`

MCQ-Quiz Management Project
Project Documentation

Prepared By

Youbaraj Poudel

Msc Computer Science(Software Engineering) Epita Second Semester

||
||
||
||
||
||

Prepared to

Thomas Broussard

Submitted On : 23rd November 2018

`

Table Of Contents --

`

1. Subject Description

The main purpose of this assignment is to make a Quiz Management Java Rest API and appropriate web client to consume rest api created.This application allows the client to execute some essential REST API methods(POST,DELETE,PUT,GET).

In order to build this project, I have used sessionfactory for session management and H2 database for managing all the data required for this application.

I have used Java Rest Service as a backend and React Js as a Front End to consume api.

2. Concept

The concept behind this Quiz Management system is to provide efficient and optimized way to perform CRUD operations in database.I have configured Spring and Hibernate framework to facilitate data flow and data mapping between relational database and java project.

3. Major Features

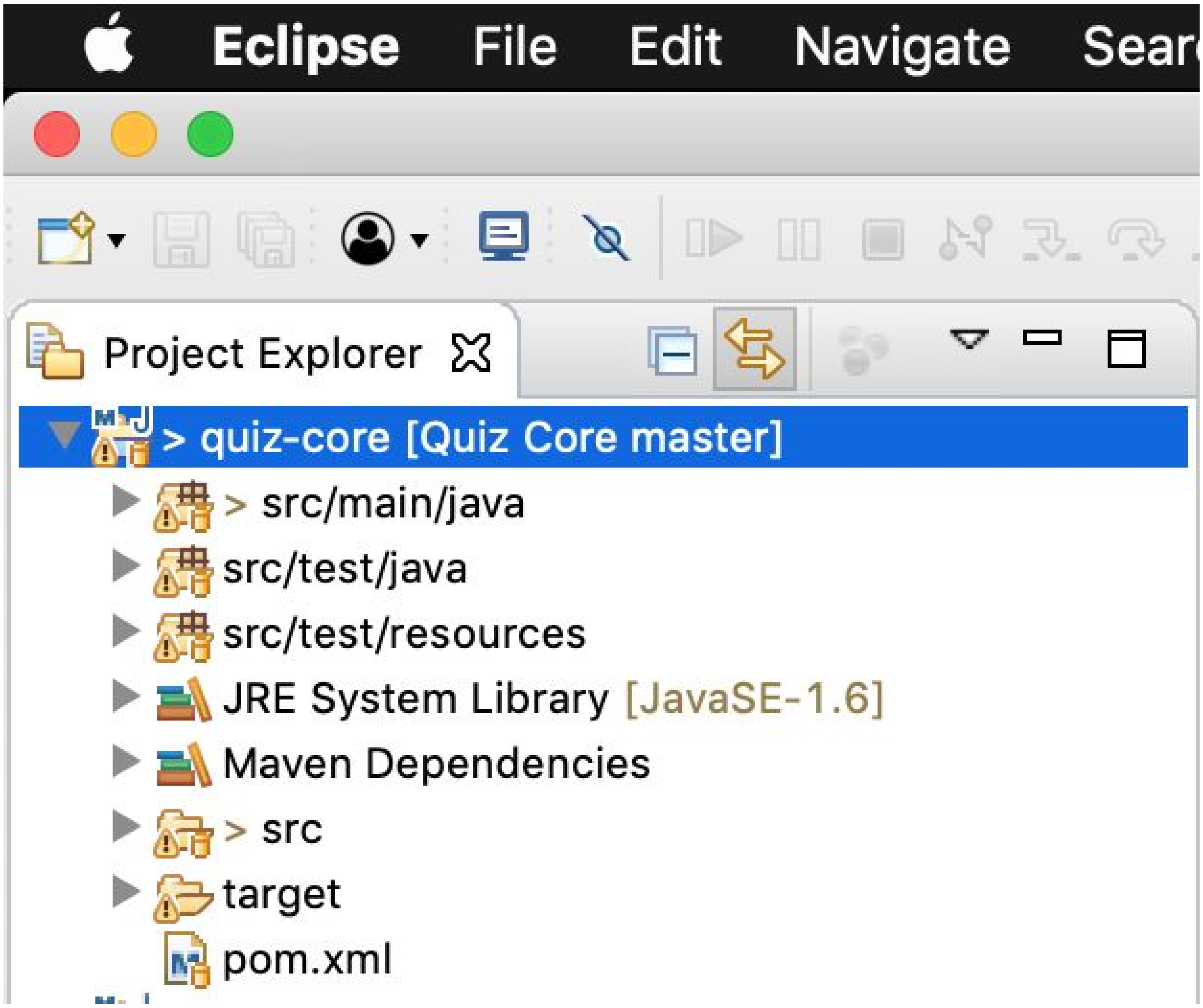
- User Should be able to create a new Quiz
- User Should be able to delete an existing Quiz by id
- Update an existing suiz by id
- User Should be able to set all quizzes
- User Should be able to search a Quiz by id
- User Should be able to create a Question with choices by given the quiz id it belongs to
- User Should be able to delete a Question with choices by id
- Get all Questions with choices
- User Should be able to update a Question with choices by id
- User Should be able to search a Question with choices by id
- User Should be able to create a Question True/False by given the quiz id it belongs to.
- User Should be able to delete a Question True/false by id
- User Should be able to get all Questions True/False
- User Should be able to update a Question True/False by id
- User Should be able to search a Question True/false by id
- User Should be able to create a choice given by id of the Question it belongs to
- User Should be able to get all choices from a question given by id of a question
- User Should be able to get choice given by id of the Question it belongs to
- User Should be able to update choice given by id of the Question it belongs to
- User Should be able to delete choice given by id

4. Project Setup and Configuration.

Setup project is not very complex , It involves couple of steps to follow.

Backend :

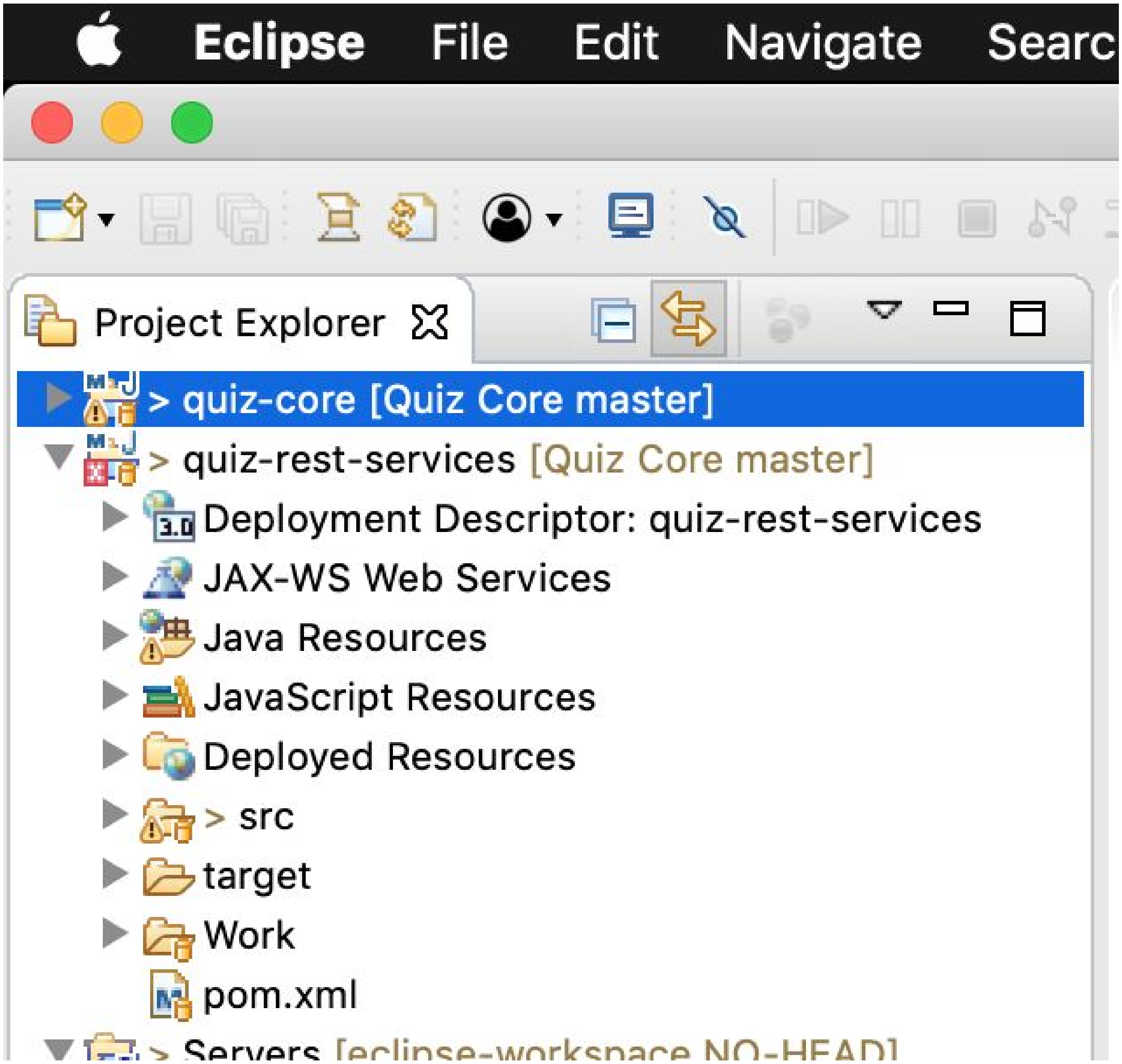
a.Create maven Project : By creating maven project you will have following project structure automatically.



```
> quiz-core [Quiz Core master]
    > src/main/java
    src/test/java
    src/test/resources
    JRE System Library [JavaSE-1.6]
    Maven Dependencies
    > src
    target
    pom.xml
```

b.Create Dynamic Project

`

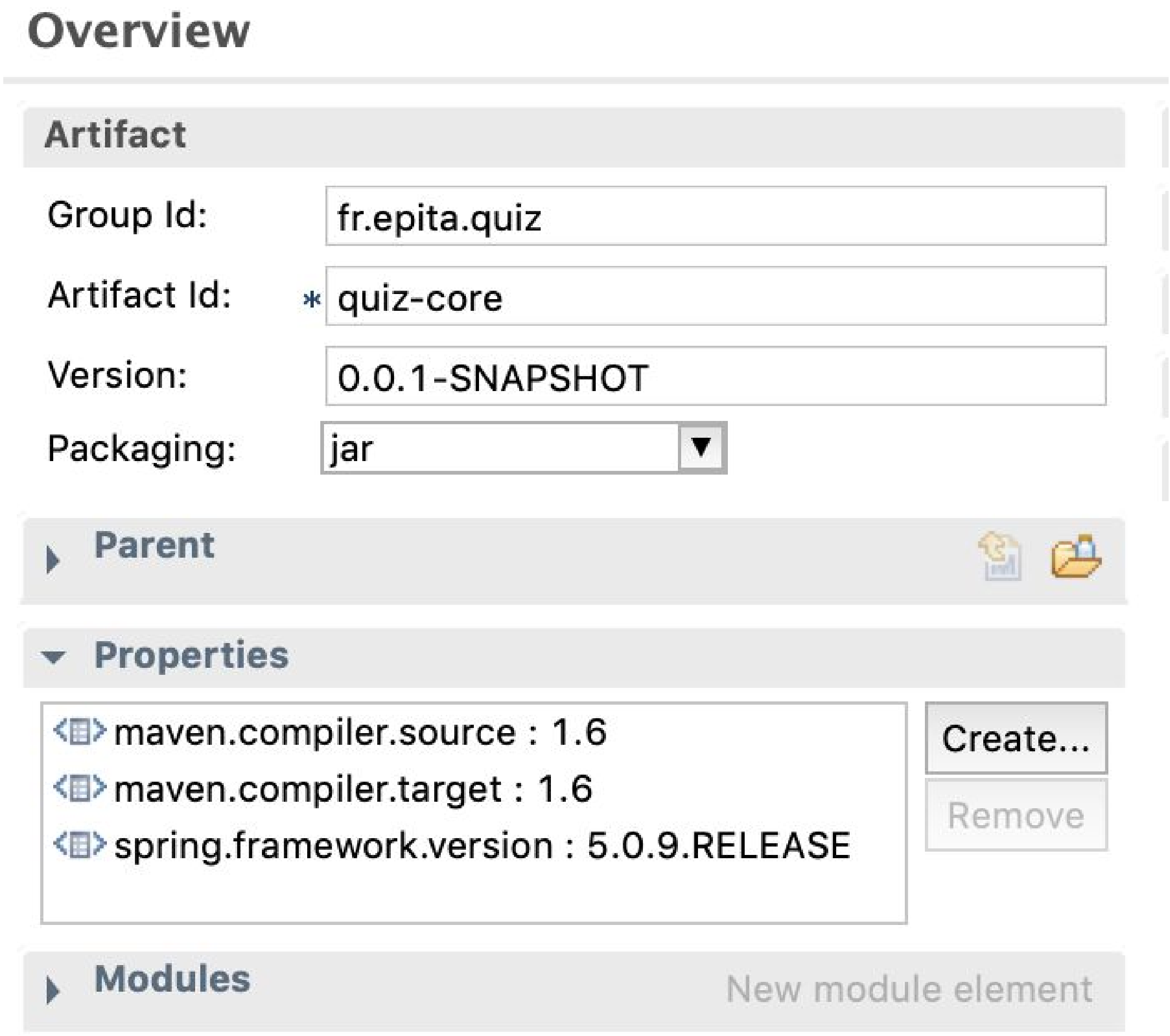

Eclipse
File
Edit
Navigate
Searc
Project Explorer
> quiz-core [Quiz Core master]
> quiz-rest-services [Quiz Core master]
Deployment Descriptor: quiz-rest-services
JAX-WS Web Services
Java Resources
JavaScript Resources
Deployed Resources
> src
target
Work
pom.xml
> Servers [eclipse-workspace NO-HEAD]

c.Configure all the maven dependencies on .pom xml file.

Group Id : Is package identifier ,we define following group id to all the modules we created for this project.

`

SNAPSHOT: Usually means that this version is still under heavy development.



Overview

Artifact

Group Id: fr.epita.quiz

Artifact Id: * quiz-core

Version: 0.0.1-SNAPSHOT

Packaging: jar ▼

Parent

Properties

- maven.compiler.source : 1.6
- maven.compiler.target : 1.6
- spring.framework.version : 5.0.9.RELEASE

Front End

1.Install Node.js : Download node.js from the following link for your suitable machine environment.

`

https://nodejs.org/en/download/

2.Configure and Install “npm”

“npm install”

3.Create react project on desired workspace.

“create-react-app myquizcore ”

4.Finally we created a project.Run the FE using this command.

“npm start client” or “npm start”

5. Feasibility Study

1.1.1. Application Feasibility :

This current application's compulsory parameters are altogether known. The User ought to have a login Name and Password and the Identity's parameters have been resolved. Likewise we will utilize a Derby Database as a backend database with the end goal to make our program's information control less demanding to comprehend and to program. The majority of the application necessities have been resolved and incorporated into the program

1.1.2. Time Feasibility :

In software development and delivery time is key factor, However this project is for learning purpose and evaluation purpose. I have dedicated time frame to work on it which is feasible.

Submission Date : 23rd November 2018

Time Frame : Roughly Around 4 weeks.

1.1.3. Cost Feasibility :

This system is my assignment task and my curricular activity and i myself is developing this system so it is feasible to me.

`

1.1.4 Technical Feasibility : Java is consistent and strong programming language. Because of OOP concept this system can be flexible, testable, maintainable and more organized. Due to already saturated platform there is no challenge to develop this system.

Spring framework

a. Spring Model View Controller (MVC):
   i. MVC helps for app development
b. Spring Core:
   i. It provides Inversion of Control (IOC) or dependency Injection
c. Spring Transaction Management:
   i. It provides transaction management for apps
d. Data Access Framework:
   i. It provides data or information management functionality.

Hibernate Framework : Hibernate is an Object-Relational Mapping (ORM) framework. It uses Java Database Connectivity (JDBC). It provides flexibility to change the database

6.Data Description

Quiz-Core Management System Operates on following data

1. A login username and password for the User to have access platform.
2. The input data for the creation of an question(id, questionLabel and valid).
3. The input data for the creation of MCQ choice.
4. The input “id” from the user that is needed for the deletion of an question and mcq choices.
5. The input data that is based for the search criteria (question string)of an question that are question Label and question id.

6. The data stored in our backend database is a table names QUESTION and MCQCHOICE

   Queston.

   Table : QUESTION

   Fields :ID, VALID, and QUESTIONLABEL

`

MCQ choices.

Table : MCQCHOICE

Fields :ID, VALID, CHOICELABEL and QUESTIONLABEL

7.Expected Results

As a developer, I expect my application to run safely and able to perform user identification creation, searching, updating and deleting question and mcq choices from H2 database. Identities.

This is a functional prototype and i don't have any potential clients and market view so i have focused only on features and technical aspects like, Code efficiency, clan code,best approaches,error free, organized and maintainable source code. Basically, System should run smoothly, CRUD Operation should be performed without any issues.

I performed test cases for every CRUD operations using JUnit Test in java.

Some of the Expected Results are ;

1. Questions and MCQ Choices should inserted successfully with 200 Ok Response code.
2. Should Be able to Search Questions by search string in question.And in response list question with mcq is expected.
3. Should be able to Search MCQ Choices by question object.
4. Should be able to delete question along with associated mcq choices.
5. Should be able to update any question and MCQ choice.

8.Algorithm Study

Quiz Management system is very simple and common concept, So we are using basic operations like basic sql queries, mathematical manipulations and some design patterns. So we don't have such a advance algorithm to study and analyze.

Some standard data manipulation algorithms are :

- Create an Entity
- Update an Entity
- Search an Entity
- Delete an Entity

9.Scope Of Application

This system is developed on certain consideration .It is a single prototype for POC as my curricular activity. So I have developed very simple user interface and it is not currently deployable to real clients. Let's say on real market.

Scope of Quiz management platform.

- Strong Architecture
- Well Code organization
- Standard Framework used Spring and Hibernate.

10.Conception

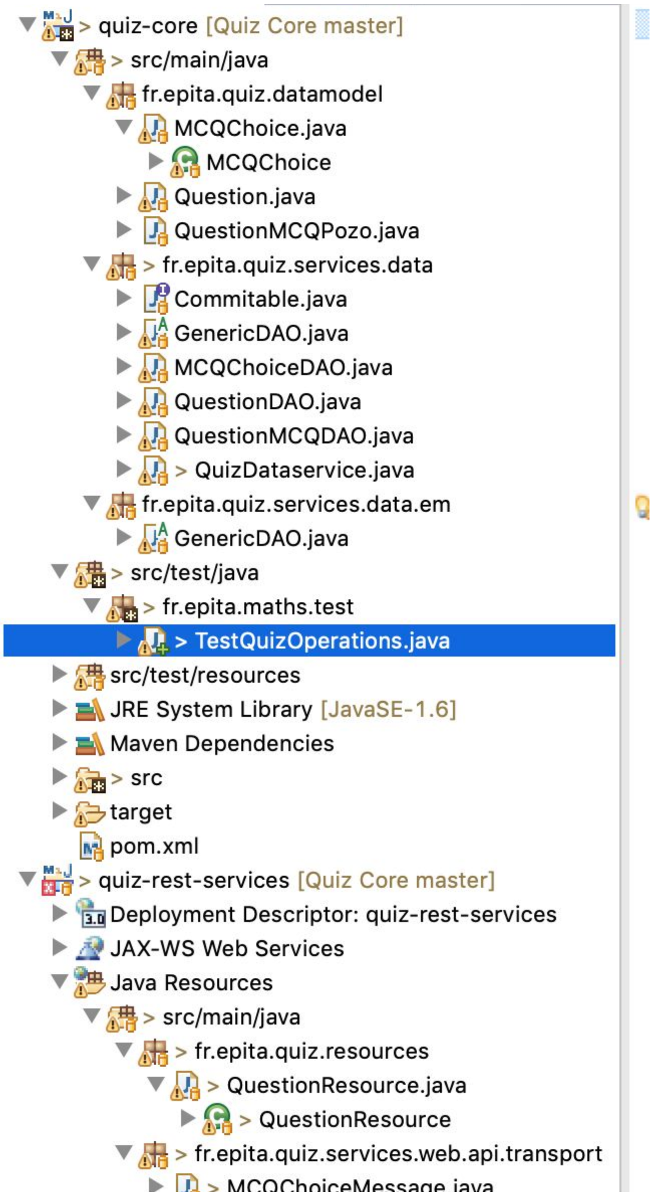
- **Data Structures**

  Data Structures. A data structure is a particular way of storing and organizing data in a computer so that it can be used efficiently. Data structures provide a means to manage large amounts of data efficiently. efficient data structures are a key to designing efficient algorithms.

  For Example : Hashmaps, QuestionDAO, MCQChoiceDAO, GenericeDAO etc.

- **Application Structure**

`

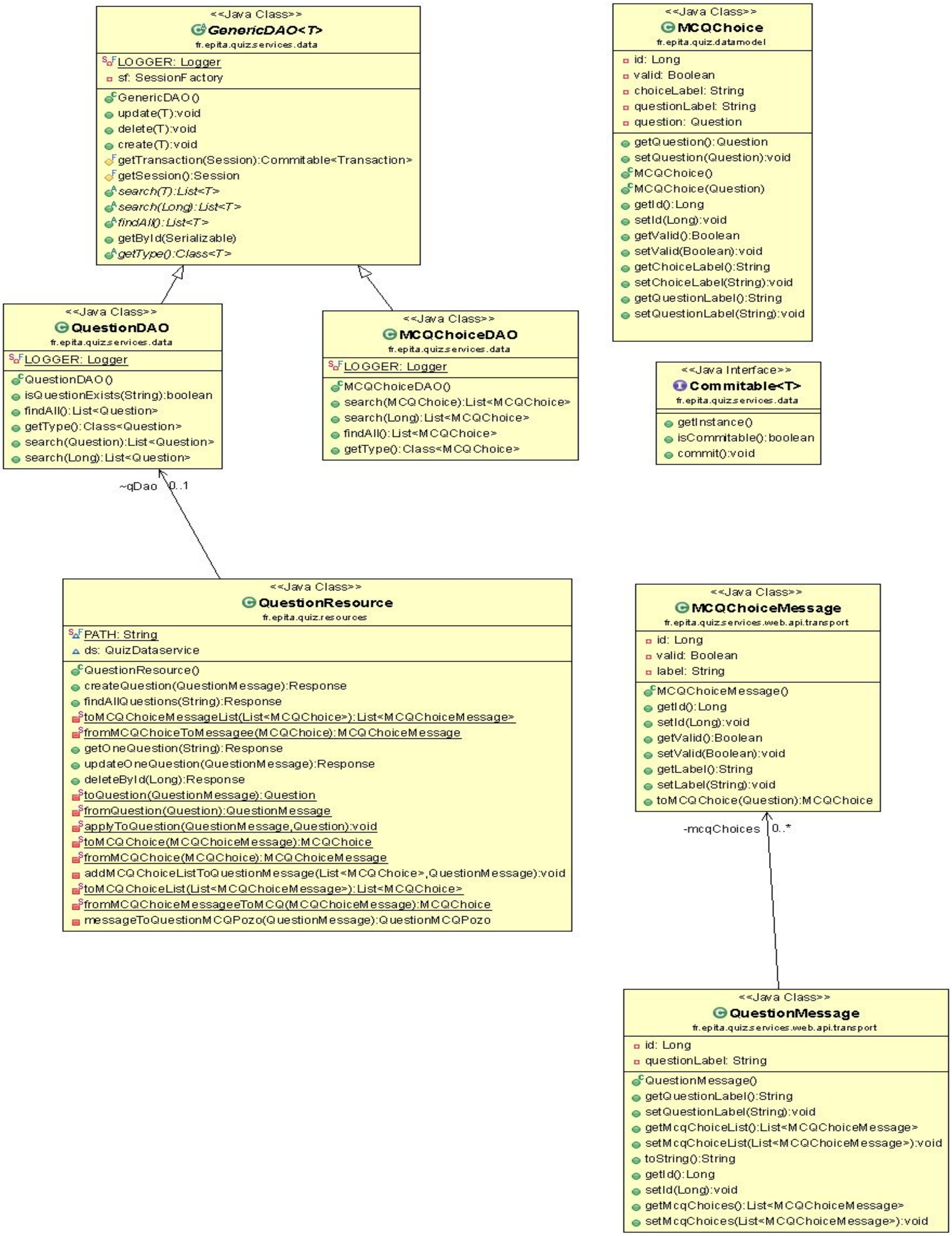


```
▼ > quiz-core [Quiz Core master]
  ▼ > src/main/java
    ▼ fr.epita.quiz.datamodel
      ▼ MCQChoice.java
        ▶ MCQChoice
      ▶ Question.java
      ▶ QuestionMCQPozo.java
    ▼ > fr.epita.quiz.services.data
      ▶ Commitable.java
      ▶ GenericDAO.java
      ▶ MCQChoiceDAO.java
      ▶ QuestionDAO.java
      ▶ QuestionMCQDAO.java
      ▶ > QuizDataservice.java
    ▼ fr.epita.quiz.services.data.em
      ▶ GenericDAO.java
  ▼ > src/test/java
    ▼ > fr.epita.maths.test
      ▶ > TestQuizOperations.java
  ▶ src/test/resources
  ▶ JRE System Library [JavaSE-1.6]
  ▶ Maven Dependencies
  ▶ > src
  ▶ target
    pom.xml
▼ > quiz-rest-services [Quiz Core master]
  ▶ Deployment Descriptor: quiz-rest-services
  ▶ JAX-WS Web Services
  ▼ Java Resources
    ▼ > src/main/java
      ▼ > fr.epita.quiz.resources
        ▼ > QuestionResource.java
          ▶ > QuestionResource
      ▼ > fr.epita.quiz.services.web.api.transport
        ▶ > MCQChoiceMessage.java
```

11.UML and Data Flow.

- Open Quiz
- Create Question with MCQ Choices.
- Update Question
- Delete Question
- List All Question With MCQ

`


<<Java Class>>
GenericDAO<T>
fr.epita.quizservices.data
LOGGER: Logger
sf: SessionFactory
GenericDAO()
update(T):void
delete(T):void
create(T):void
getTransaction(Session):Commitable<Transaction>
getSession():Session
search(T):List<T>
search(Long):List<T>
findAll():List<T>
getById(Serializable)
getType():Class<T>
<<Java Class>>
MCQChoice
fr.epita.quiz.datamodel
id: Long
valid: Boolean
choiceLabel: String
questionLabel: String
question: Question
getQuestion():Question
setQuestion(Question):void
MCQChoice()
MCQChoice(Question)
getId():Long
setId(Long):void
getValid():Boolean
setValid(Boolean):void
getChoiceLabel():String
setChoiceLabel(String):void
getQuestionLabel():String
setQuestionLabel(String):void
<<Java Class>>
QuestionDAO
fr.epita.quizservices.data
LOGGER: Logger
QuestionDAO()
isQuestionExists(String):boolean
findAll():List<Question>
getType():Class<Question>
search(Question):List<Question>
search(Long):List<Question>
<<Java Class>>
MCQChoiceDAO
fr.epita.quizservices.data
LOGGER: Logger
MCQChoiceDAO()
search(MCQChoice):List<MCQChoice>
search(Long):List<MCQChoice>
findAll():List<MCQChoice>
getType():Class<MCQChoice>
<<Java Interface>>
Commitable<T>
fr.epita.quizservices.data
getInstance()
isCommitable():boolean
commit():void
~qDao
0..1
<<Java Class>>
QuestionResource
fr.epita.quiz.resources
PATH: String
ds: QuizDataservice
QuestionResource()
createQuestion(QuestionMessage):Response
findAllQuestions(String):Response
toMCQChoiceMessageList(List<MCQChoice>):List<MCQChoiceMessage>
fromMCQChoiceToMessagee(MCQChoice):MCQChoiceMessage
getOneQuestion(String):Response
updateOneQuestion(QuestionMessage):Response
deleteById(Long):Response
toQuestion(QuestionMessage):Question
fromQuestion(Question):QuestionMessage
applyToQuestion(QuestionMessage,Question):void
toMCQChoice(MCQChoiceMessage):MCQChoice
fromMCQChoice(MCQChoice):MCQChoiceMessage
addMCQChoiceListToQuestionMessage(List<MCQChoice>,QuestionMessage):void
toMCQChoiceList(List<MCQChoiceMessage>):List<MCQChoice>
fromMCQChoiceMessageeToMCQ(MCQChoiceMessage):MCQChoice
messageToQuestionMCQPozo(QuestionMessage):QuestionMCQPozo
<<Java Class>>
MCQChoiceMessage
fr.epita.quizservices.web.api.transport
id: Long
valid: Boolean
label: String
MCQChoiceMessage()
getId():Long
setId(Long):void
getValid():Boolean
setValid(Boolean):void
getLabel():String
setLabel(String):void
toMCQChoice(Question):MCQChoice
-mcqChoices
0..*
<<Java Class>>
QuestionMessage
fr.epita.quizservices.web.api.transport
id: Long
questionLabel: String
QuestionMessage()
getQuestionLabel():String
setQuestionLabel(String):void
getMcqChoiceList():List<MCQChoiceMessage>
setMcqChoiceList(List<MCQChoiceMessage>):void
toString():String
getId():Long
setId(Long):void
getMcqChoices():List<MCQChoiceMessage>
setMcqChoices(List<MCQChoiceMessage>):void

`

12.Design Patterns

Design pattern is a general, reusable solution to a commonly occurring problem within a given context in software design.It is not a mandatory to use design patterns in every projects however based on the project requirement and design specification we can use design patterns in software design. It is not a finished design that can be transformed directly into source or machine code.

There are various design patterns suggested in software development. In my IMS project i have used only Factory Pattern , Singleton and Builder Pattern.

Factory Pattern Implementation : Factory pattern deals with object creation and behaviour of object when communicating with other objects.

Singleton Pattern : Singleton Pattern states that, There should be only one instance of any class at runtime which will be accessible by every entities required. Which means no multiple objects of same class exists in the application during the runtime. Which will help to maintain consistency , conflictless flow of application.

Builder Pattern : Builder pattern deals with the behaviour of accessing data model for example. In IMS system In identity class we have variables name,uid and email. Builder pattern in this case encourage to use getter setter in the model. So that data model class is encapsulated and the way of accessing data model is changed.

Now, If we have to change to value of name for particular email we will simply call *setName(param)* instead calling constructor every time.

`

13.Configuration Instruction

11.0.Ide and Tools

Eclipse is widely used and dynamic tool for java application development. I have used eclipse for development tool and Sonar as a code quality analysis.

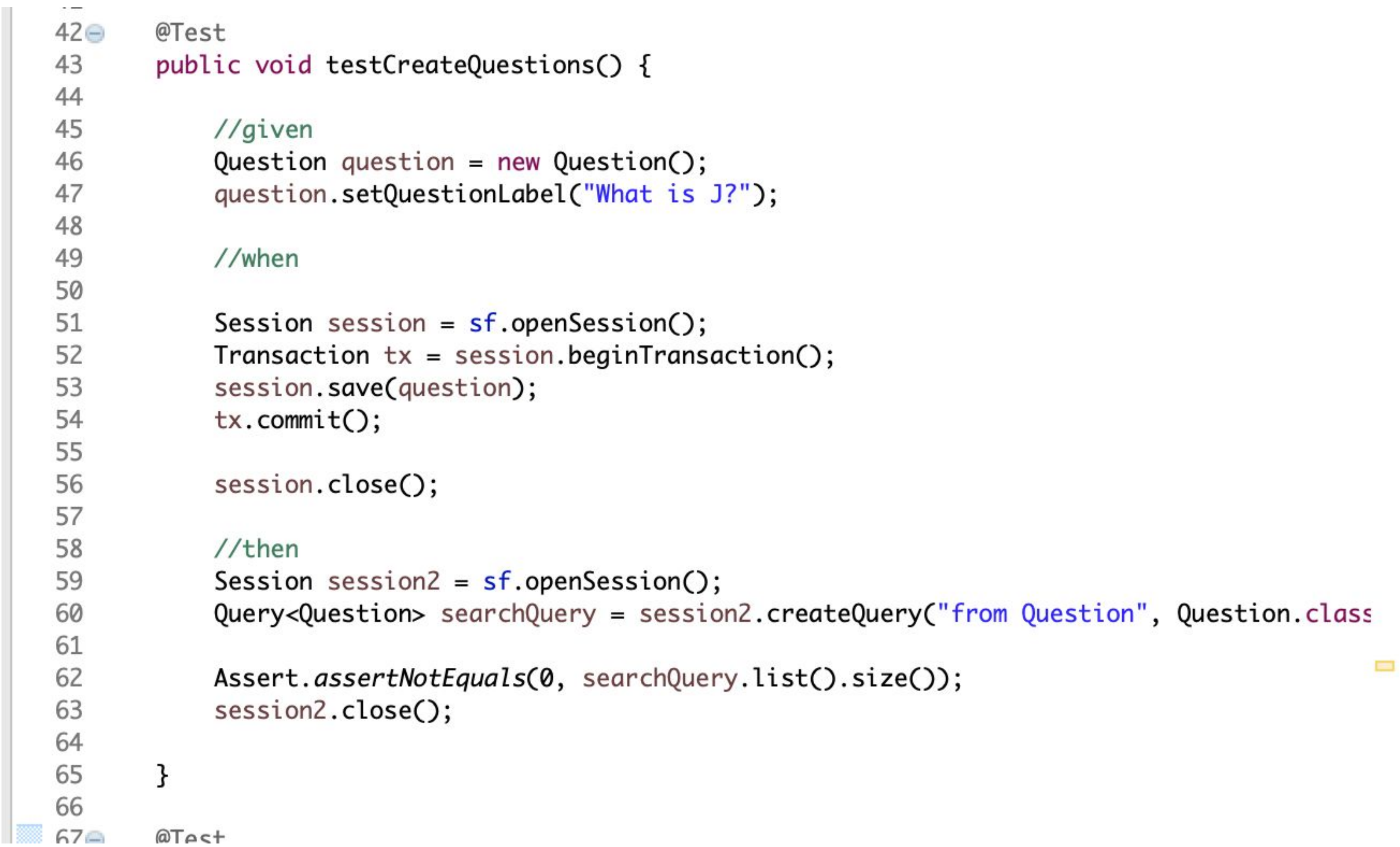
14.Testing

API testing can be one of the most challenging parts of software and QA testing because APIs can be complicated, they are often based on protocols and standards that we often do no encounter in other kinds of testing.

While developers tend to test only the basic functionality they are working on, testers are in charge of testing functionality, performance and security of APIs, discovering how all components work together from end to end.

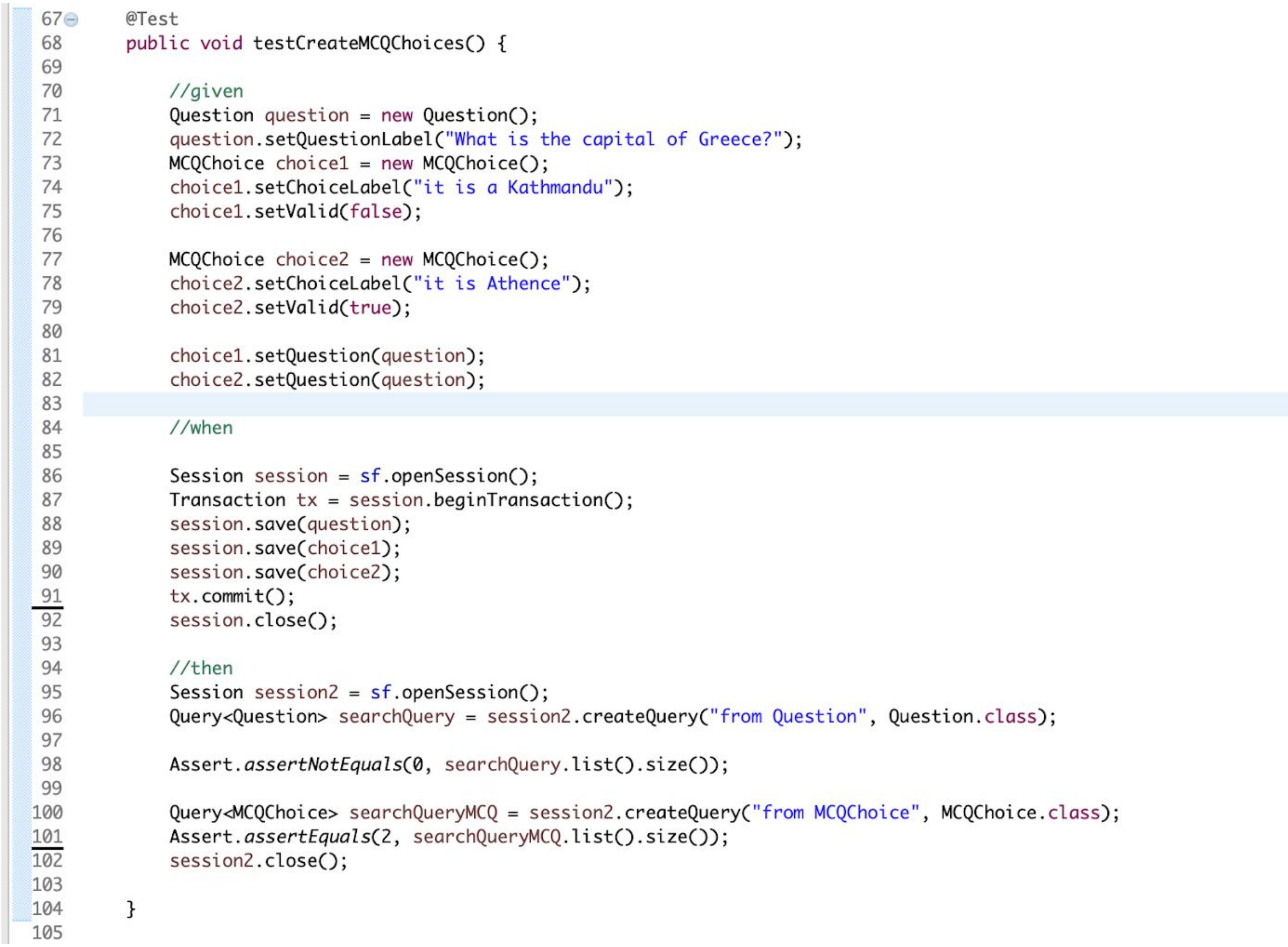
`

1. **Junit Test [Code Level]**
    a. **Create Question**



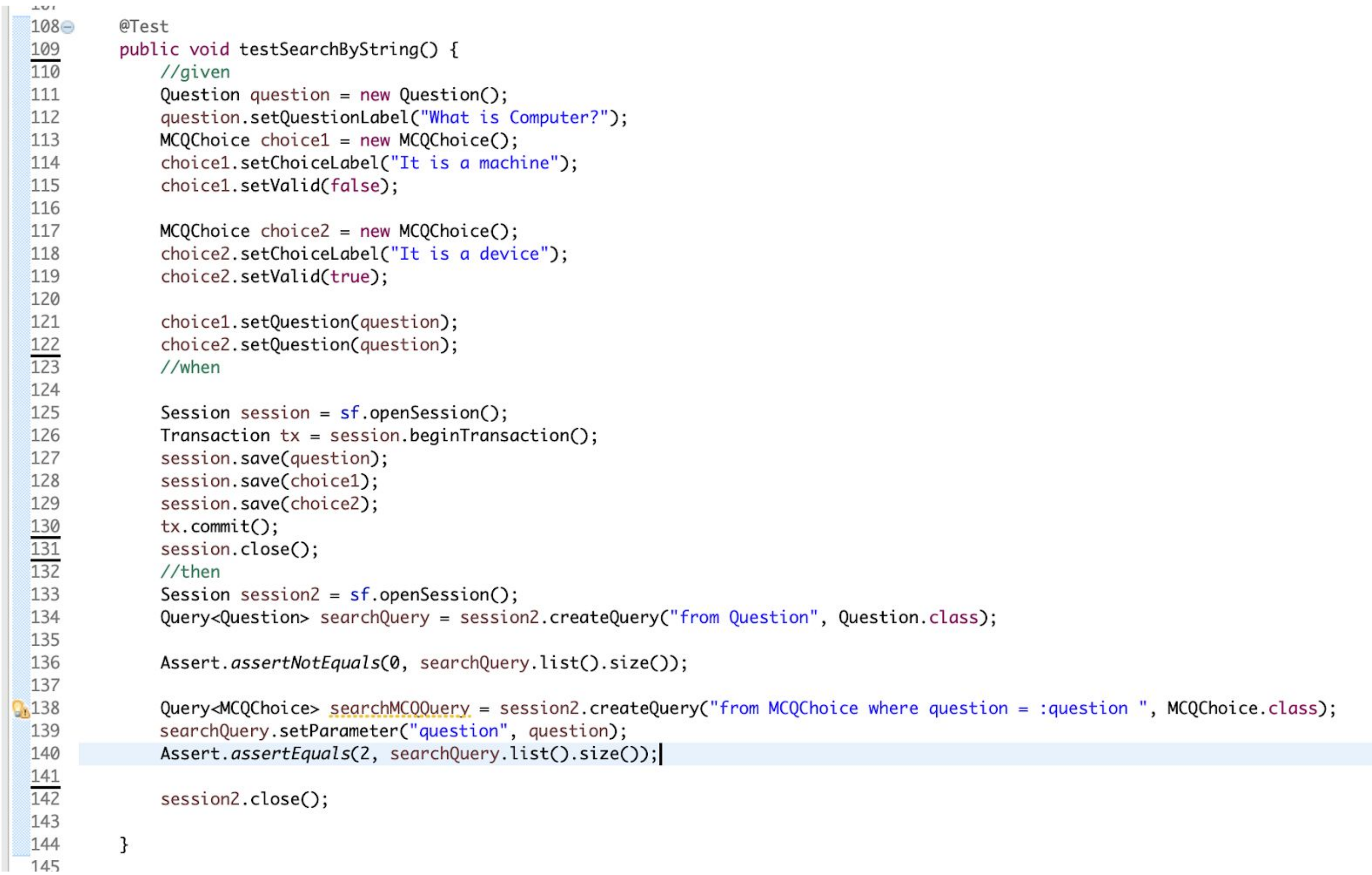
```
    @Test
    public void testCreateQuestions() {

        //given
        Question question = new Question();
        question.setQuestionLabel("What is J?");

        //when

        Session session = sf.openSession();
        Transaction tx = session.beginTransaction();
        session.save(question);
        tx.commit();

        session.close();

        //then
        Session session2 = sf.openSession();
        Query<Question> searchQuery = session2.createQuery("from Question", Question.class

        Assert.assertNotEquals(0, searchQuery.list().size());
        session2.close();

    }

    @Test
```

b. Create MCQ Choices



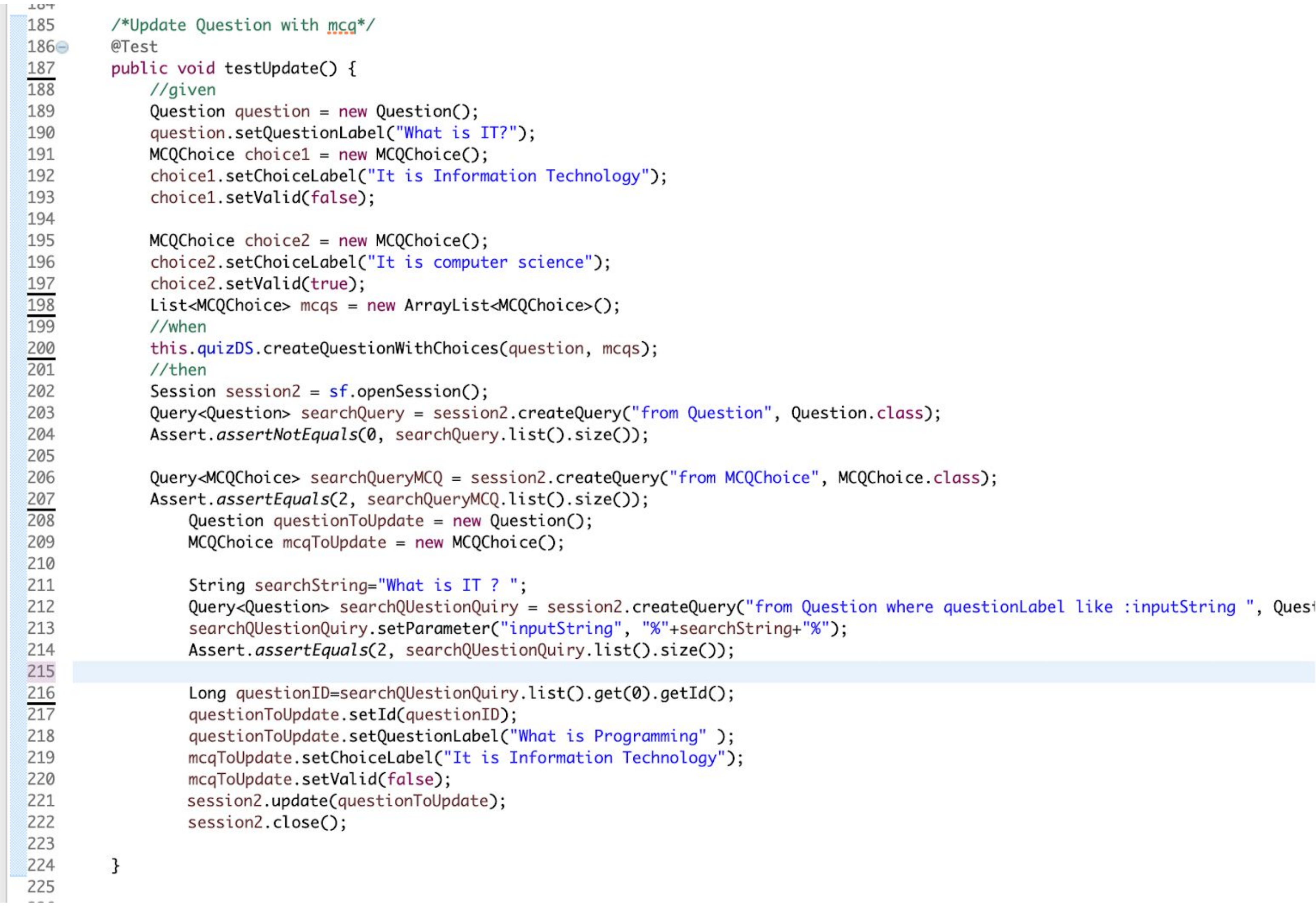
```java
    @Test
    public void testCreateMCQChoices() {

        //given
        Question question = new Question();
        question.setQuestionLabel("What is the capital of Greece?");
        MCQChoice choice1 = new MCQChoice();
        choice1.setChoiceLabel("it is a Kathmandu");
        choice1.setValid(false);

        MCQChoice choice2 = new MCQChoice();
        choice2.setChoiceLabel("it is Athence");
        choice2.setValid(true);

        choice1.setQuestion(question);
        choice2.setQuestion(question);

        //when

        Session session = sf.openSession();
        Transaction tx = session.beginTransaction();
        session.save(question);
        session.save(choice1);
        session.save(choice2);
        tx.commit();
        session.close();

        //then
        Session session2 = sf.openSession();
        Query<Question> searchQuery = session2.createQuery("from Question", Question.class);

        Assert.assertNotEquals(0, searchQuery.list().size());

        Query<MCQChoice> searchQueryMCQ = session2.createQuery("from MCQChoice", MCQChoice.class);
        Assert.assertEquals(2, searchQueryMCQ.list().size());
        session2.close();

    }
```

c. Get Questions



```
    @Test
    public void testSearchByString() {
        //given
        Question question = new Question();
        question.setQuestionLabel("What is Computer?");
        MCQChoice choice1 = new MCQChoice();
        choice1.setChoiceLabel("It is a machine");
        choice1.setValid(false);

        MCQChoice choice2 = new MCQChoice();
        choice2.setChoiceLabel("It is a device");
        choice2.setValid(true);

        choice1.setQuestion(question);
        choice2.setQuestion(question);
        //when

        Session session = sf.openSession();
        Transaction tx = session.beginTransaction();
        session.save(question);
        session.save(choice1);
        session.save(choice2);
        tx.commit();
        session.close();
        //then
        Session session2 = sf.openSession();
        Query<Question> searchQuery = session2.createQuery("from Question", Question.class);

        Assert.assertNotEquals(0, searchQuery.list().size());

        Query<MCQChoice> searchMCQQuery = session2.createQuery("from MCQChoice where question = :question ", MCQChoice.class);
        searchQuery.setParameter("question", question);
        Assert.assertEquals(2, searchQuery.list().size());

        session2.close();

    }
```

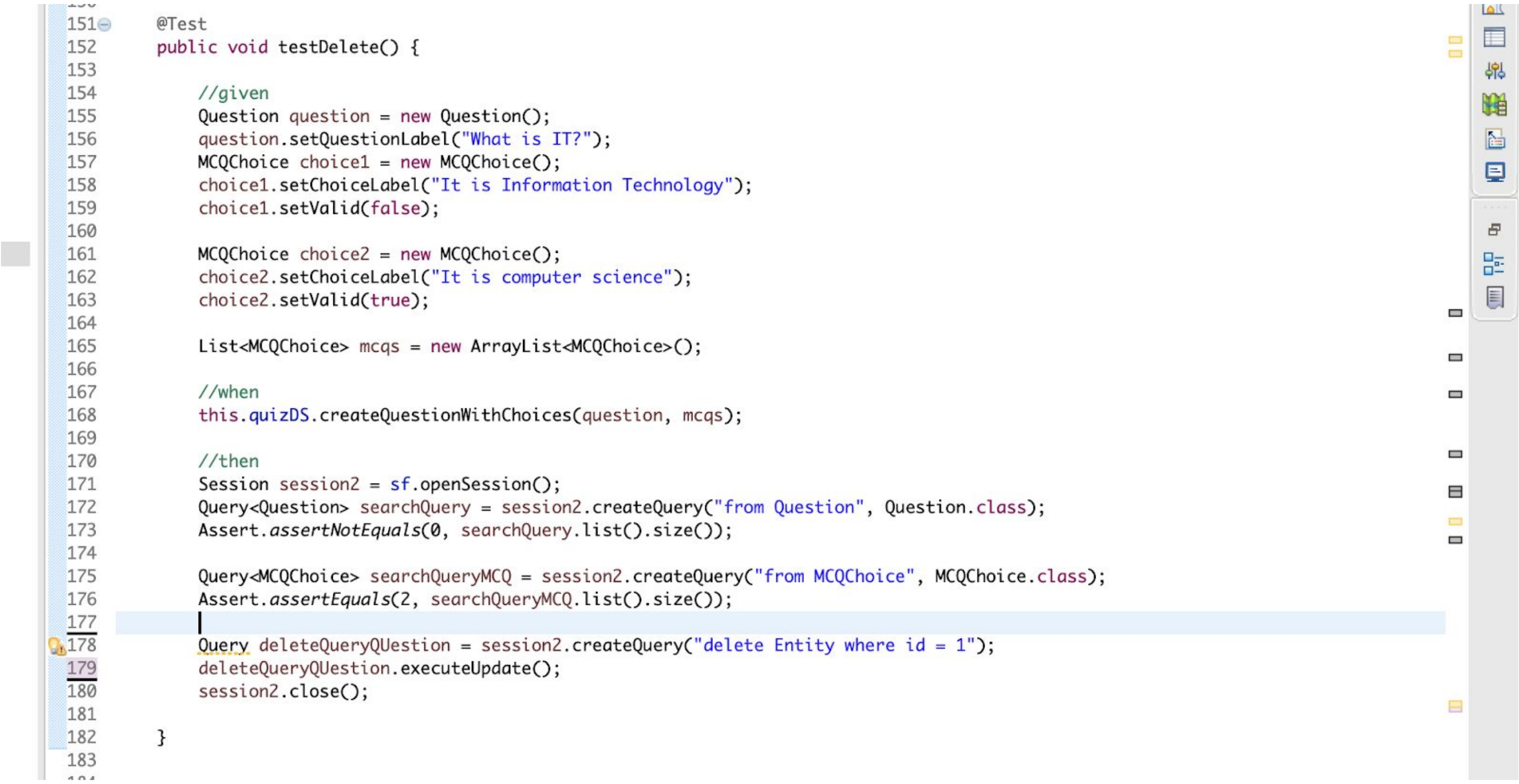
d. Update Questions and MCQChoices.



```
    /*Update Question with mcq*/
    @Test
    public void testUpdate() {
        //given
        Question question = new Question();
        question.setQuestionLabel("What is IT?");
        MCQChoice choice1 = new MCQChoice();
        choice1.setChoiceLabel("It is Information Technology");
        choice1.setValid(false);

        MCQChoice choice2 = new MCQChoice();
        choice2.setChoiceLabel("It is computer science");
        choice2.setValid(true);
        List<MCQChoice> mcqs = new ArrayList<MCQChoice>();
        //when
        this.quizDS.createQuestionWithChoices(question, mcqs);
        //then
        Session session2 = sf.openSession();
        Query<Question> searchQuery = session2.createQuery("from Question", Question.class);
        Assert.assertNotEquals(0, searchQuery.list().size());

        Query<MCQChoice> searchQueryMCQ = session2.createQuery("from MCQChoice", MCQChoice.class);
        Assert.assertEquals(2, searchQueryMCQ.list().size());
            Question questionToUpdate = new Question();
            MCQChoice mcqToUpdate = new MCQChoice();

            String searchString="What is IT ? ";
            Query<Question> searchQUestionQuiry = session2.createQuery("from Question where questionLabel like :inputString ", Quest
            searchQUestionQuiry.setParameter("inputString", "%"+searchString+"%");
            Assert.assertEquals(2, searchQUestionQuiry.list().size());

            Long questionID=searchQUestionQuiry.list().get(0).getId();
            questionToUpdate.setId(questionID);
            questionToUpdate.setQuestionLabel("What is Programming" );
            mcqToUpdate.setChoiceLabel("It is Information Technology");
            mcqToUpdate.setValid(false);
            session2.update(questionToUpdate);
            session2.close();

    }
```

`

e. Delete Question with MCQ Choice.

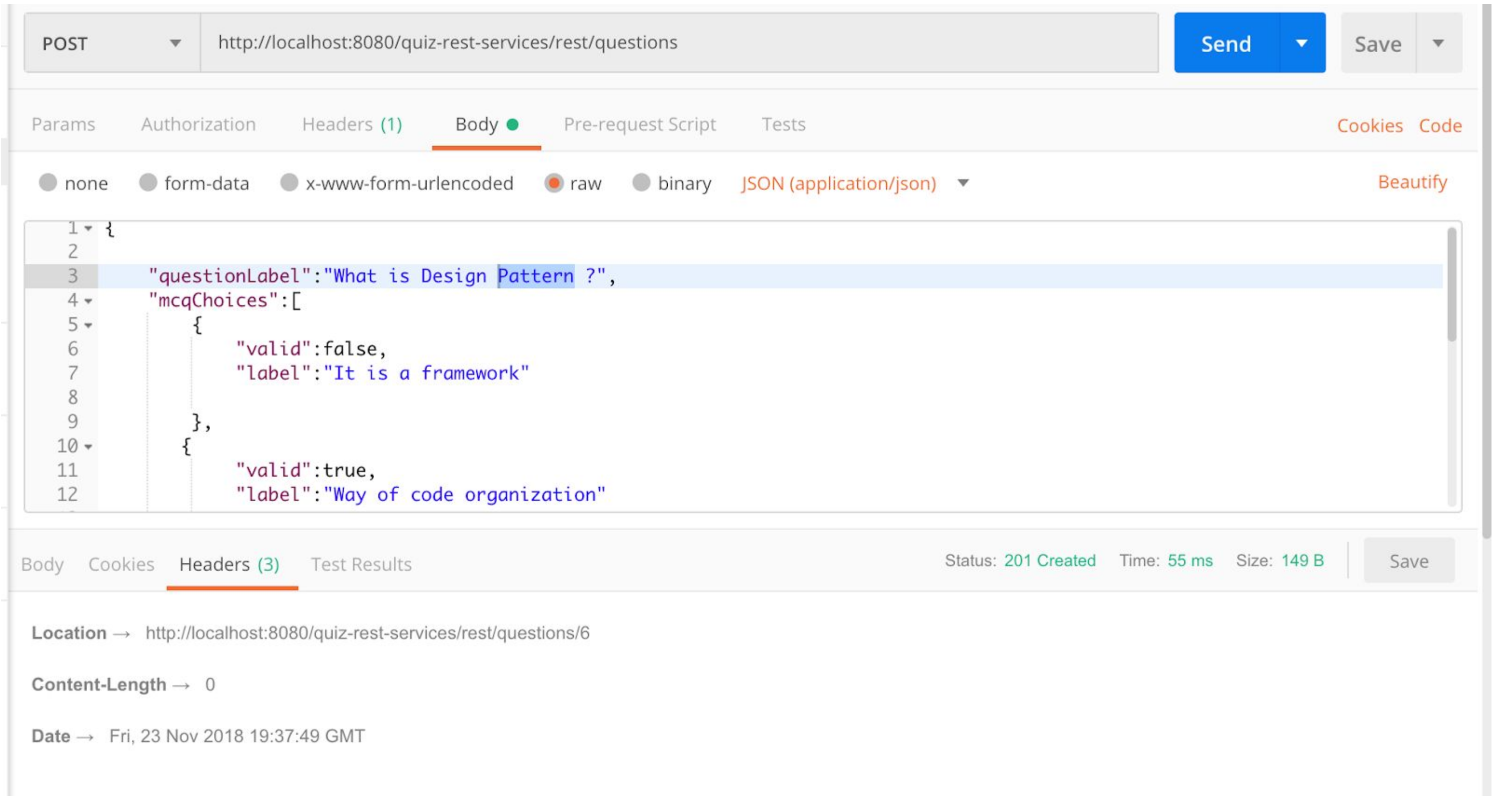


```
    @Test
    public void testDelete() {

        //given
        Question question = new Question();
        question.setQuestionLabel("What is IT?");
        MCQChoice choice1 = new MCQChoice();
        choice1.setChoiceLabel("It is Information Technology");
        choice1.setValid(false);

        MCQChoice choice2 = new MCQChoice();
        choice2.setChoiceLabel("It is computer science");
        choice2.setValid(true);

        List<MCQChoice> mcqs = new ArrayList<MCQChoice>();

        //when
        this.quizDS.createQuestionWithChoices(question, mcqs);

        //then
        Session session2 = sf.openSession();
        Query<Question> searchQuery = session2.createQuery("from Question", Question.class);
        Assert.assertNotEquals(0, searchQuery.list().size());

        Query<MCQChoice> searchQueryMCQ = session2.createQuery("from MCQChoice", MCQChoice.class);
        Assert.assertEquals(2, searchQueryMCQ.list().size());

        Query deleteQueryQUestion = session2.createQuery("delete Entity where id = 1");
        deleteQueryQUestion.executeUpdate();
        session2.close();

    }
```

`

2. Restful services [Service Layer]

a. Create Question



POST http://localhost:8080/quiz-rest-services/rest/questions

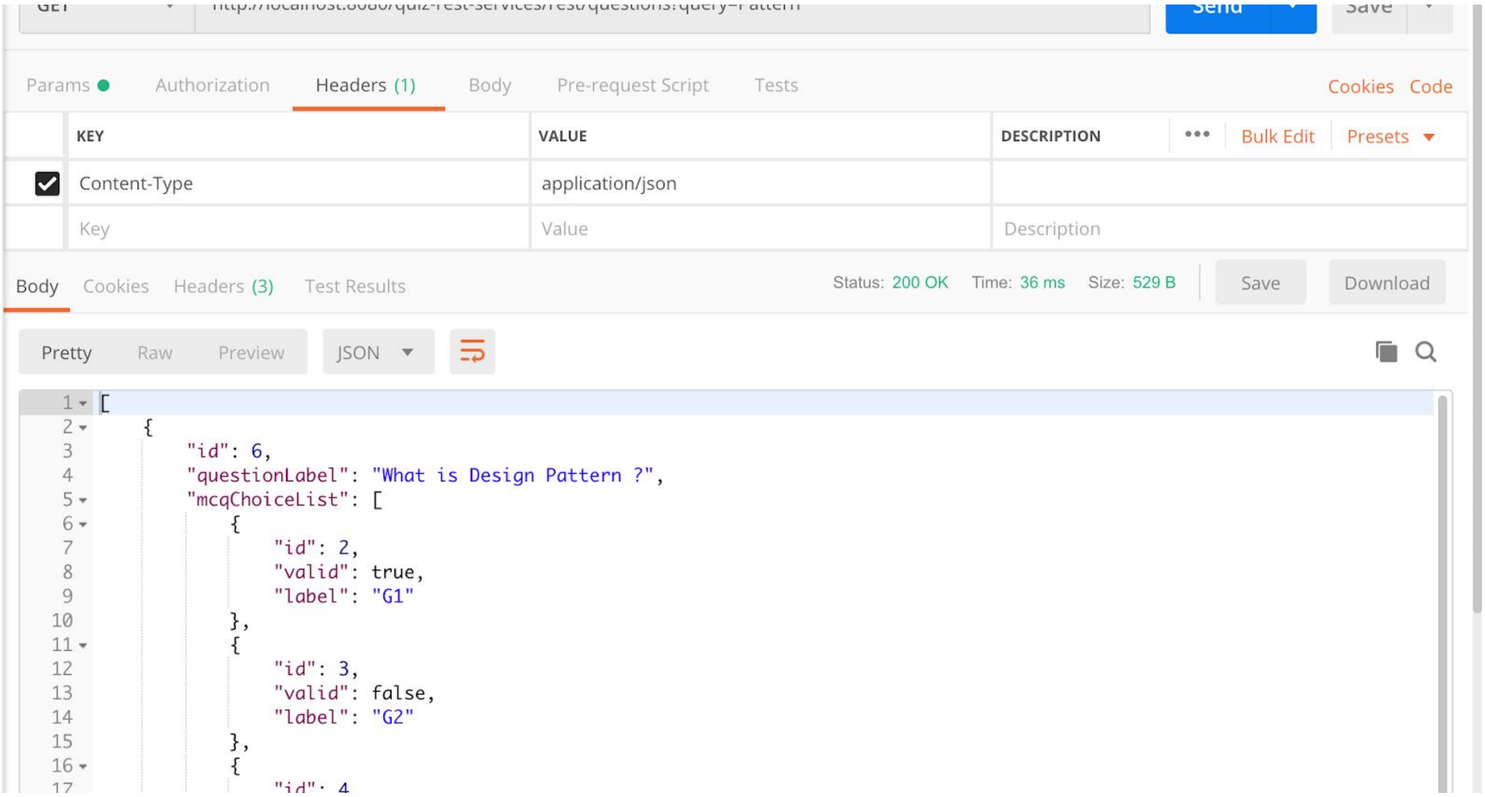
```
{

    "questionLabel":"What is Design Pattern ?",
    "mcqChoices":[
        {
            "valid":false,
            "label":"It is a framework"

        },
       {
            "valid":true,
            "label":"Way of code organization"
```

Status: 201 Created   Time: 55 ms   Size: 149 B

```
Location → http://localhost:8080/quiz-rest-services/rest/questions/6

Content-Length → 0

Date → Fri, 23 Nov 2018 19:37:49 GMT
```

`

b. Search Question



GET http://localhost:8080/quiz-rest-services/rest/questions?query=Pattern

	KEY	VALUE	DESCRIPTION
☑	Content-Type	application/json	
	Key	Value	Description

Status: 200 OK Time: 36 ms Size: 529 B

```
[
    {
        "id": 6,
        "questionLabel": "What is Design Pattern ?",
        "mcqChoiceList": [
            {
                "id": 2,
                "valid": true,
                "label": "G1"
            },
            {
                "id": 3,
                "valid": false,
                "label": "G2"
            },
            {
                "id": 4
```

15.Bibliography

- StackOverflow.
- TutorialsPoint
- Google
- Medium.com

||
||
||
||
||

-------------------------------- Thank You --

------------------------------------ END --

`