**КАЗАХСКИЙ НАЦИОНАЛЬНЫЙ ПЕДАГОГИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ АБАЯ**

**«Тестовая система»**

(проект)

**Выполнила:** Рахметалла У.Т.

**Специальность:** 6В01509 –

Информатика и робототехника

**Руководитель:** Киселёва Е. А.

**Алматы-2022**

**Оглавление**

|  |  |
| --- | --- |
|  |  |
| Паспорт проектной работы | 3 |
| Введение | 4 |
| Глава 1. Рассмотрение задачи и способы реализации. | 4 |
| 1.1 Требуемые функции проекта | 4 |
| 1.2 Способ реализации в Scene Builder. | 5 |
| Глава 2. Описание кода | 5 |
| 2.1 Описание классов  2.2 Весь код | 6  9 |
| Заключение | 9 |

**Паспорт проектной работы**

|  |  |
| --- | --- |
| Название проекта | Тестовая система |
| Участник проекта | Рахметалла Уазифа |
| Цель проекта | Написать программу, позволяющая создавать наборы тестовых заданий. |
| Задачи проекта | 1. Создать кнопку Add, для добавления новых вопросов. 2. Создать кнопку Save, в которую сохраняются вопросы и ответы. 3. Создать кнопку Edit, которая позволяет редактировать вопросы и ответы. |

**Введение**

**Актуальность проекта.** Для проверки знаний часто используют тесты. Разработать приложение для добавления, редактирования и сохранения их. Платформа будет служить как система, где хранятся тесты.

**Цель проекта.** Написать проект, где будет платформа, в которую возможно добавление новых вопросов с вариантами ответов, а также их сохранение и редактирование.

В соответствие с целью были рассмотрены такие **задачи**:

1. Создания метода для добавления вопроса
2. Создания метода для сохранения и его добавление в data.json
3. Создания метода для редактирования

**Глава 1. Рассмотрение задачи и способы реализации.**

* 1. **Требуемые функции проекта.**

Сама панель, в которой будет список. В сам список будет сохранять вопросы и ответы теста. Список нужен для того, чтобы записывать несколько вопросов и вариантов ответов. Добавления функциональных кнопок, которые будут выполнять такие функции как добавление, сохранение и редактирование. Также добавление текстовых полей для записи вопроса и вариантов ответа.

* 1. **Способ реализации в Scene Builder.**

Общая панель сделана из BorderPane в которой находится ListView и VBox.

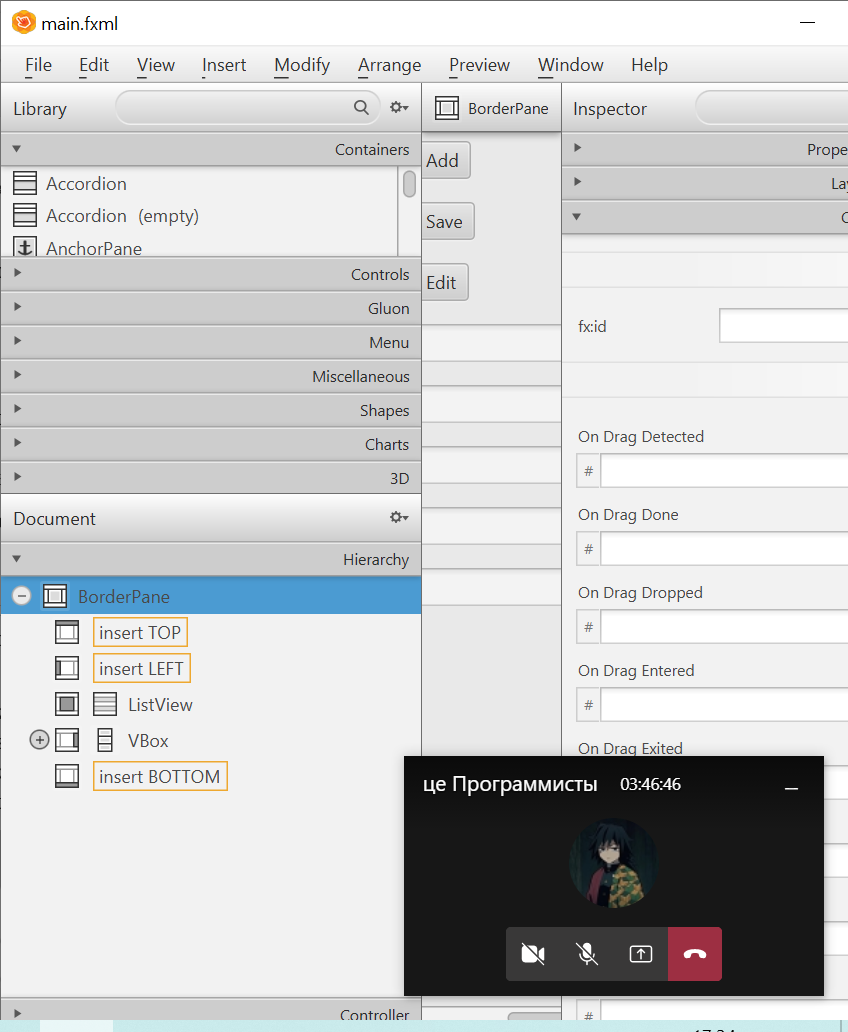


Рис.1 - Начальная иерархия проекта.

Внутри VBox три кнопки и четыре текстовых поля.

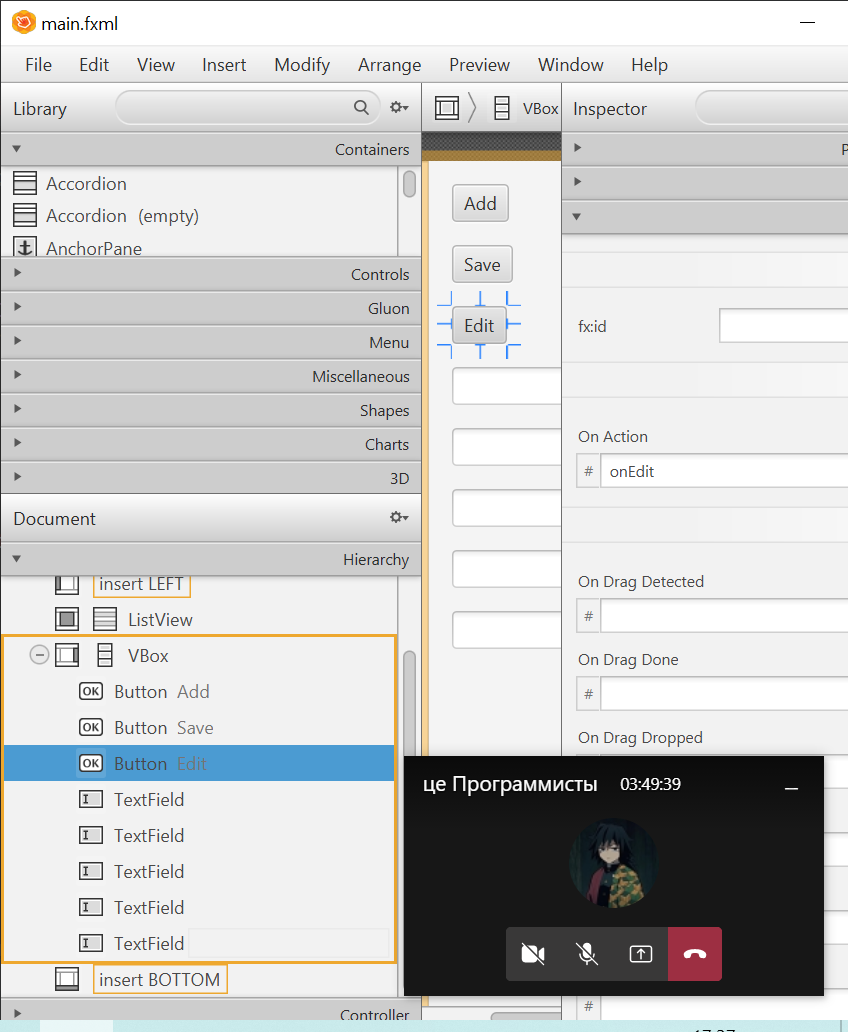


Рис.2 - Иерархия внутри VBox.

Таким образом мы имеем окно со списком, тремя функциональными кнопками и пятью текстовыми полями.

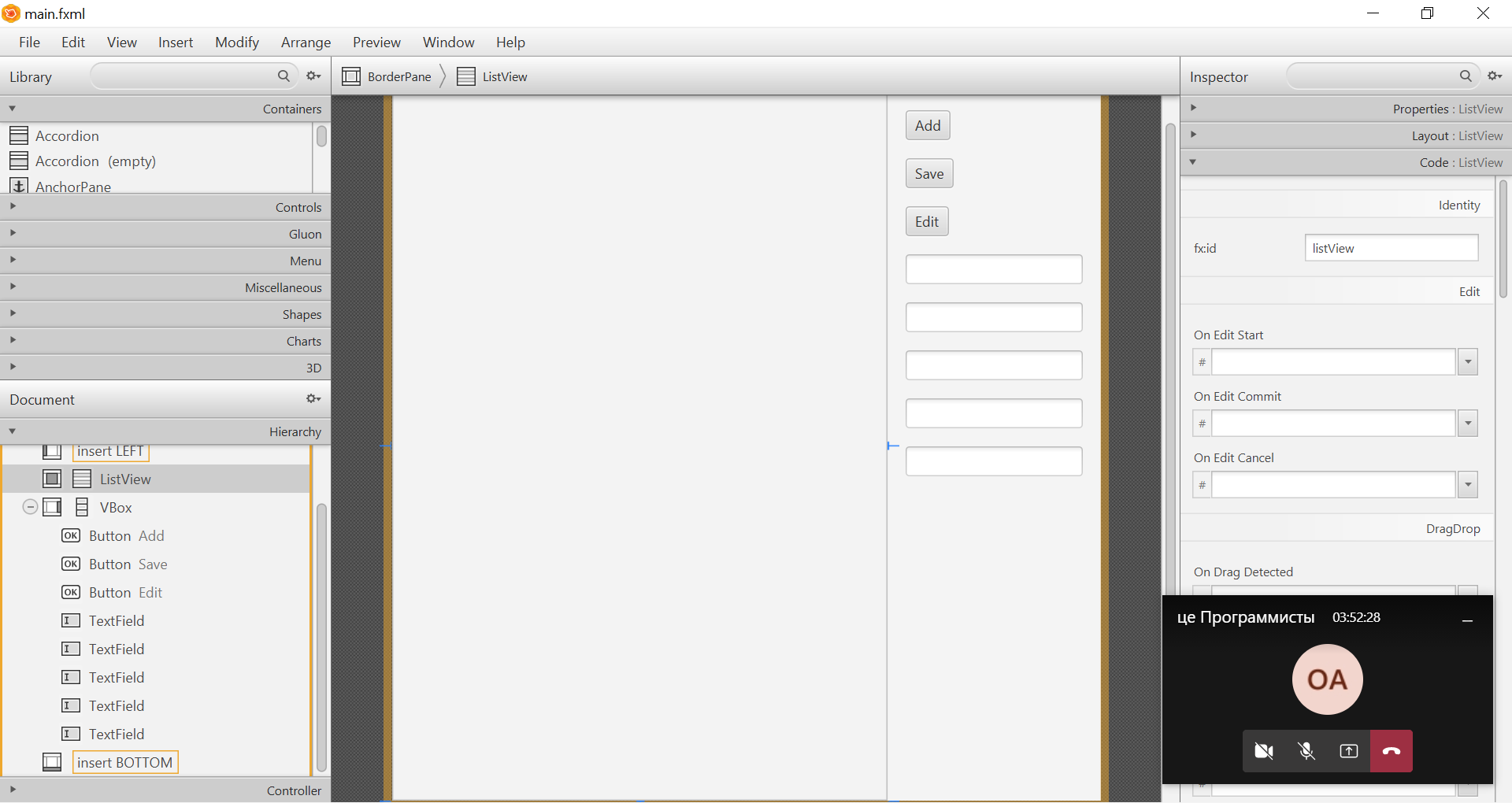


Рис.3 - Конечный вид окна.

**Глава 2. Описание кода.**

**2.1 Описание классов.**

В проекте QuizApplication имеется пакет com.example.quizapplication в котором пакет Model и два класс Controller и QuizApp.

Изображение выглядит как текст, монитор, снимок экрана, черный

Автоматически созданное описание

Рис.4 Классы проекта.

Пакет Model имеет два класса такие как QuestionModel и Quiz. Класс QuestionModel содержит в себе поля такие как text(Строка куда записывается вопрос) и answerA, answerB, answerC, answerD(Строки куда записываются варианты ответов).

public StringProperty text = new SimpleStringProperty(); // StringProperty - это абстрактный базовый класс для наблюдаемых свойств строки

public StringProperty answerA = new SimpleStringProperty(); // SimpleStringProperty - это конкретная реализация.

public StringProperty answerB = new SimpleStringProperty();

public StringProperty answerC = new SimpleStringProperty();

public StringProperty answerD = new SimpleStringProperty();

Также класс содержит в себе класс Question и внутри него реализованны конструкторы этих полей. Сам класс QuestionModel имеет свой конструктор, который устанавливает варианты ответов.

public QuestionModel toModel() {

var model = new QuestionModel();

model.text.set(text);

model.answerA.set(answerA);

model.answerB.set(answerB);

model.answerC.set(answerC);

model.answerD.set(answerD);

return model;

}

Класс Quiz может содержать в себе объекты только класса QuestionModel

public class Quiz {

private List<QuestionModel.Question> questions;

public Quiz() {

super();

}

public Quiz(List<QuestionModel.Question> questions) {

this.questions = questions;

}

public List<QuestionModel.Question> getQuestions() {

return questions;

}

public void setQuestions(List<QuestionModel.Question> questions) {

this.questions = questions;

}

}

Класс Controller содержит все три метода Add, Save и Edit. Метод Add создает экземпляр от QuestionModel и устанавливает его значение в варианты ответов (answerA, answerB, answerC, answerD), которые в свою очередь принимают значение от текстовых полей. И в конечном итоге мы добавляем эти предметы в наш список.

public void onAdd() {

QuestionModel question = new QuestionModel();

question.text.setValue(fieldName.getText());

question.answerA.setValue(fieldA.getText());

question.answerB.setValue(fieldB.getText());

question.answerC.setValue(fieldC.getText());

question.answerD.setValue(fieldD.getText());

listView.getItems().add(question);

}

Метод Save получает список и сохраняет его в файле "data.json"

public void onSave() throws Exception {

var list = listView.getItems()

.stream()

.map(QuestionModel::toData)

.collect(Collectors.toList());

var writer = new ObjectMapper();

writer.enable(SerializationFeature.INDENT\_OUTPUT);

writer.writeValue(new File("data.json"), new Quiz(list));

}

Также в классе Controller создаем поле предыдущий выбранный вопрос (prevSelectedQuestion), и оно используется в методе Edit. В Edit берется экземпляр класса QuestionModel, который содержит в себе список с выбранным предметом. Содержит условие если вопрос равен нулю, то возвращает ничего. Также условие если вопрос не равен нулю, то замена предыдущего вопроса.

public void onEdit() {

QuestionModel question = listView.getSelectionModel().getSelectedItem();

if (question == null)

return;

if (prevSelectedQuestion != null) {

prevSelectedQuestion.text.unbind();

prevSelectedQuestion.answerA.unbind();

prevSelectedQuestion.answerB.unbind();

prevSelectedQuestion.answerC.unbind();

prevSelectedQuestion.answerD.unbind();

}

fieldName.setText(question.text.getValue());

fieldA.setText(question.answerA.getValue());

fieldB.setText(question.answerB.getValue());

fieldC.setText(question.answerC.getValue());

fieldD.setText(question.answerD.getValue());

question.text.bind(fieldName.textProperty());

question.answerA.bind(fieldA.textProperty());

question.answerB.bind(fieldB.textProperty());

question.answerC.bind(fieldC.textProperty());

question.answerD.bind(fieldD.textProperty());

prevSelectedQuestion = question;

}

**2.2 Код**

**Controller**

package com.example.quizapplication;  
  
import com.example.quizapplication.Model.QuestionModel;  
import com.example.quizapplication.Model.Quiz;  
import com.fasterxml.jackson.databind.ObjectMapper;  
import com.fasterxml.jackson.databind.SerializationFeature;  
import javafx.collections.FXCollections;  
import javafx.fxml.FXML;  
import javafx.scene.control.ListView;  
import javafx.scene.control.TextField;  
  
import java.io.File;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.stream.Collectors;  
  
  
public class Controller {  
  
 @FXML  
 private ListView<QuestionModel> listView;  
  
 @FXML  
 private TextField fieldName;  
 @FXML  
 private TextField fieldA;  
 @FXML  
 private TextField fieldB;  
 @FXML  
 private TextField fieldC;  
 @FXML  
 private TextField fieldD;  
  
 public void initialize() throws Exception {  
 File file = new File("data.json");  
  
 if (!file.exists())  
 return;  
  
 var reader = new ObjectMapper();  
 Quiz quiz = reader.readValue(file, Quiz.class);  
  
 List<QuestionModel> modelList = quiz.getQuestions().stream()  
 .map(QuestionModel.Question::toModel)  
 .collect(Collectors.*toList*());  
  
 listView.setItems(  
 FXCollections.*observableArrayList*(modelList)  
 );  
 }  
  
  
  
 public void onAdd() {  
 QuestionModel question = new QuestionModel();  
 question.text.setValue(fieldName.getText());  
 question.answerA.setValue(fieldA.getText());  
 question.answerB.setValue(fieldB.getText());  
 question.answerC.setValue(fieldC.getText());  
 question.answerD.setValue(fieldD.getText());  
  
 listView.getItems().add(question);  
 }  
  
 public void onSave() throws Exception {  
 var list = listView.getItems()  
 .stream()  
 .map(QuestionModel::toData)  
 .collect(Collectors.*toList*());  
  
 var writer = new ObjectMapper();  
 writer.enable(SerializationFeature.*INDENT\_OUTPUT*);  
 writer.writeValue(new File("data.json"), new Quiz(list));  
 }  
  
 private QuestionModel prevSelectedQuestion;  
  
 public void onEdit() {  
 QuestionModel question = listView.getSelectionModel().getSelectedItem();  
 if (question == null)  
 return;  
  
 if (prevSelectedQuestion != null) {  
 prevSelectedQuestion.text.unbind();  
 prevSelectedQuestion.answerA.unbind();  
 prevSelectedQuestion.answerB.unbind();  
 prevSelectedQuestion.answerC.unbind();  
 prevSelectedQuestion.answerD.unbind();  
 }  
  
 fieldName.setText(question.text.getValue());  
 fieldA.setText(question.answerA.getValue());  
 fieldB.setText(question.answerB.getValue());  
 fieldC.setText(question.answerC.getValue());  
 fieldD.setText(question.answerD.getValue());  
  
 question.text.bind(fieldName.textProperty());  
 question.answerA.bind(fieldA.textProperty());  
 question.answerB.bind(fieldB.textProperty());  
 question.answerC.bind(fieldC.textProperty());  
 question.answerD.bind(fieldD.textProperty());  
  
 prevSelectedQuestion = question;  
 }  
}

**QuizApp**

package com.example.quizapplication;  
  
import javafx.application.Application;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.stage.Stage;  
  
  
public class QuizApp extends Application {  
  
 @Override  
 public void start(Stage stage) throws Exception {  
 Parent root = FXMLLoader.*load*(getClass().getResource("main.fxml"));  
  
 stage.setScene(new Scene(root));  
 stage.show();  
 }  
  
 public static class Launcher {  
 public static void main(String[] args) {  
 Application.*launch*(QuizApp.class, args);  
 }  
 }  
}

**QuestionModal**

package com.example.quizapplication.Model;  
import javafx.beans.property.SimpleStringProperty;  
import javafx.beans.property.StringProperty;  
  
  
public class QuestionModel {  
  
 public StringProperty text = new SimpleStringProperty(); // StringProperty - это абстрактный базовый класс для наблюдаемых свойств строки  
 public StringProperty answerA = new SimpleStringProperty(); // SimpleStringProperty - это конкретная реализация.  
 public StringProperty answerB = new SimpleStringProperty();  
 public StringProperty answerC = new SimpleStringProperty();  
 public StringProperty answerD = new SimpleStringProperty();  
  
 public Question toData() { // возвращение вопроса и ответа  
 return new Question(text.get(), answerA.get(), answerB.get(), answerC.get(), answerD.get());  
 }  
  
 @Override  
 public String toString() {  
 return text.get();  
 }  
  
  
 public static class Question {  
  
 private String text;  
  
 private String answerA;  
 private String answerB;  
 private String answerC;  
 private String answerD;  
  
 public Question() { }  
  
 public Question(String text, String answerA, String answerB, String answerC, String answerD) {  
 this.text = text;  
 this.answerA = answerA;  
 this.answerB = answerB;  
 this.answerC = answerC;  
 this.answerD = answerD;  
 }  
  
 public String getText() {  
 return text;  
 }  
  
 public void setText(String text) {  
 this.text = text;  
 }  
  
 public String getAnswerA() {  
 return answerA;  
 }  
  
 public void setAnswerA(String answerA) {  
 this.answerA = answerA;  
 }  
  
 public String getAnswerB() {  
 return answerB;  
 }  
  
 public void setAnswerB(String answerB) {  
 this.answerB = answerB;  
 }  
  
 public String getAnswerC() {  
 return answerC;  
 }  
  
 public void setAnswerC(String answerC) {  
 this.answerC = answerC;  
 }  
  
 public String getAnswerD() {  
 return answerD;  
 }  
  
 public void setAnswerD(String answerD) {  
 this.answerD = answerD;  
 }  
  
 public QuestionModel toModel() {  
 var model = new QuestionModel();  
 model.text.set(text);  
 model.answerA.set(answerA);  
 model.answerB.set(answerB);  
 model.answerC.set(answerC);  
 model.answerD.set(answerD);  
 return model;  
 }  
 }  
}

**Quiz**

package com.example.quizapplication.Model;  
  
import java.util.List;  
  
  
public class Quiz {  
  
 private List<QuestionModel.Question> questions;  
  
 public Quiz() { }  
  
 public Quiz(List<QuestionModel.Question> questions) {  
 this.questions = questions;  
 }  
  
 public List<QuestionModel.Question> getQuestions() {  
 return questions;  
 }  
  
 public void setQuestions(List<QuestionModel.Question> questions) {  
 this.questions = questions;  
 }  
}

**Заключение**

В завершение мы видим, что цели проекта были выполнены такие как сохранение тестов в файл data.json, возможность редактирования и добавления с помощью методов и рабочая платформа с кнопками и полями ввода.