TECHNICAL SPECIFICATIONS

Advanced Java – Quiz Application

Naga Sindhura Devi VEGI

Table of Contents

Introduction

This document will propose all features and technical specification of the project. It contains details about objectives, scope limitation, primary requirements, possible project risks, database design, and application flow.

The goal of this project is to develop an application (API oriented, Webbased) that deals with quiz assessments.

The usual problem while preparing an evaluation, are to:

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

The implementation of this project has been done by using Frontend (HTML, Angular JS), Backend (REST, Java 8), Database (Derby), Business Logic Framework (Hibernate, Spring) and Server (Tomcat).

Objectives and concentrations:

This Quiz Management Application handles 2 Identities -Student

-Admin

Students are having privilege to create the account, choose the relevant quiz type and quiz name, attend the quiz, view the result at the same time.

Admin on the other hand is able to create his/her account, edit/delete/view the registered student's details with their quiz score. Apart, from managing the student's details they are able to create the quiz specifying name, and type.

Scope and limitations:

- The system handles all the operations, and generates reports as soon as the test is finish, that includes name, mark, time spent to solve the exam.
- Allow students to see or display his answers after the exam is finish.
- The type of questions is only multiple choice right.
- Manage the students, the admin/professors can add/edit/delete student and review their scores

Software Requirements:

 Technology Used

BACKEND/WEBSERVER	FRONTEND	DATABASE
JAVA - version 9	HTML5	DERBY
SPRING – version 5.0.4	CSS3	DBeaver –version 5.2.5
JSP – version 2.2	JAVASCRIPT	HIBERNATE– version 5.2.14
TOMCAT – version 9	SERVLET	

Data Model:

USERS	Question
PK User_Role_ID Mail Enable Password User_name Role	PK ID Correctanswer Answer 1 Answer 2 Answer 3 Answer 4 Question Type QuizName

We kept the data model simple with 2 tables called USERS and QUESTION

Testing:

To satisfy the requirements and check whether every build services in the application are executed as expected, one of the earliest, testing efforts performed on the code is Unit Testing. To quickly test any new code or changes to existing code without the overhead and additional time involved in tasks such as server configuration, services setup and application deployment, we integrated the popular testing framework JUnit Framework to execute the unit test on every small code module.

Our objectives are in writing and executing the tests

- We want to code and run the tests without leaving the IDE (Eclipse).

- There should be no special deployment of the code required
- We should be able to exploit other code analysis tools such as Metrics and find bugs right from within the IDE so we can find any bugs right away and fix those issues.

studentLogi... QuestionLis... selectQuesti... application... quiz-manage... TestMCQAndQ...

```
54    @Inject
55    CreateQuestionDAO questDAO;
56    @Inject
57    UsersDAO userDAO;
58    @Inject
59    SessionFactory factory;
60
61    // @Test
62    public void testSaveOrUpdateQuestion() {
63        final Session session = factory.openSession();
64        final Transaction tx = session.beginTransaction();
65        final Questions question = new Questions();
66        question.setQuestion("How to configure Hibernate?");
67        question.setType(TypeOFQuestions.MCQ);
68        question.setCorrectanswer(
69                "add dependency and create a bean of session factory with data source properties and hibernat
70        questDAO.create(question);
71
72        questDAO.create(question);
73        tx.commit();
74        session.close();
75    }
76
77    // @Test
78    public void testGetQuizType() {
79        final Questions question = new Questions();
80        question.setType(TypeOFQuestions.MCQ);
81        List<Questions> stri = questDAO.getQuizName(question);
82        System.out.println(stri.size());
83    }
```

Spring Integration with Hibernate

We integrated hibernate framework with this spring application.

No.	Method	Description
1)	void persist(Object entity)	persists the given object.
2)	Serializable save(Object entity)	persists the given object and returns id.
3)	void saveOrUpdate(Object entity)	persists or updates the given object. If id is found, it updates the record otherwise saves the record.
4)	void update(Object entity)	Updates the given object.
5)	void delete(Object entity)	deletes the given object on the basis of id.
6)	Object get(ClassentityClass, Serializable id)	returns the persistent object on the basis of given id.
7)	Object load(ClassentityClass, Serializable id)	returns the persistent object on the basis of given id.

8)	List loadAll(Class entityClass)	returns the all the persistent objects.

Application Module:

This section gives a functional requirement that applicable to the Quiz Application.

There are two sub modules in this phase.

- Admin module.
- User module.
- **The functionality of each module is as follows:**
- **User module**: The candidate will logon to the application and take his quiz. He can check the list of question type, name and then attend the quiz as per his preference. The candidate will get result immediately after the completion of the examination.
- **Admin module:** The professor/admin will logon to the application and take his/her quiz. They can manage the student identities, create quiz – MCQ type, ASSOCIATIVE type, OPEN type, and view the marks of every student.

Major Features

- **Login/Sign Up Authentication**

Manage Identities

- Add Student
- Update Student
- Delete Student

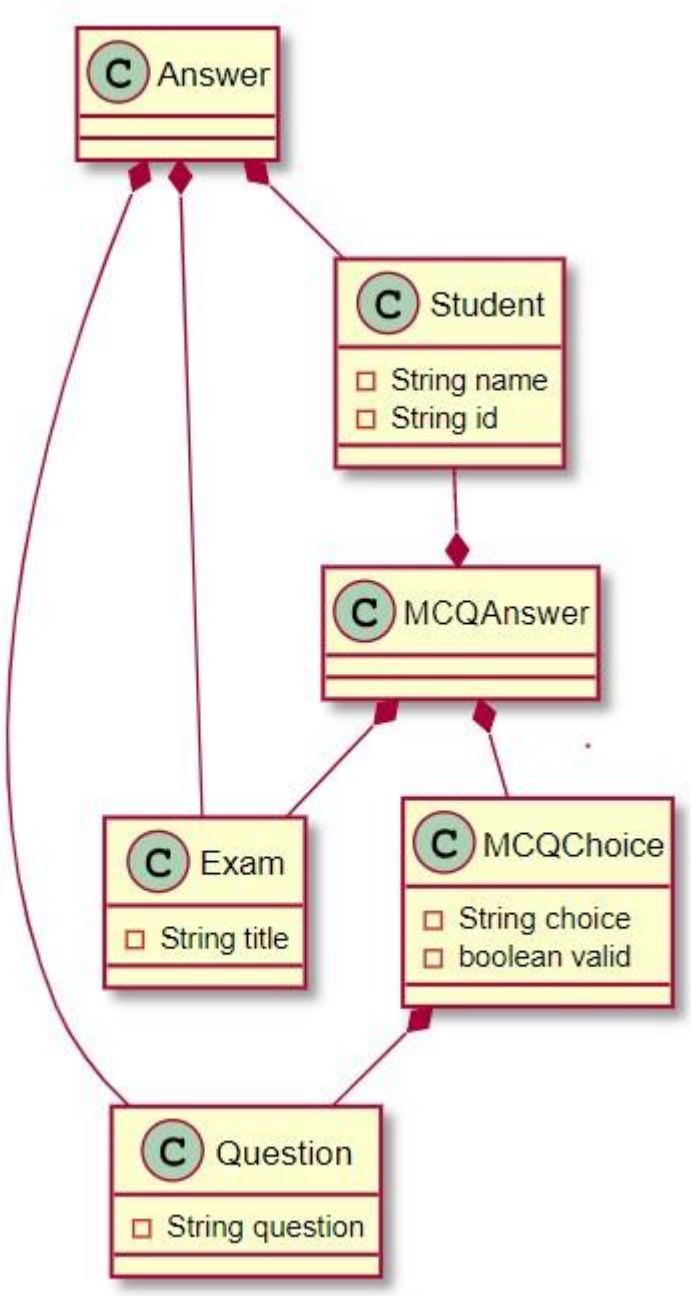
Manage Questions

- Add Questions
- Update Questions
- Delete Questions
- Search Questions

Take Quiz

- View Questions

Global Application Flow:



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.

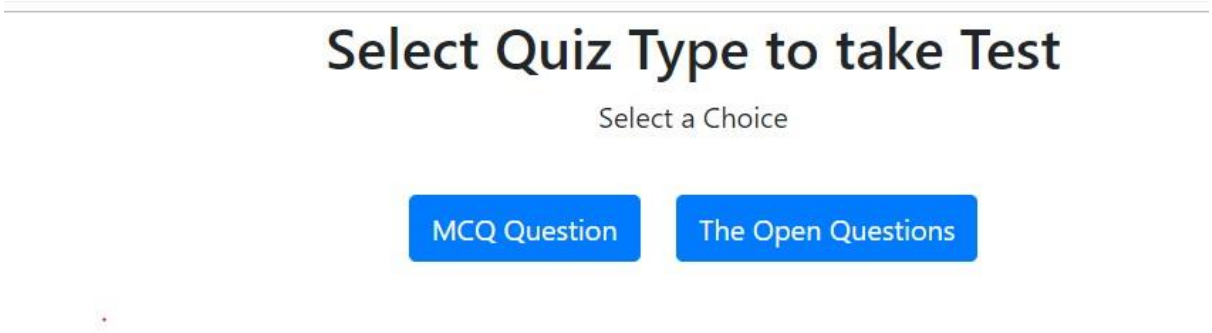
User Register Controller:

studentLogi... adminLogin.html QuestionLis... allclasses-f... registerUse... selectQuesti... application... quiz-manage...

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Quiz manager application</title>
    <link rel="stylesheet" href="resources/css/style.css">
            <link href="resources/bootstrap/css/bootstrap.min.css" rel="stylesheet">
</head>
<body>
    <h1 align="center">Register User</h1>
    <div class="container">

        <form action="login" method="post">
            <div class="row">
                <div class="col-md-12 mb-3">
                    <label for="user_name">Username</label> <input name="user_name"
                        class="form-control" type="text" />
                </div>
                <div class="col-md-12 mb-3">
                    <label>Password</label> <input class="form-control" name="psswd"
                        type="psswd" />
                </div>
                <div class="col-md-12 mb-3">
                    <label>Mail</label> <input class="form-control" name="mail"
                        type="mail" />
                </div>
                <button class="btn btn-primary" type="submit" value="RegisterUser" name="registerUser">Register</button>
            </div>
        </form>
    </div>
```

3.2. Authentication successful:

After successfully registering and logging into the application. The user 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.

populateExam... studentLogi... QuestionLis... allclasses-f... registerUse... selectQuesti... application... quiz-manage... TestMC

```
<%@ taglib prefix="core" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
<link href="resources/bootstrap/css/bootstrap.min.css" rel="stylesheet">
</head>
<body>
    <div class="container">
        <div class="jumbotron">
            <div class="container">
                <h1 class="text-info" align="center">Question Paper</h1>
            </div>


            <div class="container" align="center">
                <iframe
                    src="http://free.timeanddate.com/clock/i6an01ok/n195/szw110/szh110/hoc000/hbw2/hfceee/cf100/hncccc/fdi76/mqc0
                    frameborder="0" width="110" height="110"></iframe>
            </div>
                    <div class="container" align="left">
                <b> <%
    if (session.getAttribute("userName") != null) {
 %> Welcome : <%
    out.print(session.getAttribute("userName"));
 %>
                    <%
                        }
                    %>
                </b>
            </div>
            <div class="container" align="right">
                <b> <a href="logout.html">LogOut</a></b>
            </div>
```

3.4.Start the quiz application:

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

1 . To prevent any method from overriding, we declare the method as,

- static
- const
- final
- abstract

2 . Which of the following variable declaration would NOT compile in a java program?

- int var;
- int VAR;
- int var1;
- int 1_var;

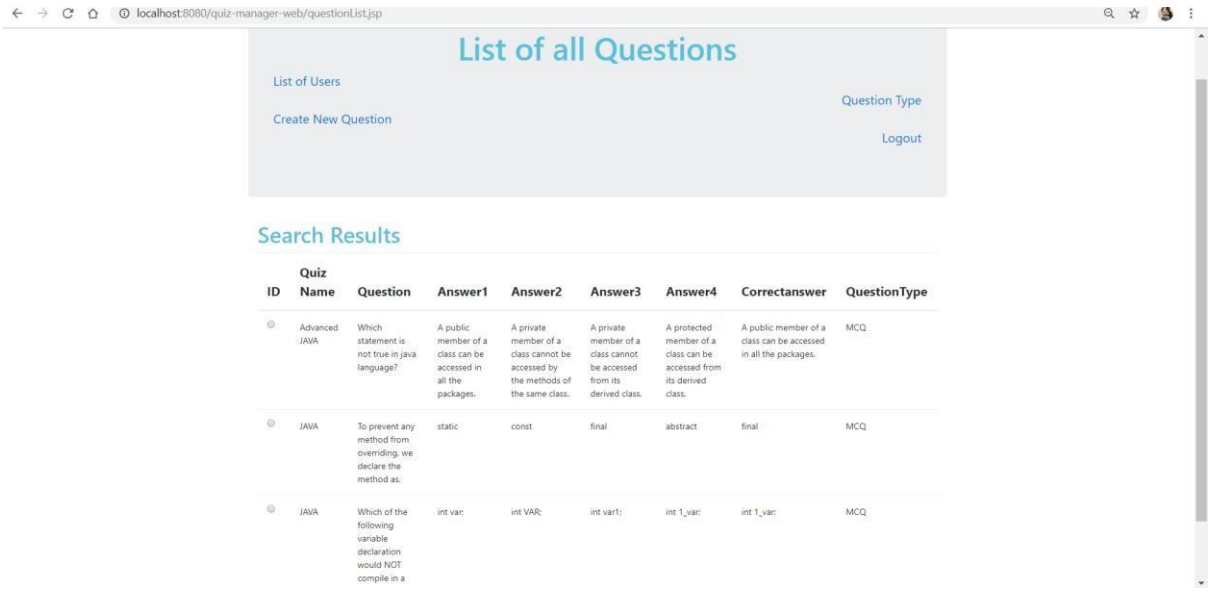
Submit

3.4.Verifying MCQ Questions:

User will get MCQ questions and admin can reset the questions

populateExam... studentLogi... QuestionLis... registerUse... application... quiz-manage... TestMCQAndQ... updateQuesti...

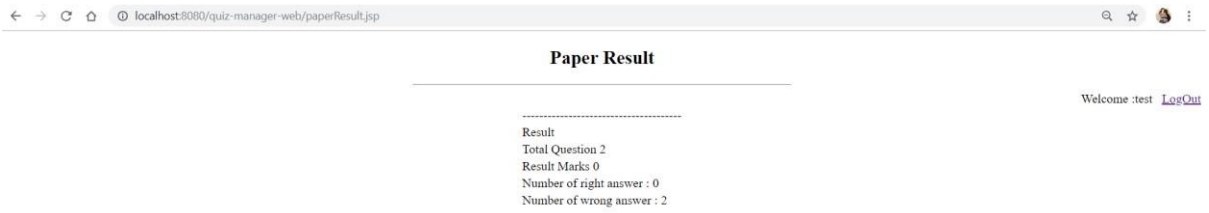
```
<%@page import="fr.epita.quiz.datamodel.Questions"%>
<%@page import="fr.epita.quiz.services.*"%>
<%@page import="java.util.List"%>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<%
    Questions addQuestion = (Questions) session.getAttribute("addQuestion");
%>
<head>
<%
    Boolean auth = (Boolean) session.getAttribute("authenticated");
    if (auth == null || !auth) {
%>

<meta http-equiv="refresh" content="0; URL=index.jsp">
<%
    }
%>

<link href="resources/bootstrap/css/bootstrap.min.css" rel="stylesheet">
</head>

                    <%
                    if (addQuestion == null){%>
                       <tr>
                            <td colspan="4">No result</td>
                       </tr>

                    <% } else{
                     %>
<body>
        <div class="container">
        <div>
            <div class="jumbotron" align="center">
                <div class="container">
                    <h1 class="text-info">Update Question</h1>
                </div>
                                <div align="left"><a href="<%=request.getContextPath() %>/usersService">List of Users</a></div>
```



3.5.Results: User will get result after submitting.

populateExam... | paperResult.jsp | studentLogi... | registerUse... | application... | quiz-manage...

```
20    <% int i=0; %>
21  <%--   <core:forEach var="questionPaperCommand" items="${questionPaperList}" >
22       <tr>
23         <td>
24          <core:out value="<%=i+1%>"></core:out>
25          <core:out value="."></core:out>
26          <core:out value="Question :"></core:out>
27          <core:out value="${questionPaperCommand.question}"></core:out></td>
28       </tr>
29       <tr>
30         <td>
31           <font color="green" ><core:out value="Answer : "></core:out></font>
32           <core:out value="${wrongAnswers}"></core:out>
33        </tr>
34       <% i++; %>
35   </core:forEach> --%>
36   <tr>
37     <td colspan="2" ><core:out value="--------------------------------------"></core:out></td>
38   </tr>
39   <tr>
40     <td><core:out value="Result"></core:out></td>
41   </tr>
42   <tr>
43     <td colspan="2">
44         <core:out value="Total Question"></core:out>
45         <core:out value="${totalQuestions}"></core:out>
46     </td>
47   </tr>
48   <tr>
49     <td colspan="2" >
50         <core:out value="Result Marks"></core:out>
51         <core:out value="${rightAnswer}"></core:out>
52     </td>
53   </tr>
54   <tr>
55     <td colspan="2" >
56        <core:out value="Number of right answer : "></core:out>
57        <core:out value="${rightAnswer}"></core:out>
58    </td>
59   </tr>
```



Bibliography:

https://github.com/thomasbroussard

https://thomas-broussard.fr/work/java/courses/project/advanced.xhtml