Base), Business Logic Framework (Hibernate, Spring ORM, Spring Boot, Spring MVC) and Server (Tomcat 9).

2.7 Objectives and concentrations:

This Quiz Management Application handles 2 Identities

-Student

-Admin

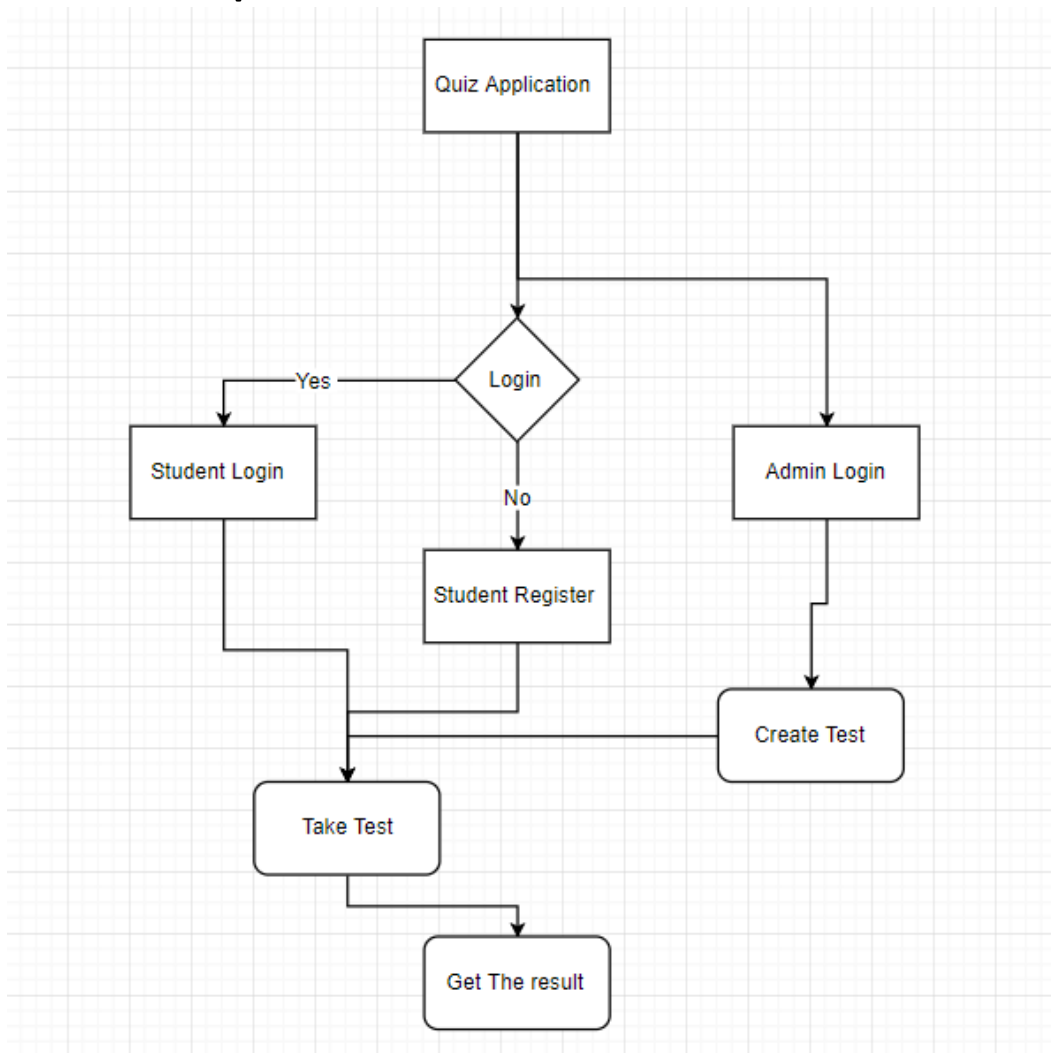
Students are having privilege to create the account with valid User Name and Password, Then attend the quiz, view the result at the same time.

Admin on the other hand is able to create Quiz Questions.

2.8 User Interface

Application will be accessed through a Browser Interface. The interface would be viewed best using 1024 x 768 and 800 x 600 pixels resolution setting. The software would be fully compatible with Microsoft Internet Explorer for version 6 and above. No user would be able to access any part of the application without logging on to the system.

3. Conception



3.1 Home Page

Here we have implemented many service calls. The Spring MVC architecture makes the code very feasible to understand



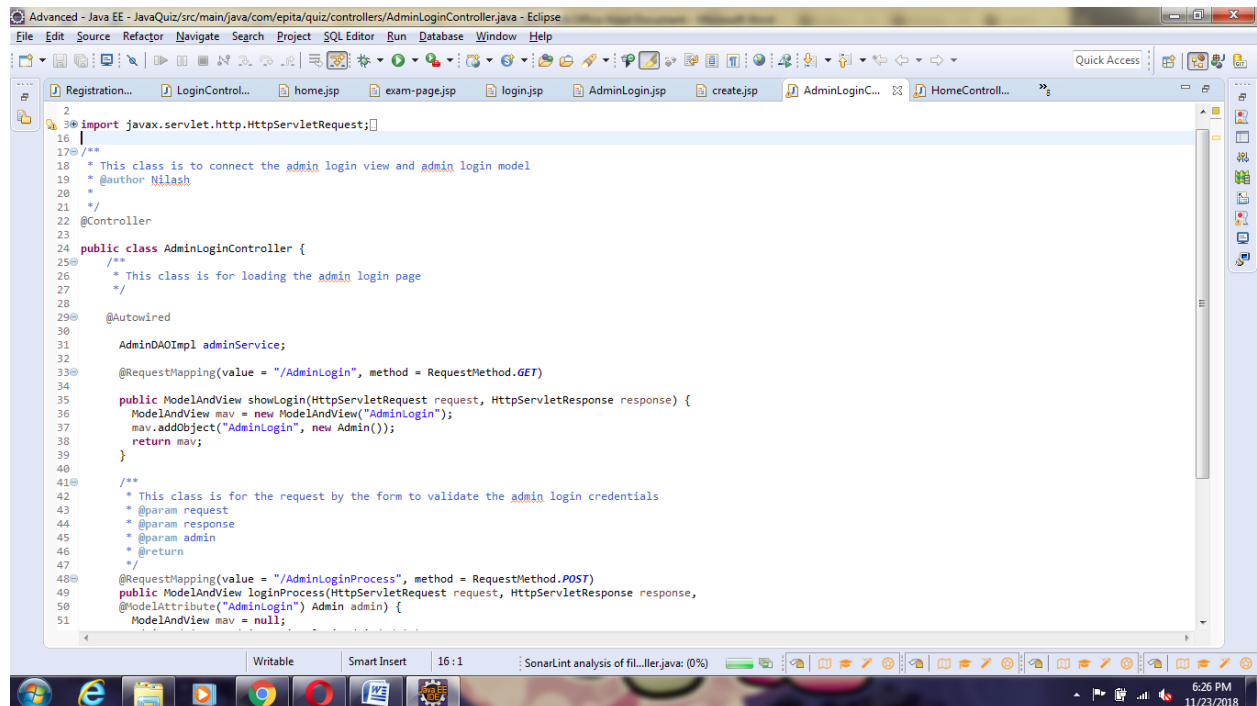
3.2 Admin Login

By providing this functionality, the admin can add questions with respective options and perform all actions required for the quiz application.

A screenshot of the 'Admin Login' form. The title 'Admin Login' is centered at the top in a large, bold, black font. Below the title, there are two input fields. The first is labeled 'Email' and contains the text 'nilash@gmail.com'. The second is labeled 'Password:' and contains four dots. Below the password field is a 'Login' button.

Authentication:

The web application cannot move forward without the login. There are two logins for student and admin. And a register page for the student, in-order to login.

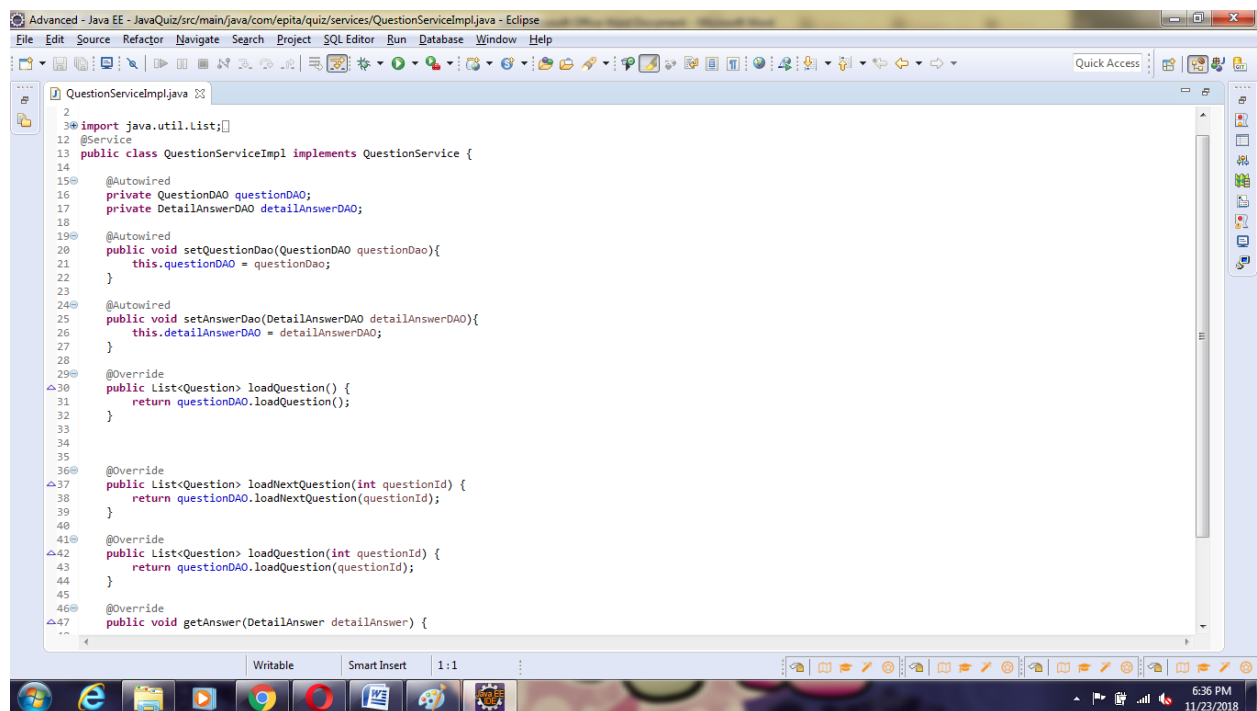


```
1 2
3  import javax.servlet.http.HttpServletRequest;
4
5  16
6  17 /**
7   * This class is to connect the admin login view and admin login model
8   * @author Nilash
9   *
10  */
11  @Controller
12
13  23
14  24 public class AdminLoginController {
15  25 /**
16   * This class is for loading the admin login page
17   */
18
19  29 @Autowired
20
21  30 AdminDAOImpl adminService;
22
23  32
24  33 @RequestMapping(value = "/AdminLogin", method = RequestMethod.GET)
25
26  34
27  35 public ModelAndView showLogin(HttpServletRequest request, HttpServletResponse response) {
28  36     ModelAndView mav = new ModelAndView("AdminLogin");
29  37     mav.addObject("AdminLogin", new Admin());
30  38     return mav;
31  39 }
32
33  40
34  41 /**
35   * This class is for the request by the form to validate the admin login credentials
36   * @param request
37   * @param response
38   * @param admin
39   * @return
40   */
41
42  48 @RequestMapping(value = "/AdminLoginProcess", method = RequestMethod.POST)
43
44  49 public ModelAndView loginProcess(HttpServletRequest request, HttpServletResponse response,
45  50 @ModelAttribute("AdminLogin") Admin admin) {
46  51     ModelAndView mav = null;
```

3.3 Register Questions

Add Questions

Question	Who is your Java Professor '
Option1	Nilash
Option2	Thomas
Option3	Shanthanu
Option4	Deepak
Correct Answer	Thomas



The screenshot shows the Eclipse IDE with the file `QuestionServiceImpl.java` open. The code is as follows:

```
2
3 * import java.util.List;
12 @Service
13 public class QuestionServiceImpl implements QuestionService {
14
15     @Autowired
16     private QuestionDAO questionDAO;
17     private DetailAnswerDAO detailAnswerDAO;
18
19     @Autowired
20     public void setQuestionDao(QuestionDAO questionDao){
21         this.questionDAO = questionDao;
22     }
23
24     @Autowired
25     public void setAnswerDao(DetailAnswerDAO detailAnswerDAO){
26         this.detailAnswerDAO = detailAnswerDAO;
27     }
28
29     @Override
30     public List<Question> loadQuestion() {
31         return questionDAO.loadQuestion();
32     }
33
34
35
36
37     @Override
38     public List<Question> loadNextQuestion(int questionId) {
39         return questionDAO.loadNextQuestion(questionId);
40     }
41
42     @Override
43     public List<Question> loadQuestion(int questionId) {
44         return questionDAO.loadQuestion(questionId);
45     }
46
47     @Override
48     public void getAnswer(DetailAnswer detailAnswer) {
```

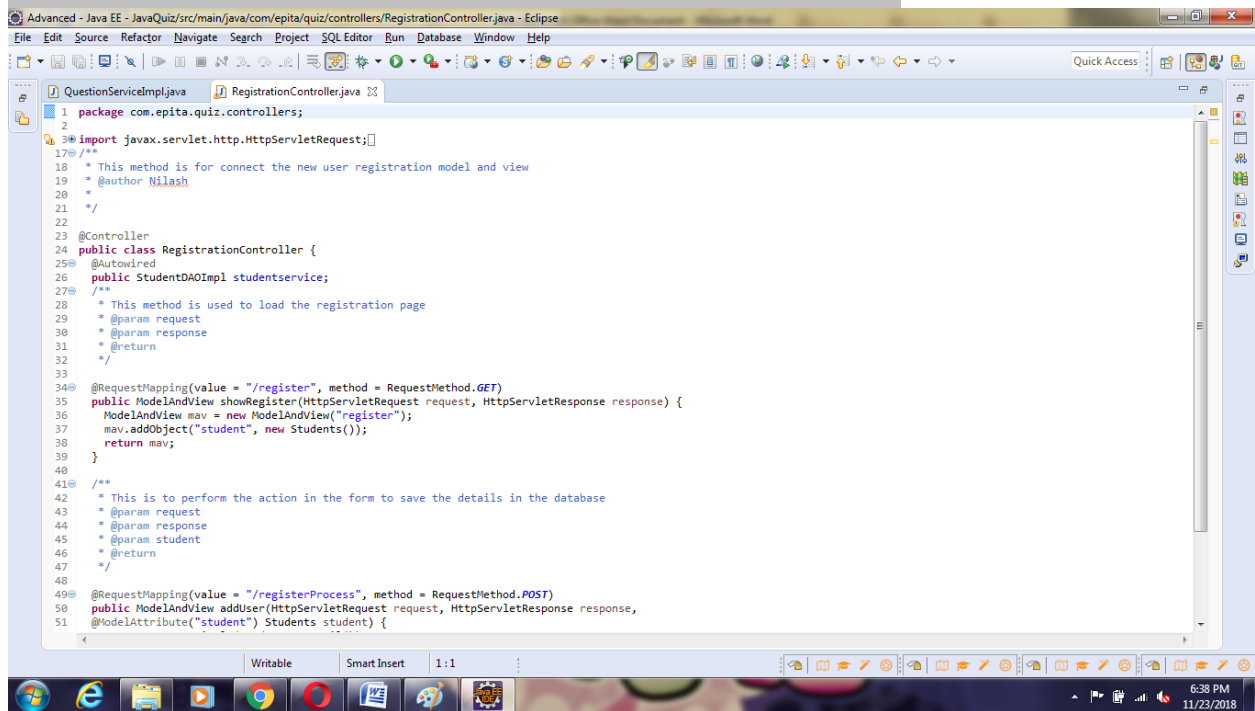
3.4 Student Registration

Student Registration

UserName

Password

Email



The screenshot shows the Eclipse IDE with the file `RegistrationController.java` open. The code is as follows:

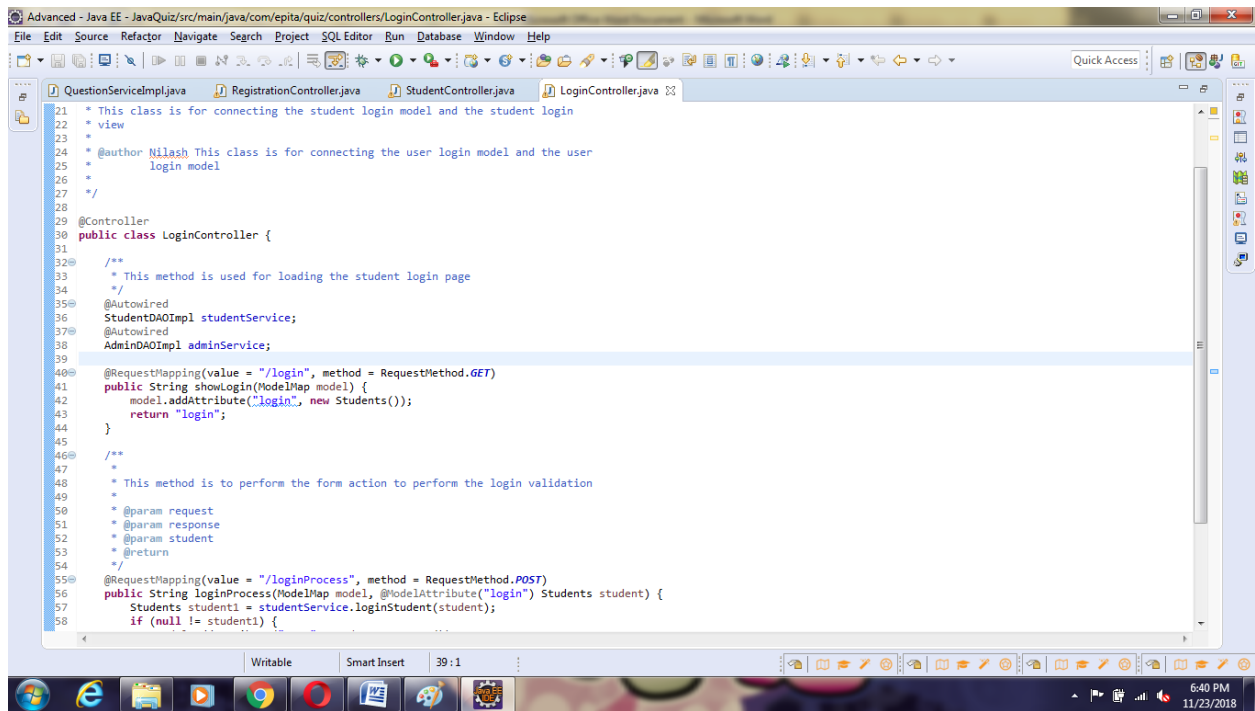
```
1 package com.epita.quiz.controllers;
2
3 import javax.servlet.http.HttpServletRequest;
4
5 /**
6  * This method is for connect the new user registration model and view
7  * @author Nilash
8  */
9
10 @Controller
11 public class RegistrationController {
12     @Autowired
13     private StudentDAOImpl studentservice;
14
15     /**
16      * This method is used to load the registration page
17      * @param request
18      * @param response
19      * @return
20      */
21     @RequestMapping(value = "/register", method = RequestMethod.GET)
22     public ModelAndView showRegister(HttpServletRequest request, HttpServletResponse response) {
23         ModelAndView mav = new ModelAndView("register");
24         mav.addObject("student", new Students());
25         return mav;
26     }
27
28     /**
29      * This is to perform the action in the form to save the details in the database
30      * @param request
31      * @param response
32      * @param student
33      * @return
34      */
35     @RequestMapping(value = "/registerProcess", method = RequestMethod.POST)
36     public ModelAndView addUser(HttpServletRequest request, HttpServletResponse response,
37                               @ModelAttribute("student") Students student) {
38     }
```

3.5 Student Login

Email

Password:

Student and Admin Controller:

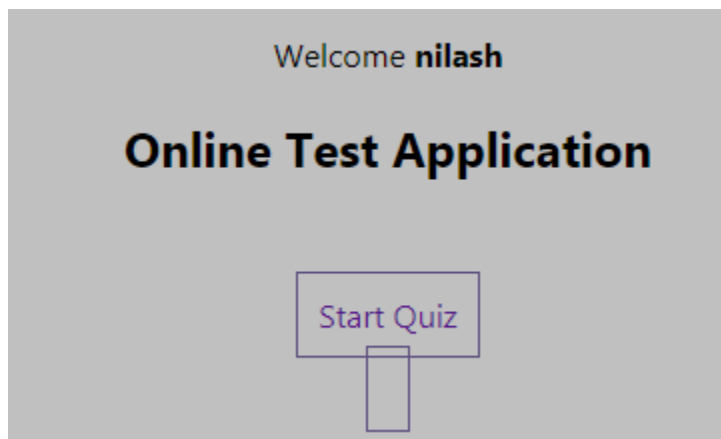


The screenshot shows the Eclipse IDE with the file `LoginController.java` open. The code is as follows:

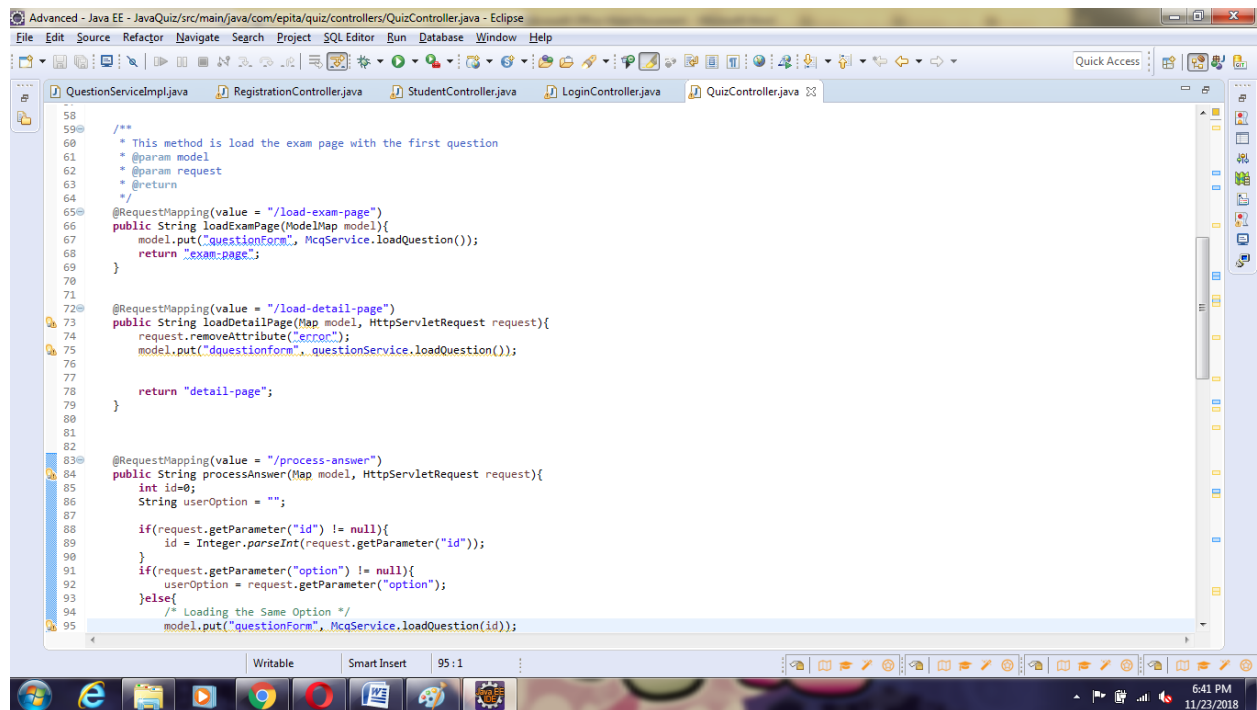
```
21 * This class is for connecting the student login model and the student login
22 * view
23 *
24 * @author Nilash This class is for connecting the user login model and the user
25 * login model
26 *
27 */
28
29 @Controller
30 public class LoginController {
31
32     /**
33      * This method is used for loading the student login page
34      */
35     @Autowired
36     StudentDAOImpl studentService;
37     @Autowired
38     AdminDAOImpl adminService;
39
40     @RequestMapping(value = "/login", method = RequestMethod.GET)
41     public String showLogin(ModelMap model) {
42         model.addAttribute("login", new Students());
43         return "login";
44     }
45
46     /**
47      * This method is to perform the form action to perform the login validation
48      *
49      * @param request
50      * @param response
51      * @param student
52      * @return
53      */
54     @RequestMapping(value = "/loginProcess", method = RequestMethod.POST)
55     public String loginProcess(ModelMap model, @ModelAttribute("login") Students student) {
56         Students student1 = studentService.loginStudent(student);
57         if (null != student1) {
58             // ...
59         }
60     }
61 }
```

Authentication successful:

After successfully registering and logging into the application. The student can take the test.



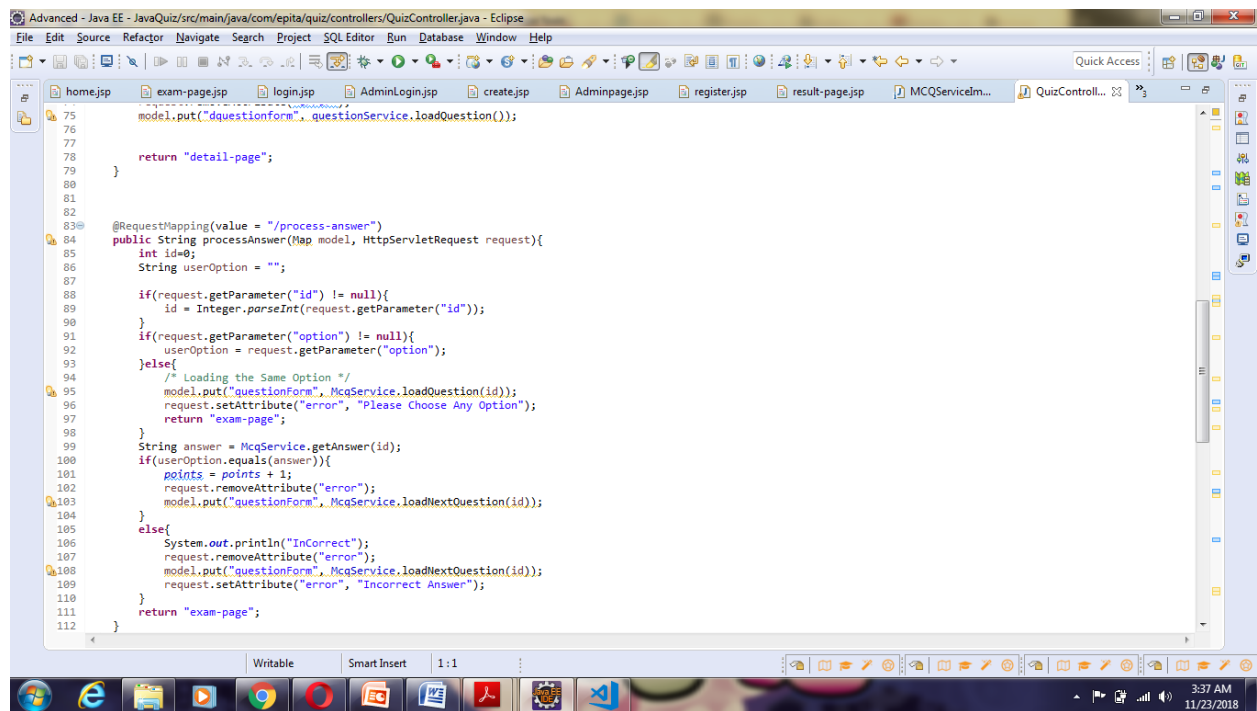
3.6 Quiz Test



```
58  
59  
60 /**  
61  * This method is load the exam page with the first question  
62  * @param model  
63  * @param request  
64  * @return  
65  */  
66 @RequestMapping(value = "/load-exam-page")  
67 public String loadExamPage(ModelMap model){  
68     model.put("questionForm", McqService.loadQuestion());  
69     return "exam-page";  
70 }  
71  
72 @RequestMapping(value = "/load-detail-page")  
73 public String loadDetailPage(Map model, HttpServletRequest request){  
74     request.removeAttribute("error");  
75     model.put("questionForm", questionService.loadQuestion());  
76  
77     return "detail-page";  
78 }  
79  
80  
81  
82  
83 @RequestMapping(value = "/process-answer")  
84 public String processAnswer(Map model, HttpServletRequest request){  
85     int id=0;  
86     String userOption = "";  
87  
88     if(request.getParameter("id") != null){  
89         id = Integer.parseInt(request.getParameter("id"));  
90     }  
91     if(request.getParameter("option") != null){  
92         userOption = request.getParameter("option");  
93     }else{  
94         /* Loading the Same Option */  
95         model.put("questionForm", McqService.loadQuestion(id));
```

3.7 Verifying the MCQ-Choice questions and providing the answers:

Here we have implemented the model class to reuse the functionality for all questions and making the code easily accessible.



```
75     model.put("questionForm", questionService.loadQuestion());
76
77     return "detail-page";
78 }
79
80
81
82
83 @RequestMapping(value = "/process-answer")
84 public String processAnswer(Map model, HttpServletRequest request){
85     int id=0;
86     String userOption = "";
87
88     if(request.getParameter("id") != null){
89         id = Integer.parseInt(request.getParameter("id"));
90     }
91     if(request.getParameter("option") != null){
92         userOption = request.getParameter("option");
93     }else{
94         /* Loading the Same Option */
95         model.put("questionForm", McqService.loadQuestion(id));
96         request.setAttribute("error", "Please Choose Any Option");
97         return "exam-page";
98     }
99     String answer = McqService.getAnswer(id);
100     if(userOption.equals(answer)){
101         points = points + 1;
102         request.removeAttribute("error");
103         model.put("questionForm", McqService.loadNextQuestion(id));
104     }
105     else{
106         System.out.println("Incorrect");
107         request.removeAttribute("error");
108         model.put("questionForm", McqService.loadNextQuestion(id));
109         request.setAttribute("error", "Incorrect Answer");
110     }
111     return "exam-page";
112 }
```

Providing the result to student after the test:
Here we will be evaluating the student marks based on his answers.

You have answered 0

Your score 0

4 Future Scopes

Edit functionality Like Delete and Update.

Student Database with Scores.

Advanced Security features.

Advanced UI.

Advanced Navigation features.

Timer – Timeout Functionality.

5 Bibliography

<http://thomas-broussard.fr/work/java/courses/project/advanced.xhtml>

<http://stackoverflow.com>

https://www.youtube.com/watch?v=ZfOljMf7L-o&list=PLOUYE-KsFYc_0ekC_3EEAlkiRuElaymmJ