Technical document

Advanced Java project: Quiz Manager

Roshnee Meena
Sushant Attada

Table of Contents

1. Subject Description:

The goal of this project is to develop a program (API oriented, Web-based) that helps in dealing with quiz assessments.

2. Subject Analysis:

2.1 Major Features

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

2.2 Application Feasibility

- The application is developed with Java, Springs and hibernate. The SQL calls are dynamically called by using derby connection for database.
- We have used Java, Springs and Hibernate in the backend and JSP pages in the front-end.

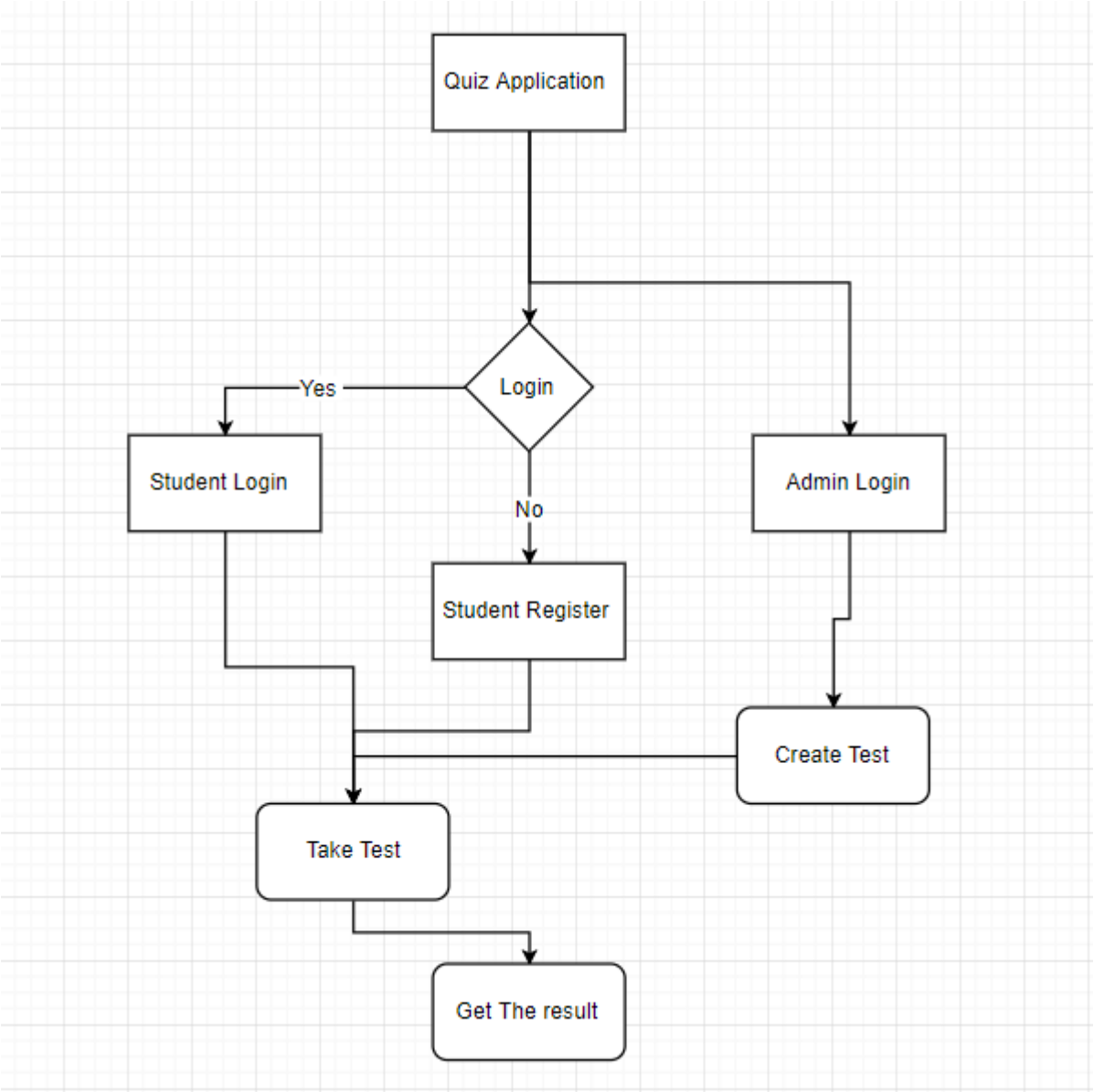
2.3 Expected Results

- The expected results of the project are make a quiz application. There would be two flows of the application. One w.r.t Admin and another w.r.t Student. Both will be having different logins. Based on the login either Student/Admin.
- 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.

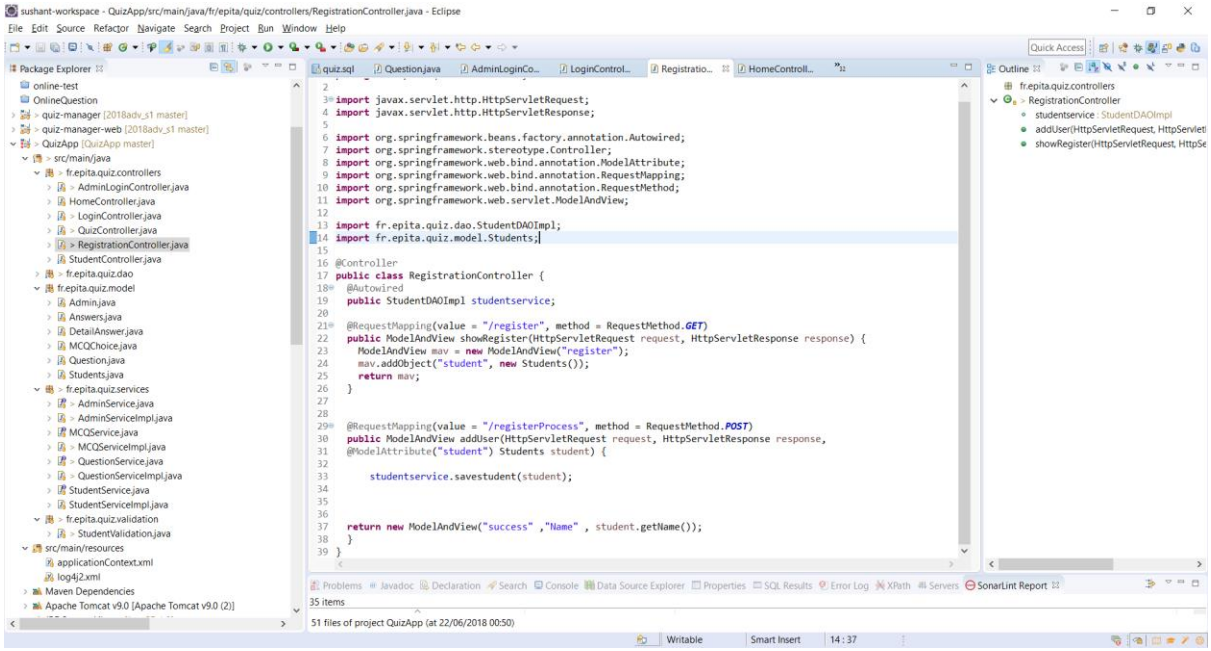
3. Conception



3.1 Authentication:

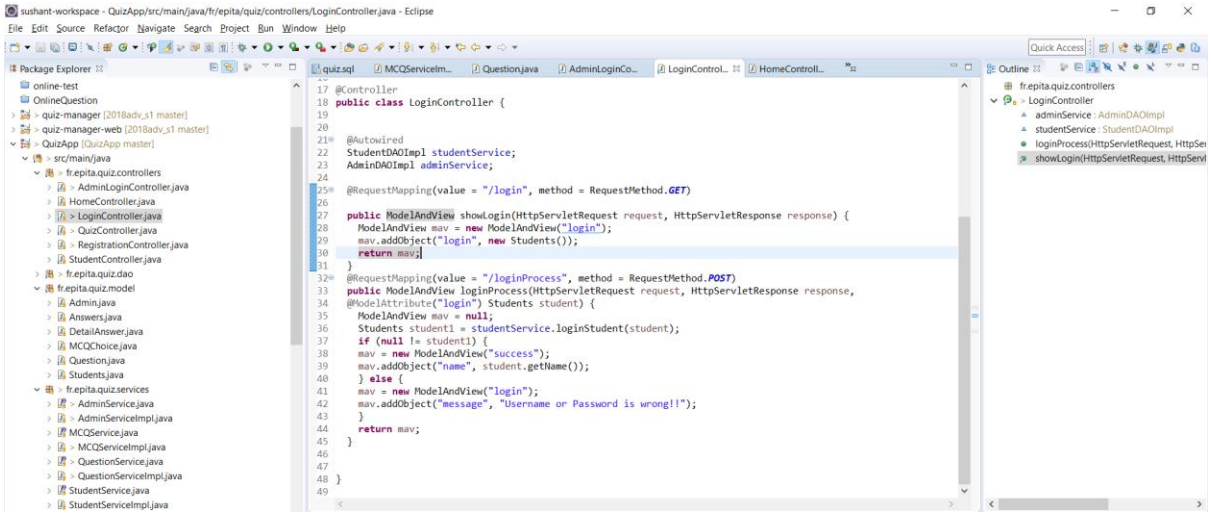
The web application cannot move forward without the login. There are two logins for student and admin. And a register page for the student, in-order to login.

Student Register Controller:



```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;

import fr.epita.quiz.dao.StudentDAOImpl;
import fr.epita.quiz.model.Students;

@Controller
public class RegistrationController {
  @Autowired
  public StudentDAOImpl studentservice;

  @RequestMapping(value = "/register", method = RequestMethod.GET)
  public ModelAndView showRegister(HttpServletRequest request, HttpServletResponse response) {
    ModelAndView mav = new ModelAndView("register");
    mav.addObject("student", new Students());
    return mav;
  }


  @RequestMapping(value = "/registerProcess", method = RequestMethod.POST)
  public ModelAndView addUser(HttpServletRequest request, HttpServletResponse response,
  @ModelAttribute("student") Students student) {

      studentservice.savestudent(student);



  return new ModelAndView("success" ,"Name" , student.getName());
  }
}
```

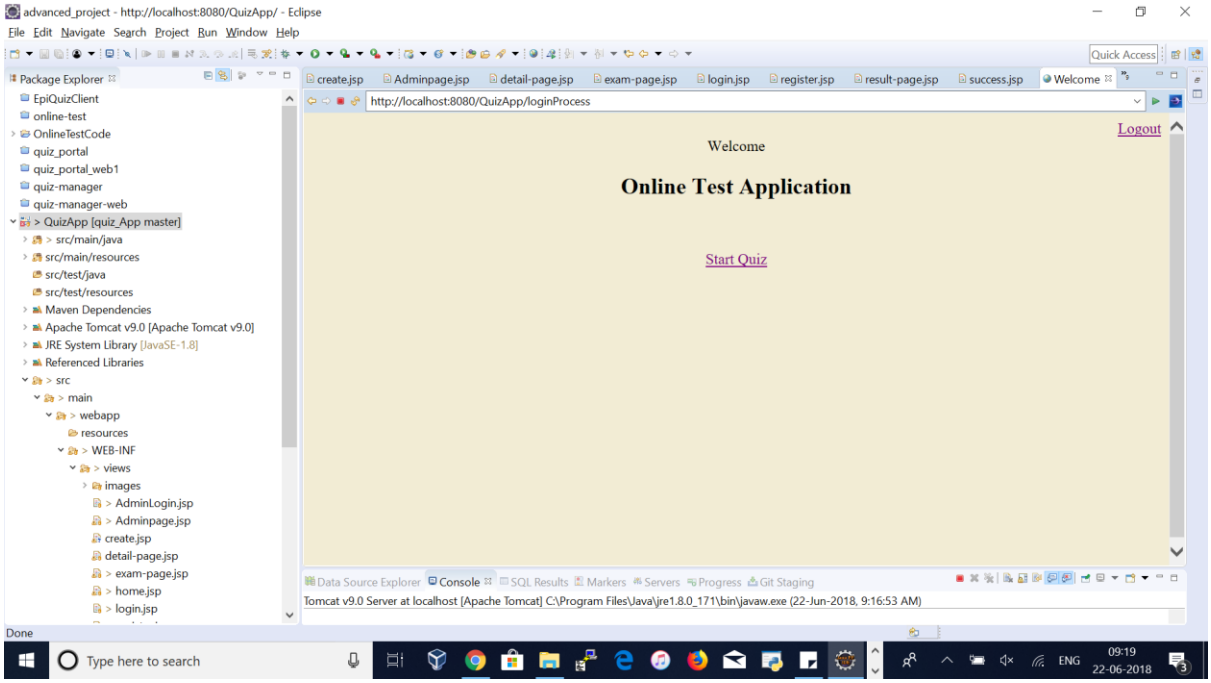
Student and Admin Controller:



```
@Controller
public class LoginController {


  @Autowired
  StudentDAOImpl studentService;
  AdminDAOImpl adminService;

  @RequestMapping(value = "/login", method = RequestMethod.GET)

  public ModelAndView showLogin(HttpServletRequest request, HttpServletResponse response) {
    ModelAndView mav = new ModelAndView("login");
    mav.addObject("login", new Students());
    return mav;
  }
  @RequestMapping(value = "/loginProcess", method = RequestMethod.POST)
  public ModelAndView loginProcess(HttpServletRequest request, HttpServletResponse response,
  @ModelAttribute("login") Students student) {
    ModelAndView mav = null;
    Students student1 = studentService.loginStudent(student);
    if (null != student1) {
    mav = new ModelAndView("success");
    mav.addObject("name", student.getName());
    } else {
    mav = new ModelAndView("login");
    mav.addObject("message", "Username or Password is wrong!!");
    }
    return mav;
  }


}
```

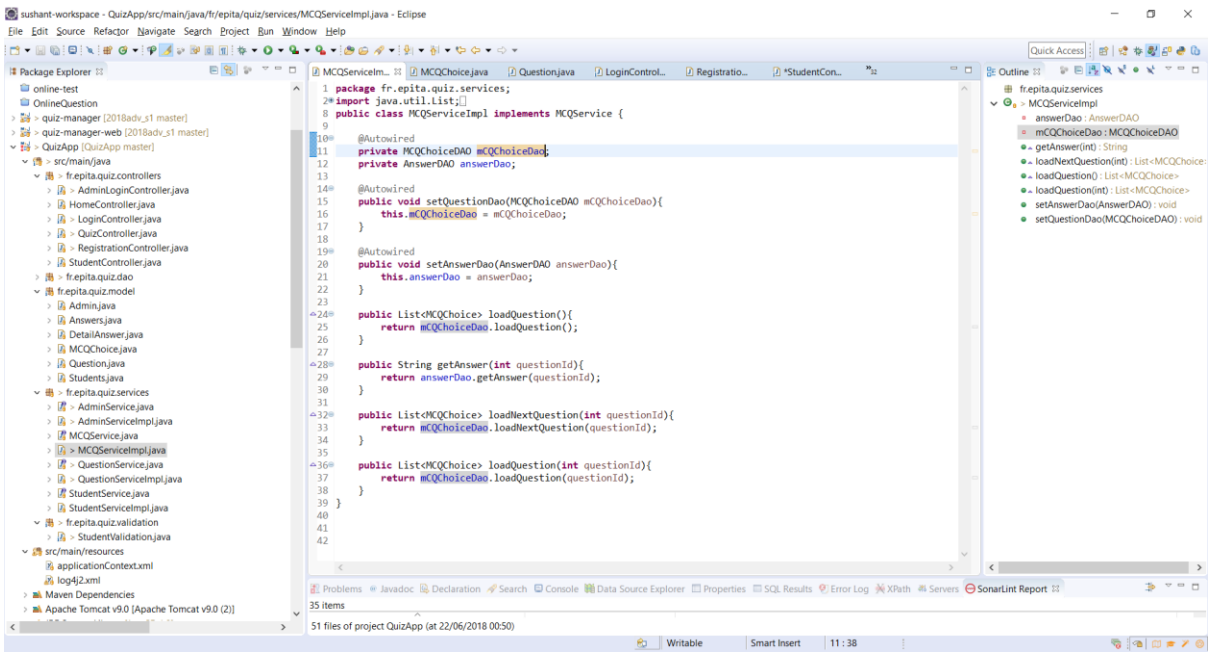
3.2 Authentication successful:

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



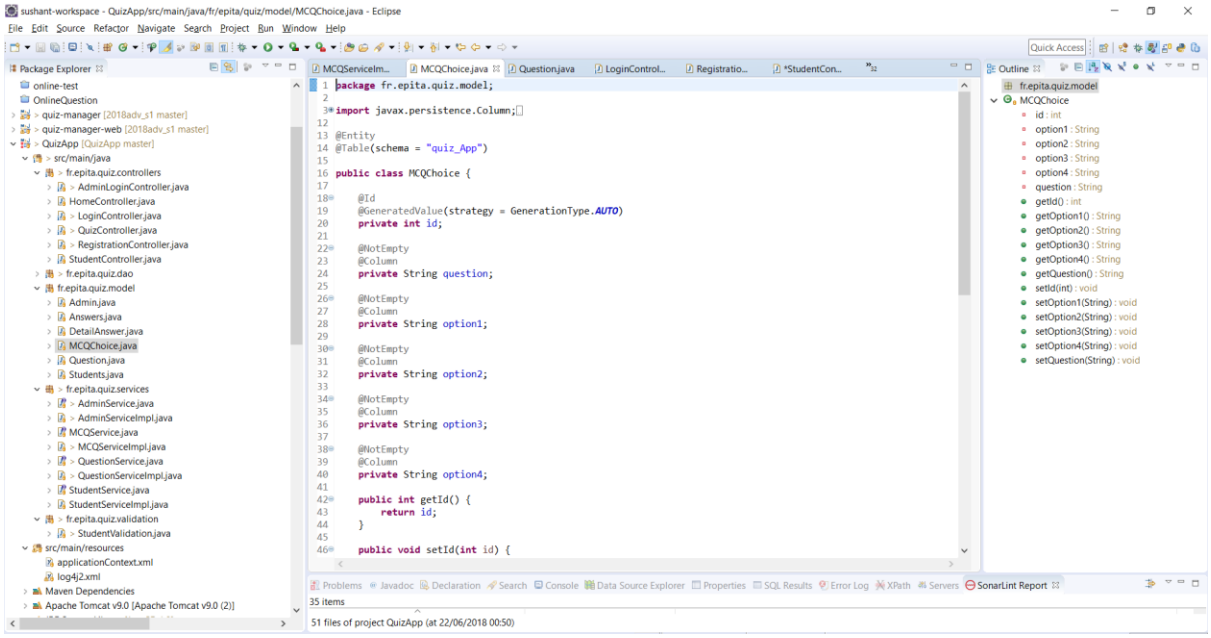
3.3 Start the quiz application:

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



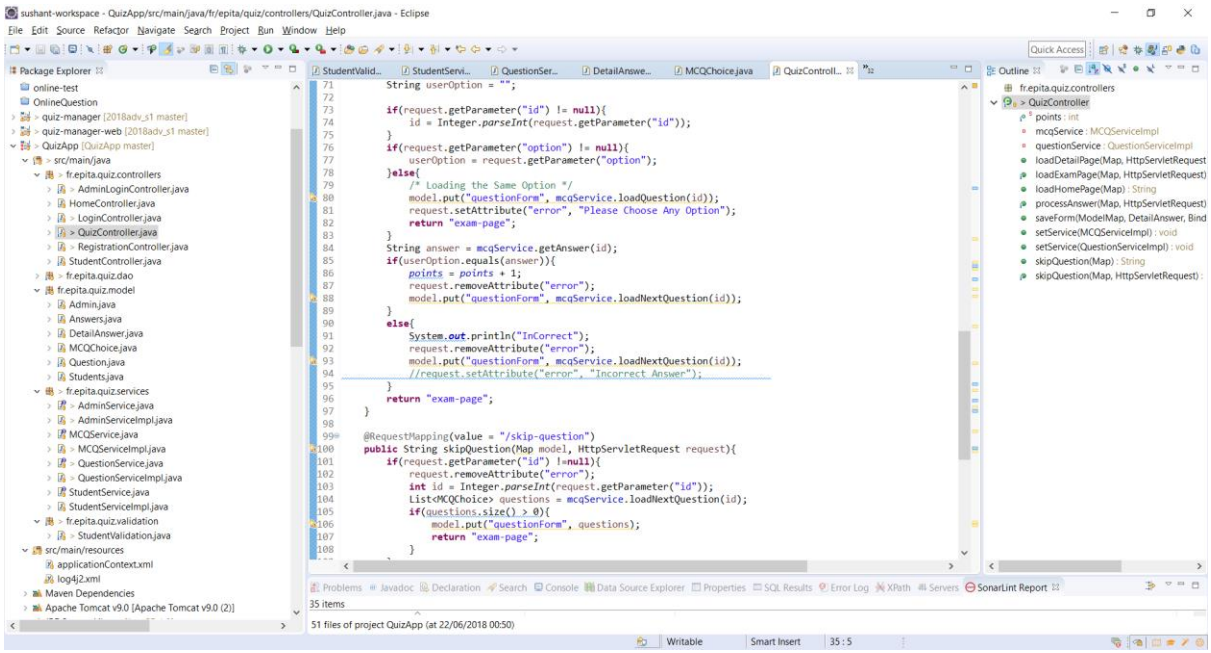
3.4 Verifying the MCQ-Choise questions and providing the answers:

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



3.5 Providing the result to student after the test:

Here we will be evaluating the student marks based on his answers.



3.6 Admin login:

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