Федеральное государственное образовательное бюджетное
учреждение высшего образования
**«Финансовый университет при Правительстве
Российской Федерации»**

Департамент анализа данных и машинного обучения

Курсовая работа по дисциплине
«Современные технологии программирования»
на тему:

«Разработка игрового приложения "Кто хочет стать миллионером?" с использованием библиотеки JavaFX»

Выполнил:
студент ПИ19-4
Бедак И. А.

Научный руководитель:
доцент, канд. тех. наук
Андриянов Н. А.

Москва
2021

ОГЛАВЛЕНИЕ

I. ВВЕДЕНИЕ

В данный момент игры занимают неотъемлемую часть IT-индустрии. Все начиналось с простых 2D игр с плохим разрешением и доросло до таких технологий как VR, Rey Tracing и т. д.

В свою очередь, и в моей жизни игры занимают большую роль. Всю свою сознательную жизнь, при наличии свободного времени, я проводил в различных играх, начиная от The Sims 3 и Need for Speed: Shift и заканчивая The Witcher 3: Wild Hunt и GTA V. Однако, с другой стороны, мне очень нравились головоломки. И я решил объединить мои увлечения для создания собственного продукта «Кто хочет стать миллионером?»

Телеигра «Кто хочет стать миллионером?» вышла в 2001 году, вскоре после появления на телеканале НТВ телевизионной программы "О, счастливчик". Правила игры знакомы всем: главный приз - один (три) миллион рублей, 15 вопросов, 4 варианта ответа, 3 подсказки, 2 несгораемых суммы. На данный момент существует множество версий игры «Кто хочет стать миллионером?» в том числе компьютерные игры и игры для мобильных телефонов.

Мой продукт будет немного отличаться от настоящей игры. Игроки за правильный ответ на вопрос будут зарабатывать баллы, а не деньги. А также будет 10 вопросов и не будет подсказок. Будет 4 варианта ответа. За каждый правильный ответ игрок будет получать по 10 баллов. Если пользователь сделает ошибку, появится окно с кнопкой выхода. Когда игра будет пройдена, игрок увидит поздравление и кнопку с возможностью пройти игру заново.

В качестве демонстрации будет разработан прототип на языке программирования Java, содержащий в себе 2 решения, взаимодействующих между собой с помощью архитектуры REST: серверная часть с использованием фреймворка Spring Boot, а также клиентская часть: с графическим интерфейсом с использованием библиотеки JavaFX. Оба решения будут использовать модель

MVC, которая разграничивает управляющую логику программы на отдельные компоненты, а за счёт применения Java и виртуальной машины JVM решение будет кроссплатформенным.

II. ПОСТАНОВКА ЗАДАЧИ

В соответствии с выбранной темой требуется разработать клиент-серверное решение с использованием библиотек Spring Boot для сервера и JavaFX для GUI клиента в виде пользовательских классов и таблиц для СУБД.

Со стороны клиента необходимо разработать несколько окон и логику переходов пользователей между ними, а также их дизайн и расположение элементов интерфейса для взаимодействия с пользователем.

Со стороны бекенда необходимо использовать ORM для связи Spring с СУБД, а также модель MVC для отдельного расположение контроллеров, сервисов и репозиториев с логикой таблиц СУБД.

Решение должно выполнять следующие операции:

- Отображать в таблице данные предметной области;
- Для информационной модели, основанной на БД, таблицы должны быть предварительно заполнены записями;
- Добавлять в БД новые объекты, удалять и редактировать их;
- Фильтровать записи в БД, которые удовлетворяют введенному пользователем сложному критерию;
- Сортировать записи;
- Обновлять изменения источника данных в базе данных;
- Загружать данные из БД;
- Отображать статистические данные.

Решение не должно завершаться аварийно: сообщения о некорректном вводе данных, противоречивых или недопустимых значениях данных, при отсутствии данных по функциональному запросу пользователя и других нештатных ситуациях отображать в окнах сообщений. Программа должна иметь содержательные комментарии, которые могут генерировать автоматически составляемую документацию при помощи инструмента JavaDoc.

III. ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

Предметной областью автоматизации является игра. Логика её работы довольно проста. В главном меню пользователь может начать игру, где ему по очереди будут выводиться 10 вопросов с 4 вариантами ответа. За каждый правильный ответ пользователю начисляется N-е количество баллов (зависит от сложности вопроса). Если пользователь отвечает на вопрос неправильно, то ему выводится отдельное окно с сообщением, что он проиграл и кнопкой для возвращения в главное меню. Если пользователь ответил на все 10 вопросов верно, то ему выводится окно с сообщением о победе и кнопкой возвращения в главное меню.

Со стороны клиента было реализовано 3 модели для представления данных:

- Игра – сущность конкретной игры, в которой записано количество баллов и дата создания
- Вопрос – сущность конкретного вопроса. Содержит информацию о вопросе и ответах на него.
- Вопрос в игре – сущность вопроса в конкретной игре, содержит информацию об игре, в которой этот вопрос попался пользователю

Со стороны сервера были реализованы идентичные модели. Более подробно об атрибутах моделей можно ознакомиться в главе «Состав приложения»

IV. АКТУАЛЬНОСТЬ АВТОМАТИЗАЦИИ

Актуальность заключается в том, что видеоигры в настоящее время пользуются огромнейшим спросом. Кроме того, компьютерные игры, ввиду своего экономического успеха, вносят внушительный вклад в развитие мировой экономики в целом. Например, игра Call of Duty: Black Ops в первые 5 дней продаж смогла заработать более 600 млн долларов США, это является рекордной суммой кассовых сборов за 5 дней среди фильмов, книг и видеоигр. Индустрия компьютерных игр дает возможность развитию еще и музыкальной отрасли, благодаря использованию в своих продуктах саундтреков, которые нередко становятся популярными. Существует также сервис, направленный на сбор «пожертвований» для создания компьютерной игры, являющейся, например, ответвлением от существующей популярной игры. Если идея разработчика насчет продолжения придется фанатом по нраву, то они соберут нужную сумму вместе и вскоре появится новая игра.

Это же касается и интеллектуальных игр. Да, на них не заработаешь состояния, но они могут выполнять совершенно другие функции. Например, для обычных посиделок и развлечений с друзьями. Или даже как платформа для изучения материала, в которой за правильный ответ тебе начисляются какие-то баллы.

V. АЛГОРИТМИЧЕСКИЕ РЕШЕНИЯ

Клиент

На рисунке 1 представлена диаграмма переходов между окнами приложения.

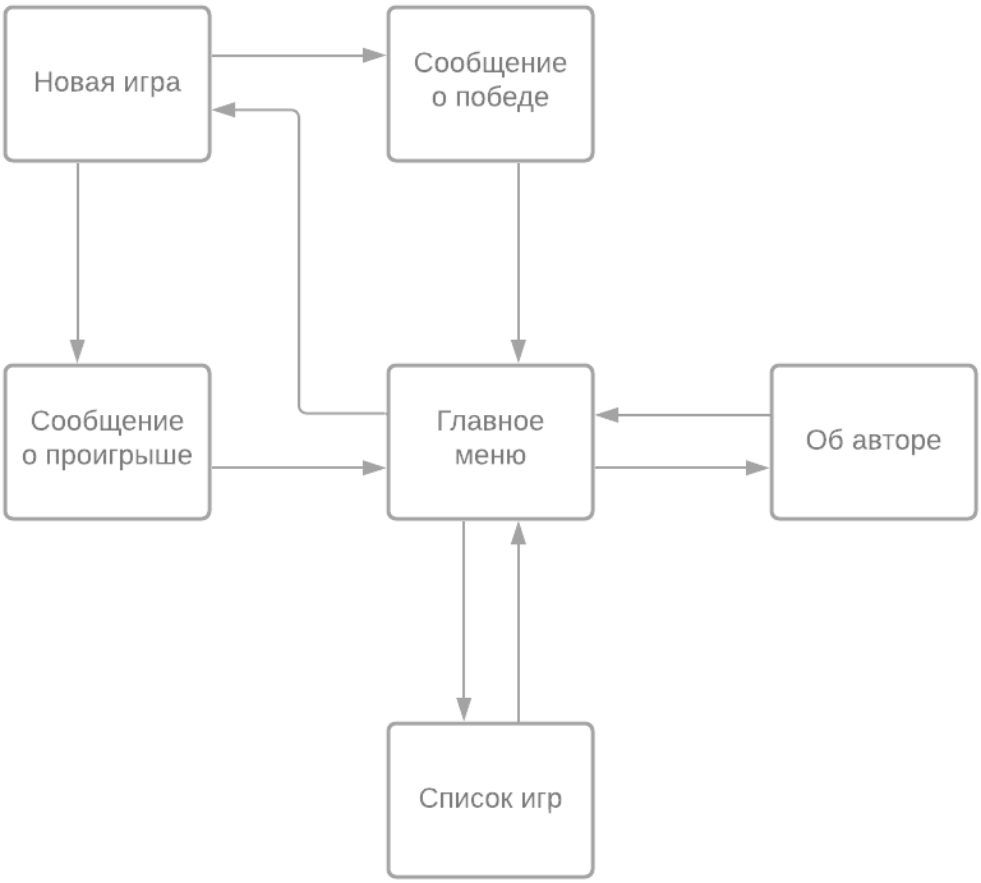

Рисунок 1 – переходы между окнами приложения

При запуске программы запускается главное меню. Тут пользователю предоставляется выбор действий. Он может перейти в раздел «Об авторе». В нём будет написана информация о студенте, написавшем данное приложение. После пользователь может вернуться обратно в главное меню. Второе окно, в которое может попасть пользователь из главного меню – это список игр. Здесь будут показаны все завершенные игры вместе с информацией о них. Из этого окна так же можно вернуться в главное меню. И последний переход, который возможен из главного меню – это создание новой игры. Пользователь перейдёт на новое

окно, где будут по очереди показываться вопросы с вариантами ответа, на которые он должен будет ответить. Всего существует 10 уровней вопросов по сложности. Методы клиента выбирают случайный вопрос по текущему уровню сложности, и передают его. После любого из исходов завершения игры, пользователю будет показано окно с его результатом и кнопкой возврата на главное меню.

Сервер

При получении запроса от пользователя, сервер анализирует переданный путь URL и сопоставляет его с одним из существующих контроллеров.

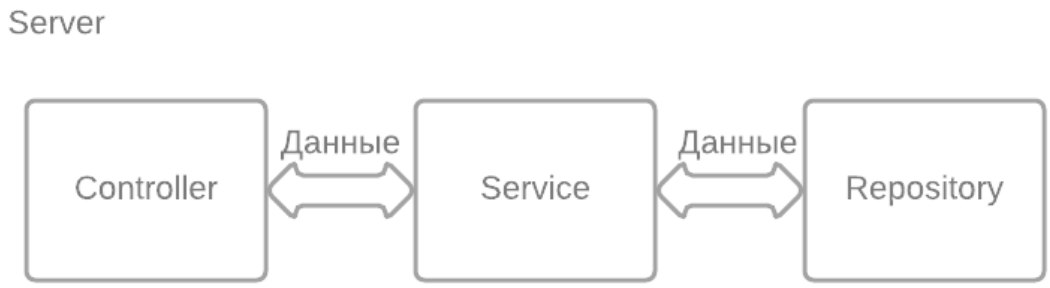

Рисунок 2 – иерархия обработки и передачи данных на сервере

Ошибки, которые могут вернуться от сервера – это ошибка 404 Not Found (ошибка, которая возвращается если в пути URL вызывается не существующий контроллер) и ошибка 400 Bad Request (появляется в случае некорректных или неполных переданных данных).

Если же все данные полные и верные, то контроллер, указанный в пути URL, вызывает соответствующий сервис, который обрабатывает данные и передаёт их дальше в репозиторий (в моей программе – наследник интерфейса JpaRepository). Далее репозиторий обрабатывает данные, возвращает результат сервису, который в свою очередь возвращает данные контроллеру. Контроллер

снова обрабатывает данные и возвращает уже соответствующий ответ пользователю.

VI. ОПИСАНИЕ ИНТЕРФЕЙСА ПРОГРАММЫ

У сервера не существует графического интерфейса. Однако клиент полностью к этому расположен. Для создания графического интерфейса была выбрана библиотека JavaFX. В основном, данная библиотека была выбрана из-за своей простоты и совпадением с изучаемым материалом в университете. Однако, я считаю, что та библиотека полностью выполняет все задачи, для которых она нужна в данной работе.

При создании форм интерфейса были использованы следующие элементы:

- Button
- Label
- SplitPane

Окно главного меню

Данное окно запускается при старте приложения. Пользователь может начать новую игру, посмотреть статистику по завершенным играм и открыть окно информации об авторе. При запуске данного окна в первый раз клиент отправляет запрос и получает от сервера список всех Игр.

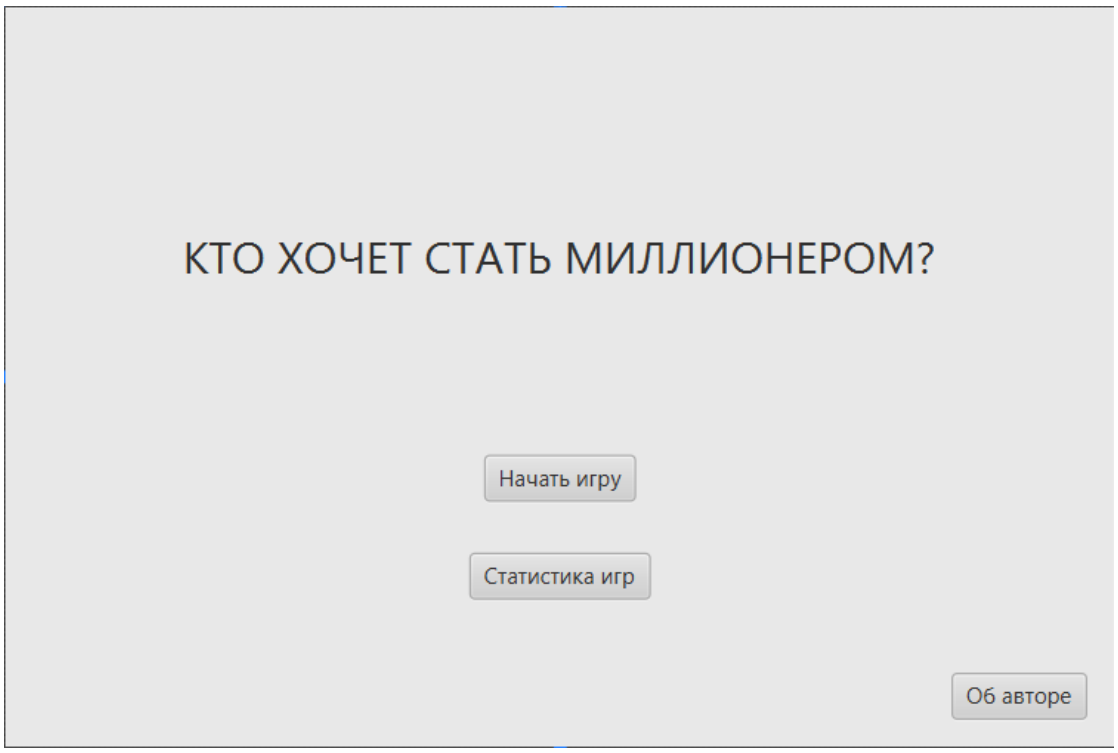

Рисунок 3 – окно главного меню

Интерфейс главного меню представлен на рисунке 3 выше.

Окно об авторе

Окно показывает основную информацию о разработанной программе, а также ФИО автора. Интерфейс формы представлен на рисунке 4 ниже.

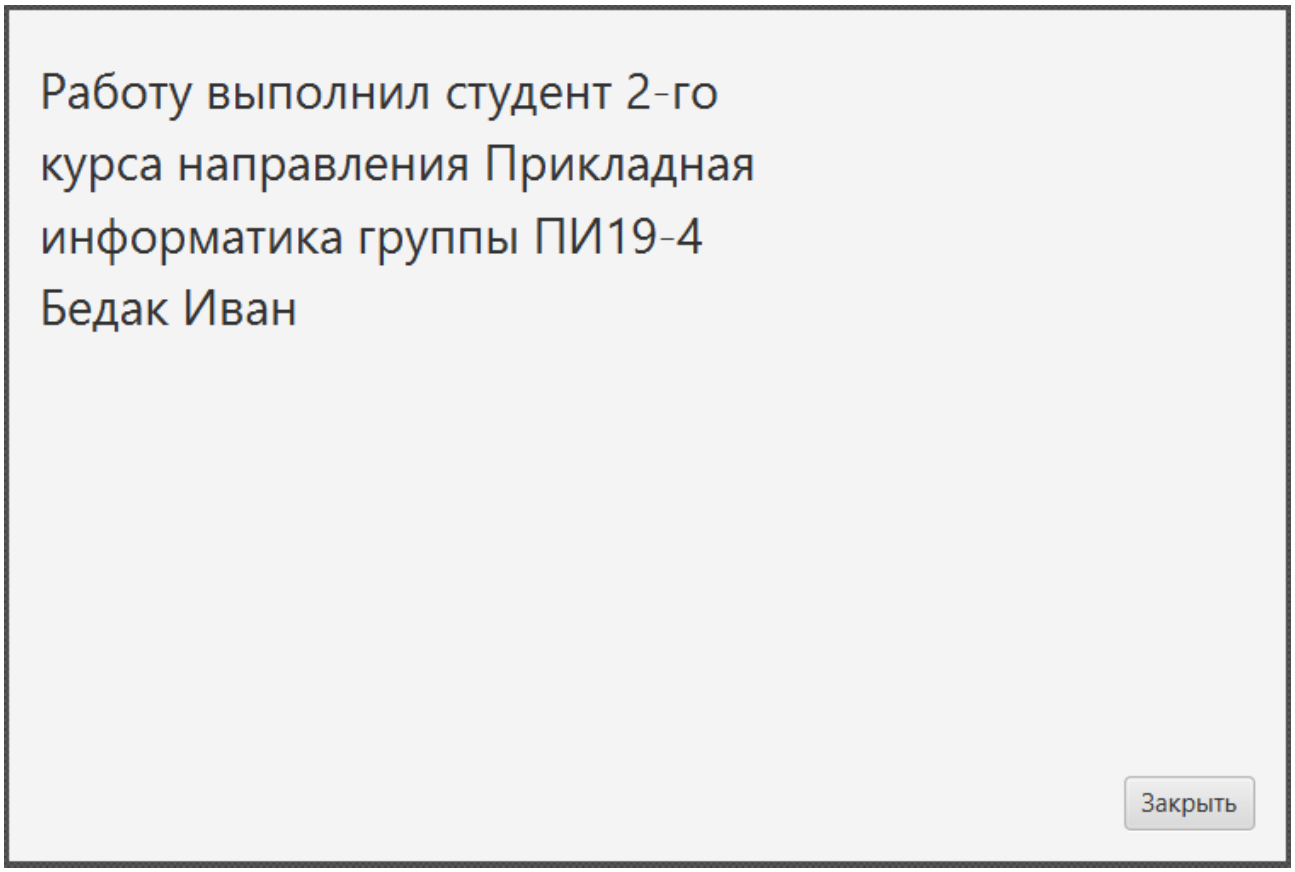

Рисунок 4 – информация о программе и авторе

При нажатии на кнопку «Закрыть» пользователь перенаправляется обратно на главное меню программы.

Окно список игр

В это окно можно попасть из главного меню. Слева окна расположены все игры, которые записаны в базе данных. Справа – динамическая таблица, с информацией об игре. При нажатии на любую из игр слева – таблица справа будет обновляться и показывать актуальную информацию.

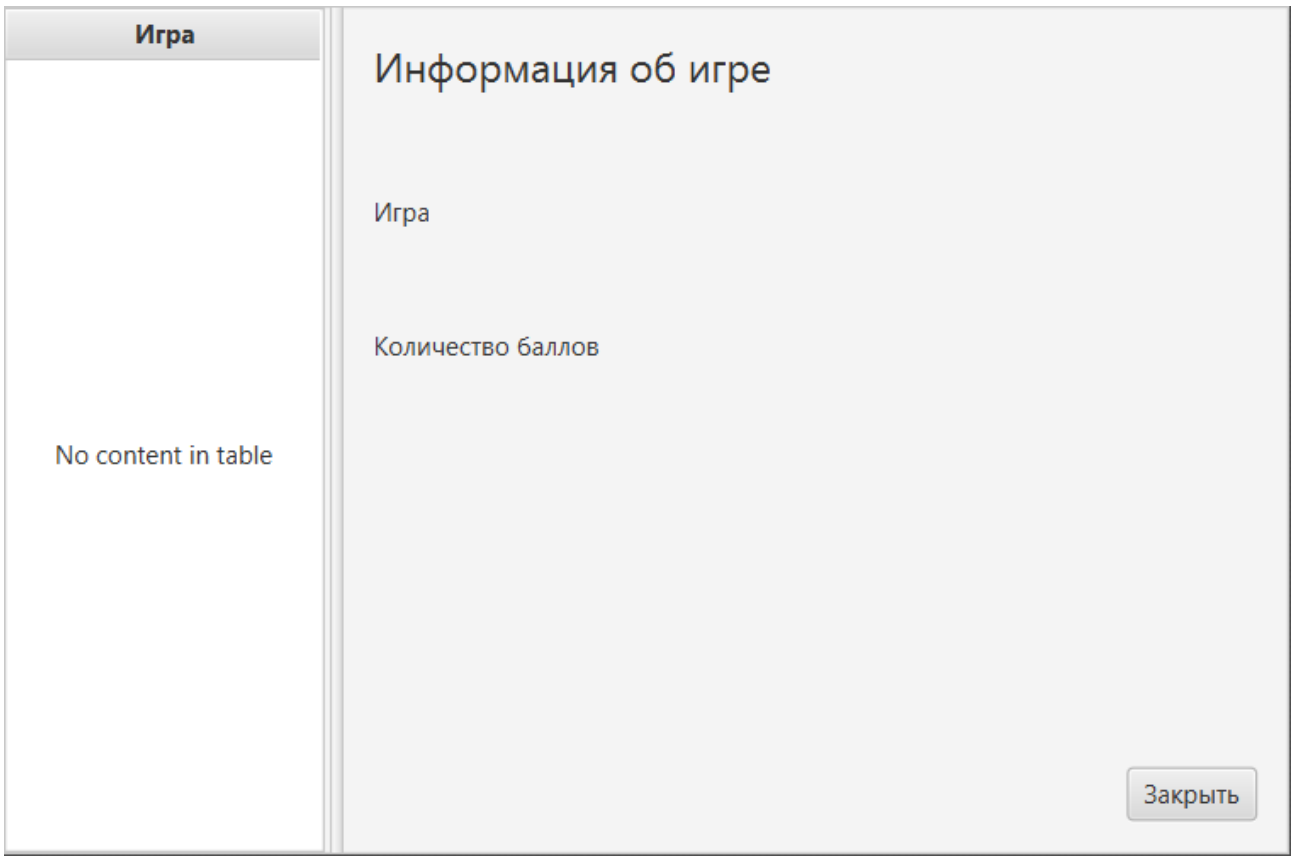

Рисунок 5 – интерфейс окно статистики игр

При нажатии кнопки «Закрыть» пользователь будет перенаправлен в окно главного меню.

Окно игры

Данное окно представляет из себя Label с вопросом, 4 кнопками и подсчетом количества баллов. Все объекты окна меняются при смене вопроса. Пользователь остаётся в этом окне до тех пор, пока не ответит на все 10 вопросов верно, или пока не ошибётся.

Рисунок 6 – интерфейс окна игры

Интерфейс окна игры с вопросом представлена на рисунке 6 выше.

Окно результата игры

Данное окно появляется после завершения игры, не зависимо от того, проиграл пользователь или выиграл. На экране будет запись о результате и количестве баллов, которые заработал пользователь.

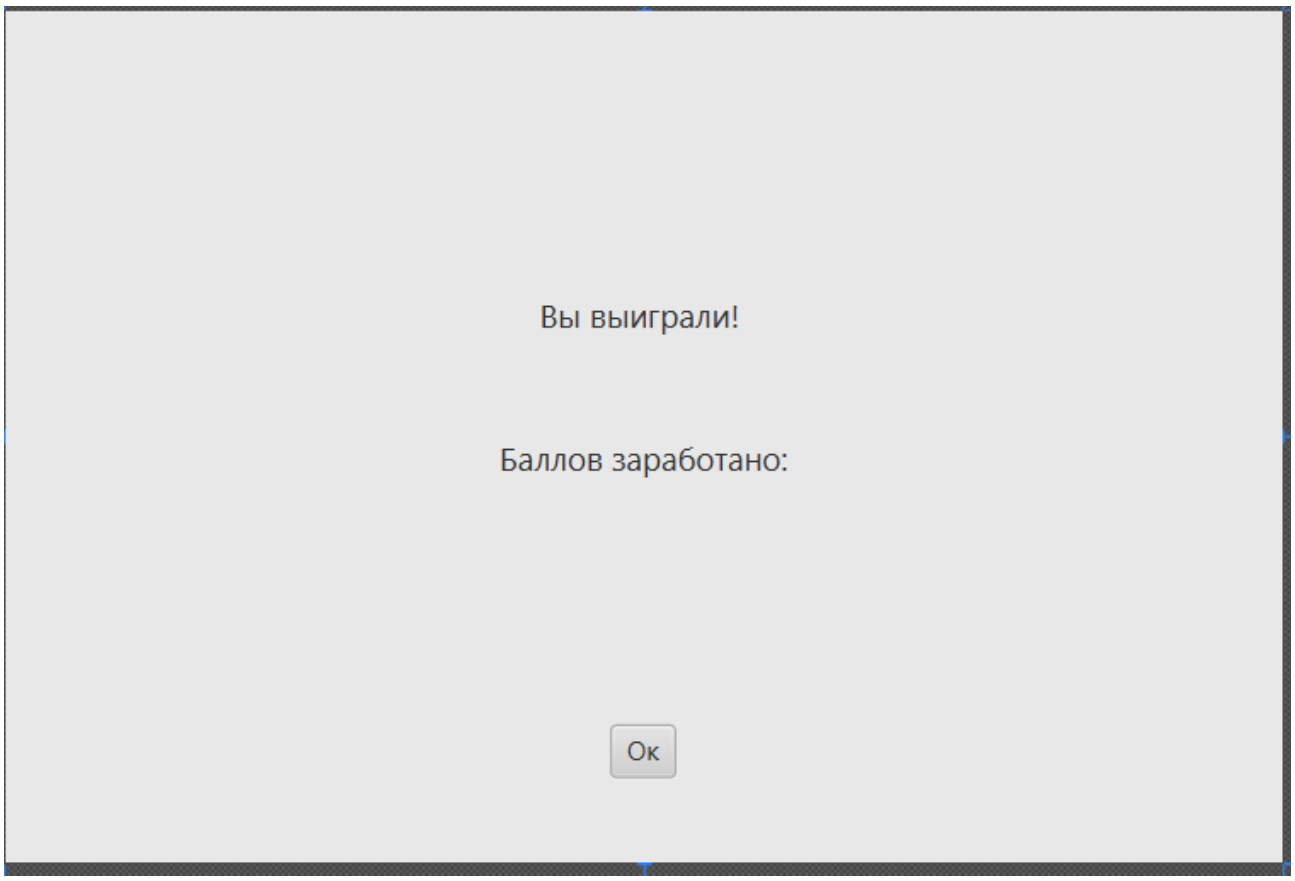

Рисунок 7 – интерфейс окна завершения игры

После нажатия на кнопку «Ок» пользователь будет перенаправлен в главное меню.

VII. СОСТАВ ПРИЛОЖЕНИЯ

Сервер

В состав сервера входят следующие компоненты:

- Spring Web для создания web-приложений, в том числе RESTful, с использованием Spring MVC (Model-View-Controller).
- MySQL Driver – используется для связи сервера и базы данных MySQL
- Lombok – удобный набор аннотаций для логирования и автопостроения setter’ов и getter’ов;
- Apache maven – основной сборщик проекта;
- Spring-boot-maven-plugin – плагин для сборки решения.

База данных

На рисунке ниже представлена диаграмма таблиц в БД.

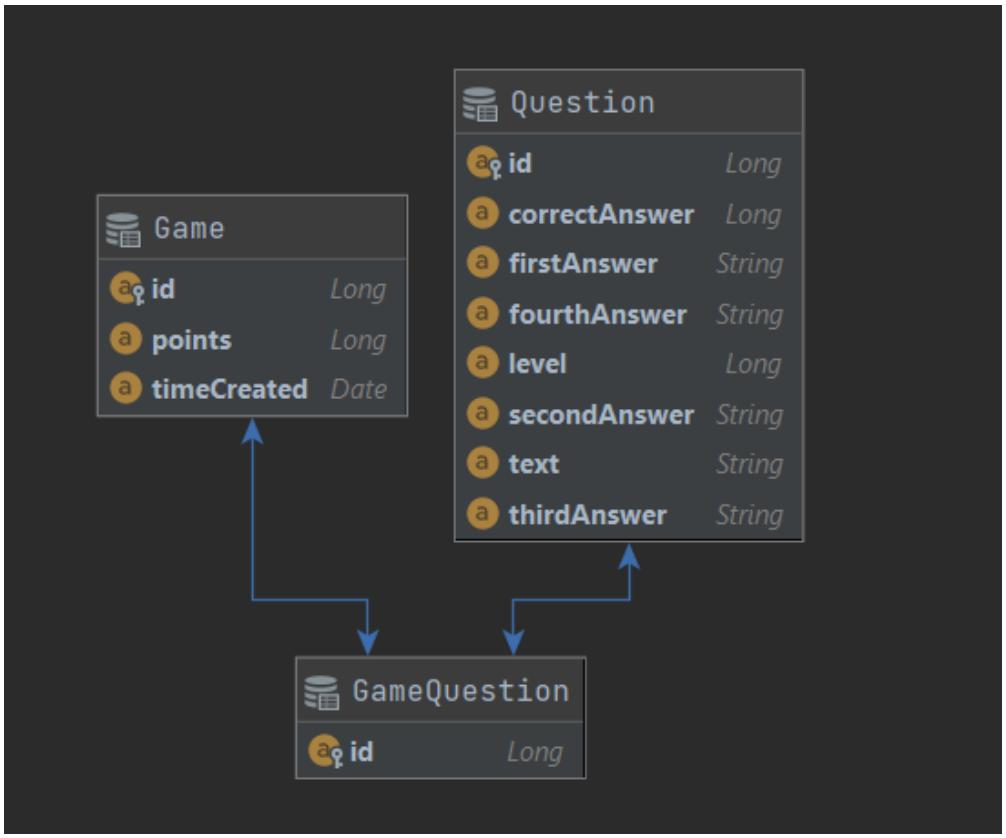

Рисунок 8 – диаграмма таблиц базы данных

В качестве базы данных была выбрана MySQL, которая состоит из следующий таблиц и полей:

1. Games – таблица игр завершенных пользователем. Хранит в себе следующие атрибуты:
    a. id – уникальный идентификатор игры.
    b. points – количество баллов, заработанных пользователем в игре.
    c. time_created – время начала игры.

2. Questions – таблица, в которой хранятся все вопросы, которые могут попасться пользователю при игре. Хранит в себе следующие атрибуты:
    a. id – уникальный идентификатор вопроса.
    b. level – уровень сложности вопроса, необходим для того, чтобы со временем в игре вопросы становились сложнее.
    c. text – текст вопроса.
    d. first_answer – первый ответ.
    e. second_answer – второй ответ.
    f. third_answer – третий ответ.
    g. forth_answer – четвёртый ответ.
    h. correct_answer – номер правильного ответа (1-4).

3. Game_questions – таблица для хранения буфера вопросов, которые попадались пользователю. Необходима для отображения информации об игре. Хранит в себе следующие атрибуты:
    a. id – уникальный идентификатор вопроса в игре.
    b. game_id – уникальный идентификатор игры, в которой попался определённый вопрос.
    c. question_id – уникальный идентификатор вопроса, который попался в определённой игре.

Клиент

В состав клиента входят следующие компоненты/внешние библиотеки:

- JavaFX (javafx-controls, javafx-fxml, javafx-graphics) – библиотека для отображения GUI и её компоненты.
- Google GSON – модуль для работы с запаковкой/распаковкой данных в формате JSON.
- Apache maven – основной сборщик решения (автозагрузка и менеджмент необходимых библиотек, работа с плагинами).
- Javafx-maven-plugin – плагин для автоподстановки флагов виртуальной машины Java (VM Options) библиотеки JavaFX при компиляции и запуске проекта.

VIII. НАЗНАЧЕНИЕ И СОСТАВ КЛАССОВ ПРОГРАММЫ

Сервер

Всего на сервере задействовано 20 классов, которые можно объединить в следующие группы:

- Сущности (3) – представляют из себя коллекцию для работы с элементами базы данных
- Репозитории (3) – представления таблиц БД
- Контроллеры (3) – классы для обработки HTTP запросов от клиента
- Сервисы (6) – необходимы для связи репозитория и контроллера
- Обработчики ошибок (3) – необходимы для разделения ошибок при некорректных HTTP запросах
- Класс для тестов (1)
- Основной класс (1)

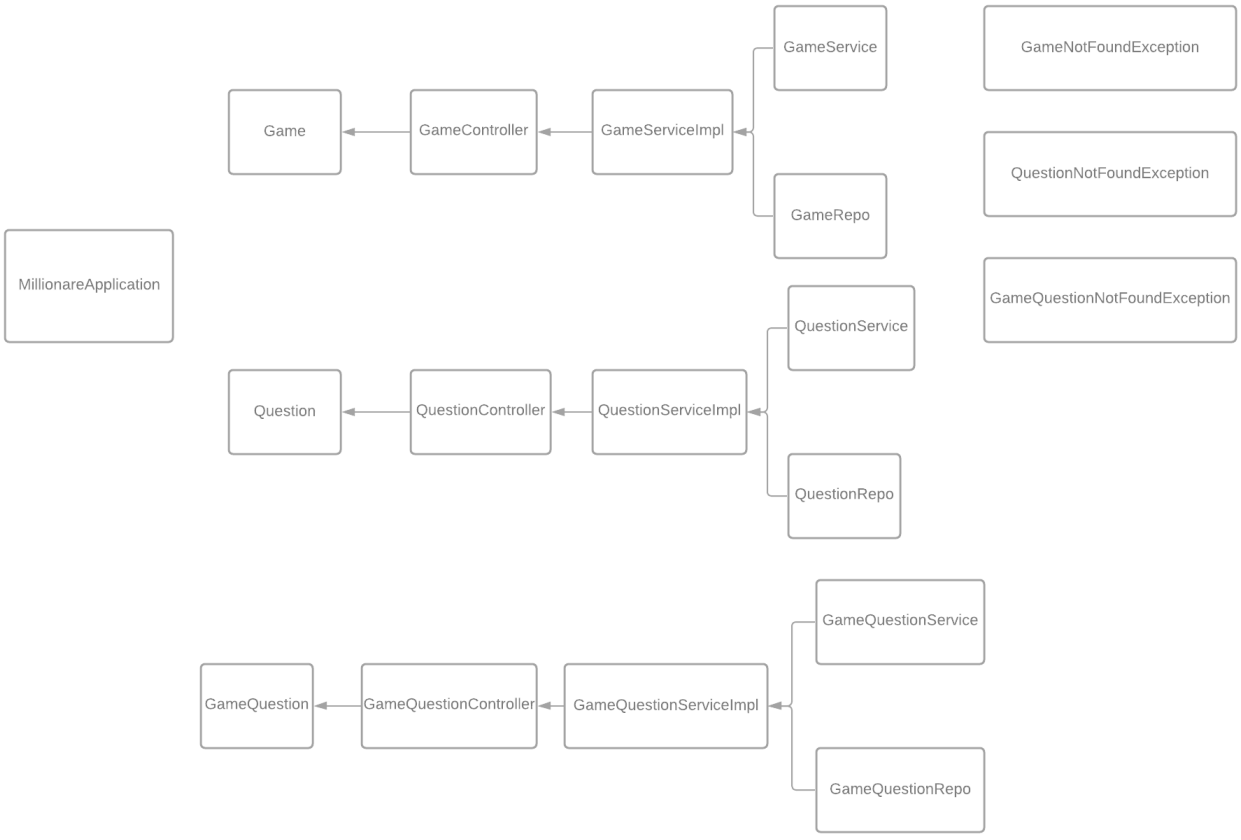

Рисунок 9 – диаграмма классов сервера

Перечень классов сервера:

- Game – класс для представления игры в СУБД
- Question – класс для представления вопроса в СУБД
- GameQuestion – класс для представления вопроса в определённой игре в СУБД
- GameController – класс для обработки HTTP запросов, связанных с игрой
- QuestionController – класс для обработки HTTP запросов, связанных с вопросом
- GameQuestionController – класс для обработки HTTP запросов, связанных с вопросом в определённой игре
- GameRepo – интерфейс для работы с коллекцией игр, наследуется от JpaRepository
- QuestionRepo – интерфейс для работы с коллекцией вопросов, наследуется от JpaRepository
- GameQuestionRepo – интерфейс для работы с коллекцией вопросов в определённой игре, наследуется от JpaRepository
- GameService – интерфейс сервиса для работы с играми
- GameServiceImpl – класс-сервис для работы с играми, наследуется от GameService
- QuestionService – интерфейс сервиса для работы с вопросами
- QuestionServiceImpl – класс-сервис для работы с вопросами, наследуется от QuestionService
- GameQuestionService – интерфейс сервиса для работы с вопросами в определённой игре
- GameQuestionServiceImpl – класс-сервис для работы с вопросами в определённой игре, наследуется от GameQuestionService
- GameNotFoundException – исключение, вызываемое в случае, когда нет игры по id заданному в HTTP запросе. Наследуется от RuntimeException

- QuestionNotFoundException – исключение, вызываемое в случае, когда нет вопроса по id заданному в HTTP запросе. Наследуется от RuntimeException
- GameQuestionNotFoundException – исключение, вызываемое в случае, когда нет вопроса в определённой игре по id заданному в HTTP запросе. Наследуется от RuntimeException
- MillionareApplication – основной класс программы
- MillionareApplicationTests – класс для тестов

Клиент

На стороне клиента было разработано 11 классов для работы приложения, которые можно объединить в данные группы:

- Модели (3) – модели для обработки сущностей со стороны сервера
- Контроллеры (5) – классы для обработки действий во время работы пользователя с формами интерфейса
- Классы работы с API (2) – классы для обобщенной работы с сервером
- Главный класс приложения

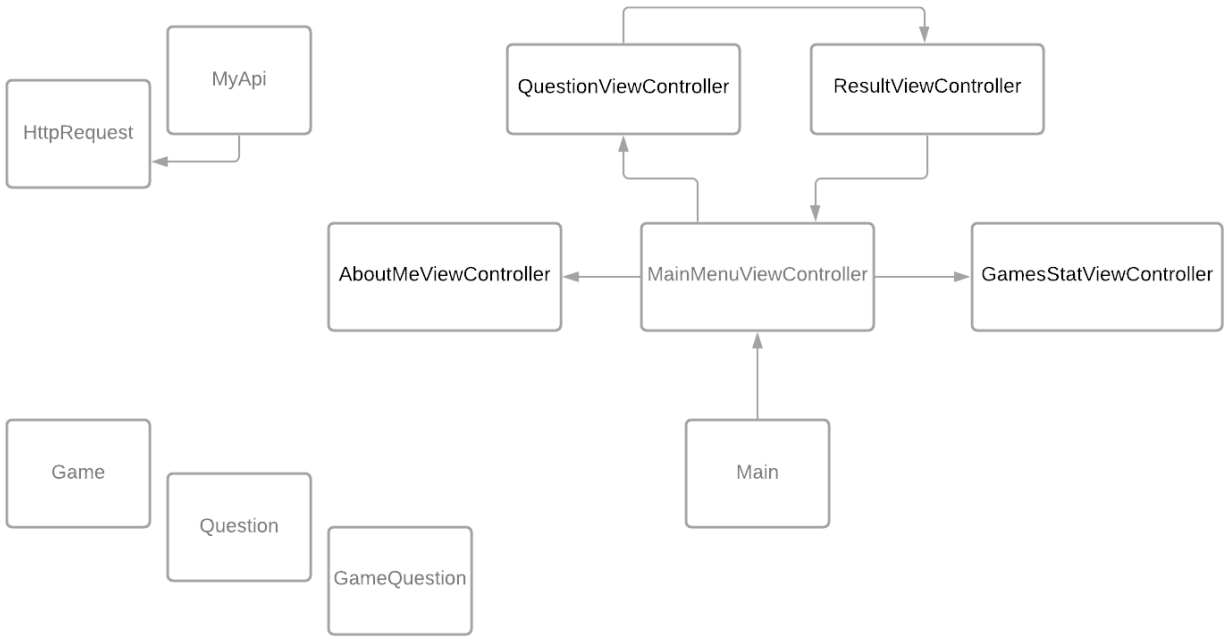

Рисунок 10 – диаграмма классов клиента

Перечень классов клиента:

- HttpRequest – класс со статическими методами для HTTP запроса к серверу
- MyApi – класс для получения данных с сервера посредством HTTP запросов
- Game – модель игры
- Question – модель вопроса
- GameQuestion – модель вопроса в определённой игре
- AboutMeViewController – класс для обработки действий пользователя в окне информации о приложении и авторе
- GamesStatViewController – класс для обработки действий пользователя в окне статистики игр
- MainMenuViewController – класс для обработки действий пользователя в главном окне программы
- QuestionViewController – класс для обработки действий пользователя в окне новой игры
- – класс для обработки действий пользователя в окне вывода информации о завершенной игре
- Main – главный класс программы

Также, на стороне клиента были разработаны следующие формы интерфейса:

- AboutMeView.fxml – информация об авторе
- GamesStatView.fxml – форма для просмотра статистики по завершенным играм
- MainMenuView.fxml – главное меню, первое окно, которое запустистя после старта программы
- QuestionView.fxml – динамическая форма для игры
- ResutlView.fxml – динамическая форма с информацией о завершенной игре

IX. ЗАКЛЮЧЕНИЕ

В ходе написания данной работы было разработано клиент-серверное приложение для игры в популярную телевикторину «Кто хочет стать миллионером?», включающее в себя сервер с использованием библиотеки Spring Boot и клиент с применением GUI библиотеки JavaFX.

Для написания данного приложения была выбрана СУБД MySQL. Из её плюсов могу выделить:

1. MySQL кроссплатформенная и не требует лицензии Windows.

2. Она полностью бесплатная.

3. Не требует мощного ПО для работы.

4. Одна из самых популярных СУБД, поэтому есть четко написанная документация, и практически на любую проблему можно найти решение на специализированных сервисах.

Т. к. целью этой работы не было коммерческое приложение, готовое к выходу на рынок, я считаю, что MySQL оптимальный вариант.

Созданное решение удовлетворяет всем требованиям и задачам: реализует CRUD-методы со стороны бекенда, а фронт осуществляет получение и отправку данных через RESTful API с помощью протокола http с последующим вводом/выводом данных на элементы управления графического интерфейса JavaFX.

Решение может модернизироваться и обновляться. К примеру, могу привести следующие модификации:

- Добавить стили .css и возможность вставлять картинки
- Добавить возможность зарегистрироваться и играть под своим ником, сохранять свою статистику и сравнивать её со статистикой других игроков

- Добавить возможность использования подсказок, таких как «50:50», «Помощь зала», «Звонок другу» (в данном случае будет просто случайно генерироваться ответ с точностью в зависимости от сложности вопроса)
- Добавить музыкальное сопровождение

Как можно заметить, существует еще множество способов усовершенствовать приложение, чтобы создать уже полноценный продукт, готовый использованию вне стен университета.

X. СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

Законодательные и нормативные акты

1. ГОСТ 2.316-2008. Правила нанесения надписей, технических требований и таблиц на графических документах.
2. ГОСТ 7.1-2003. Библиографическая запись. Библиографическое описание. Общие требования и правила составления. – М.: ИПК Издательство стандартов, 2004. – 169 с.
3. ГОСТ 7.32-2001. Система стандартов по информацию, библиотечному и издательскому делу. Отчет о научно-исследовательской работе. Структура и правила оформления. – М.: ИПК Издательство стандартов, 2001. – 21 с.

Учебная и научная литература

1. Козмина Ю., Харроп Р. Spring 5 для профессионалов. – Киев: Диалектика-Вильямс, 2019. – 1120 с.
2. Коузен К. Современный Java. Рецепты программирования. – М.: ДМК Пресс, 2018. – 274 с.
3. Мартин Роберт К. Чистый код. Создание анализ и рефакторинг. – СПб: Питер, 2018. – 274 с.
4. Прохоренок Н. А. JavaFX. – СПб: БХВ-Петербург, 2020. – 768 с.

ПРИЛОЖЕНИЕ

Сервер

Game.java

```
package Millionare.Millionare.entity;

import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import org.hibernate.annotations.CreationTimestamp;

import javax.persistence.*;
import java.util.Date;
import java.util.Set;

@Setter
@Getter
@NoArgsConstructor
@Entity
@Table(name = "games", uniqueConstraints =
{@UniqueConstraint(columnNames={"id"})})
public class Game {

    public Game(Long points){
        this.points = points;
    }

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id", nullable = false)
    private Long id;

    @Column(name = "points", nullable = false)
    private Long points;

    @OneToMany(mappedBy = "gameId")
    private Set<GameQuestion> gameQuestions;

    @CreationTimestamp
    @Temporal(TemporalType.TIMESTAMP)
    @Column(name = "time_created")
    private Date timeCreated;

}
```

Question.java

```
package Millionare.Millionare.entity;

import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

import javax.persistence.*;
import java.util.Set;

@Setter
```

```
@Getter
@NoArgsConstructor
@Entity
@Table(name = "questions",uniqueConstraints =
{@UniqueConstraint(columnNames={"id"})})
public class Question {

    public Question(Long level, String text, String firstAnswer, String
secondAnswer, String thirdAnswer, String fourthAnswer, Long correctAnswer){
        this.level = level;
        this.text = text;
        this.firstAnswer = firstAnswer;
        this.secondAnswer = secondAnswer;
        this.thirdAnswer = thirdAnswer;
        this.fourthAnswer = fourthAnswer;
        this.correctAnswer = correctAnswer;
    }

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id", nullable = false)
    private Long id;

    @Column(name = "level", nullable = false)
    private Long level;

    @Column(name = "text", length = 256, nullable = false)
    private String text;

    @Column(name = "first_answer", length = 128, nullable = false)
    private String firstAnswer;

    @Column(name = "second_answer", length = 128, nullable = false)
    private String secondAnswer;

    @Column(name = "third_answer", length = 128, nullable = false)
    private String thirdAnswer;

    @Column(name = "fourth_answer", length = 128, nullable = false)
    private String fourthAnswer;

    @Column(name = "correct_answer", nullable = false)
    private Long correctAnswer;

    @OneToMany(mappedBy = "questionId")
    private Set<GameQuestion> questionGames;
}
```

GameQuestion.java

```
package Millionare.Millionare.entity;

import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

import javax.persistence.*;

@Setter
@Getter
```

```
@NoArgsConstructor
@Entity
@Table(name = "game_questions")
public class GameQuestion {

    public GameQuestion(Long gameId, Long questionId){
        this.gameId = gameId;
        this.questionId = questionId;
    }

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;


    private Long gameId;

    private Long questionId;
}
```

GameController.java

```
package Millionare.Millionare.controller;

import Millionare.Millionare.entity.Game;
import Millionare.Millionare.service.GameService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
public class GameController {
    private final GameService gameService;

    @Autowired
    public GameController(GameService gameService){
        this.gameService = gameService;
    }

    @PostMapping("/games")
    Game newGame(@RequestBody Game newGame){
        System.out.println(newGame);
        return gameService.create(newGame);
    }

    @GetMapping(value = "/games")
    public ResponseEntity<List<Game>> readGames() {
        final List<Game> games = gameService.readAll();

        return games != null && !games.isEmpty()
                ? new ResponseEntity<>(games, HttpStatus.OK)
                : new ResponseEntity<>(HttpStatus.NOT_FOUND);
    }

    @GetMapping(value = "/games/{id}")
    public ResponseEntity<Game> readGame(@PathVariable(name = "id") Long id) {
        final Game game = gameService.read(id);
```

```
        return game != null
                ? new ResponseEntity<>(game, HttpStatus.OK)
                : new ResponseEntity<>(HttpStatus.NOT_FOUND);
    }

    @PutMapping(value = "/games/{id}")
    public ResponseEntity<?> updateGame(@RequestBody Game game,
@PathVariable(name = "id") Long id) {
        if (gameService.update(game, id)) {
            return new ResponseEntity<>(HttpStatus.OK);
        } else {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
    }

    @DeleteMapping("/games/{id}")
    public ResponseEntity<?> deleteGame(@PathVariable(name = "id") Long id) {
        if (gameService.read(id) != null) {
            gameService.delete(id);
            return new ResponseEntity<>(HttpStatus.OK);
        } else {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
    }
}
```

QuestionController.java

```
package Millionare.Millionare.controller;

import Millionare.Millionare.entity.Question;
import Millionare.Millionare.service.QuestionService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
public class QuestionController {
    private final QuestionService questionService;

    @Autowired
    public QuestionController(QuestionService questionService){
        this.questionService = questionService;
    }

    @PostMapping("/questions")
    Question newQuestion(@RequestBody Question newQuestion){
        System.out.println(newQuestion);
        return questionService.create(newQuestion);
    }

    @GetMapping(value = "/questions")
    public ResponseEntity<List<Question>> readQuestions() {
        final List<Question> questions = questionService.readAll();

        return questions != null && !questions.isEmpty()
                ? new ResponseEntity<>(questions, HttpStatus.OK)
                : new ResponseEntity<>(HttpStatus.NOT_FOUND);
```

```
    }

    @GetMapping(value = "/questions/{id}")
    public ResponseEntity<Question> readQuestion(@PathVariable(name = "id") Long
id) {
        final Question question = questionService.read(id);

        return question != null
                ? new ResponseEntity<>(question, HttpStatus.OK)
                : new ResponseEntity<>(HttpStatus.NOT_FOUND);
    }

    @PutMapping(value = "/questions/{id}")
    public ResponseEntity<?> updateQuestion(@RequestBody Question question,
@PathVariable(name = "id") Long id) {
        if (questionService.update(question, id)) {
            return new ResponseEntity<>(HttpStatus.OK);
        } else {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
    }

    @DeleteMapping("/questions/{id}")
    public ResponseEntity<?> deleteQuestion(@PathVariable(name = "id") Long id)
{
        if (questionService.read(id) != null) {
            questionService.delete(id);
            return new ResponseEntity<>(HttpStatus.OK);
        } else {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
    }
}
```

GameQuestionController.java

```
package Millionare.Millionare.controller;

import Millionare.Millionare.entity.GameQuestion;
import Millionare.Millionare.service.GameQuestionService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
public class GameQuestionController {

    private final GameQuestionService gameQuestionService;

    @Autowired
    public GameQuestionController(GameQuestionService gameQuestionService){
        this.gameQuestionService = gameQuestionService;
    }

    @PostMapping("/game_questions")
    GameQuestion newGameQuestion(@RequestBody GameQuestion newGameQuestion){
        System.out.println(newGameQuestion);
        return gameQuestionService.create(newGameQuestion);
```

```
    }

    @GetMapping(value = "/game_questions")
    public ResponseEntity<List<GameQuestion>> readGameQuestions() {
        final List<GameQuestion> gameQuestions = gameQuestionService.readAll();

        return gameQuestions != null && !gameQuestions.isEmpty()
                ? new ResponseEntity<>(gameQuestions, HttpStatus.OK)
                : new ResponseEntity<>(HttpStatus.NOT_FOUND);
    }

    @GetMapping(value = "/game_questions/{id}")
    public ResponseEntity<GameQuestion> readGameQuestion(@PathVariable(name =
"id") Long id) {
        final GameQuestion gameQuestion = gameQuestionService.read(id);

        return gameQuestion != null
                ? new ResponseEntity<>(gameQuestion, HttpStatus.OK)
                : new ResponseEntity<>(HttpStatus.NOT_FOUND);
    }

    @PutMapping(value = "/game_questions/{id}")
    public ResponseEntity<?> updateGameQuestion(@RequestBody GameQuestion
gameQuestion, @PathVariable(name = "id") Long id) {
        if (gameQuestionService.update(gameQuestion, id)) {
            return new ResponseEntity<>(HttpStatus.OK);
        } else {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
    }

    @DeleteMapping("/game_questions/{id}")
    public ResponseEntity<?> deleteGameQuestion(@PathVariable(name = "id") Long
id) {
        if (gameQuestionService.read(id) != null) {
            gameQuestionService.delete(id);
            return new ResponseEntity<>(HttpStatus.OK);
        } else {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
    }

}
```

Клиент

HttpRequest.java

```
package sample.api;

import com.google.gson.Gson;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.net.HttpURLConnection;
import java.net.URL;
import java.nio.charset.StandardCharsets;
import java.util.Map;
```

```
public class HttpRequest {

    public static String sendGET(String urlString) {
        try {
            URL url = new URL(urlString);
            HttpURLConnection con = (HttpURLConnection) url.openConnection();
            con.setRequestMethod("GET");
            int responseCode = con.getResponseCode();

            if (responseCode == HttpURLConnection.HTTP_OK) {
                BufferedReader in = new BufferedReader(new
InputStreamReader(con.getInputStream()));
                String inputLine;
                StringBuilder response = new StringBuilder();
                while ((inputLine = in.readLine()) != null) {
                    response.append(inputLine);
                }
                in.close();
                return response.toString();

            } else {
                System.out.println("GET получил код " + con.getResponseCode());
                return null;
            }
        } catch (Exception e) {
            return null;
        }
    }

    public static String sendPOST(String urlString, Map<String, String>
paramsMap) {
        try {
            URL obj = new URL(urlString);
            HttpURLConnection con = (HttpURLConnection) obj.openConnection();
            con.setRequestProperty("Content-Type", "application/json; utf-8");
            con.setRequestProperty("Accept", "application/json");
            con.setRequestMethod("POST");

            Gson gsonObj = new Gson();

            String jsonStr = gsonObj.toJson(paramsMap);

            con.setDoOutput(true);
            OutputStream stream = con.getOutputStream();

            stream.write(jsonStr.getBytes(StandardCharsets.UTF_8));
            stream.flush();
            stream.close();

            int responseCode = con.getResponseCode();

            if (responseCode == HttpURLConnection.HTTP_OK) {
                BufferedReader in = new BufferedReader(new
InputStreamReader(con.getInputStream()));
                String inputLine;
                StringBuilder response = new StringBuilder();

                while ((inputLine = in.readLine()) != null) {
                    response.append(inputLine);
                }
                in.close();
                return response.toString();

            } else {
```

```
                System.out.println("POST получил код " + con.getResponseCode());
                return null;
            }
        } catch (Exception e) {
            return null;
        }
    }

    public static String sendPUT(String urlString, String jsonString) {
        try {
            URL obj = new URL(urlString);
            HttpURLConnection con = (HttpURLConnection) obj.openConnection();
            con.setRequestProperty("Content-Type", "application/json; utf-8");
            con.setRequestProperty("Accept", "application/json");
            con.setRequestMethod("PUT");
            con.setDoOutput(true);
            OutputStream stream = con.getOutputStream();

            stream.write(jsonString.getBytes(StandardCharsets.UTF_8));
            stream.flush();
            stream.close();

            int responseCode = con.getResponseCode();

            if (responseCode == HttpURLConnection.HTTP_OK) {
                BufferedReader in = new BufferedReader(new
InputStreamReader(con.getInputStream()));
                String inputLine;
                StringBuilder response = new StringBuilder();

                while ((inputLine = in.readLine()) != null) {
                    response.append(inputLine);
                }
                in.close();
                return response.toString();

            } else {
                System.out.println("PUT получил код " + con.getResponseCode());
                return null;
            }
        } catch (Exception e) {
            return null;
        }
    }

}
```

MyApi.java

```
package sample.api;

import com.google.gson.JsonArray;
import com.google.gson.JsonObject;
import com.google.gson.JsonParser;
import sample.model.Game;
import sample.model.GameQuestion;
import sample.model.Question;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
```

```
import java.util.Map;

public class MyApi {

    public List<Game> getGames(){
        List<Game> resultList = new ArrayList<>();


        String URL = "http://localhost:8080/games";
        String response = HttpRequest.sendGET(URL);
        System.out.println(response);
        if (response != null) {
            JsonArray jsonResult =
JsonParser.parseString(response).getAsJsonArray();

                for (int i = 0; i < jsonResult.size(); i++) {
                    JsonObject currentGame =
jsonResult.get(i).getAsJsonObject();

                    String id = currentGame.get("id").getAsString();
                    String points = currentGame.get("points").getAsString();

                    Game game = new Game(id, points);

                    resultList.add(game);

            }
        }
        return resultList;
    }

    public List<Question> getQuestions(){
        List<Question> resultList = new ArrayList<>();


        String URL = "http://localhost:8080/questions";
        String response = HttpRequest.sendGET(URL);

        if (response != null) {
            JsonArray jsonResult =
JsonParser.parseString(response).getAsJsonArray();

                for (int i = 0; i < jsonResult.size(); i++) {
                    JsonObject currentGame =
jsonResult.get(i).getAsJsonObject();

                    String id = currentGame.get("id").getAsString();
                    String level = currentGame.get("level").getAsString();
                    String text = currentGame.get("text").getAsString();
                    String firstAnswer =
currentGame.get("firstAnswer").getAsString();
                    String secondAnswer =
currentGame.get("secondAnswer").getAsString();
                    String thirdAnswer =
currentGame.get("thirdAnswer").getAsString();
                    String fourthAnswer =
currentGame.get("fourthAnswer").getAsString();
                    String correctAnswer =
currentGame.get("correctAnswer").getAsString();

                    Question question = new Question(id, level, text,
firstAnswer, secondAnswer, thirdAnswer, fourthAnswer, correctAnswer);

                    resultList.add(question);
```

```
                }

            }
            return resultList;
        }

        public List<GameQuestion> getGameQuestions(){
            List<GameQuestion> resultList = new ArrayList<>();


            String URL = "http://localhost:8080/game_questions";
            String response = HttpRequest.sendGET(URL);
            System.out.println(response);
            if (response != null) {
                JsonArray jsonResult =
JsonParser.parseString(response).getAsJsonArray();

                for (int i = 0; i < jsonResult.size(); i++) {
                    JsonObject currentGame = jsonResult.get(i).getAsJsonObject();

                    String id = currentGame.get("id").getAsString();
                    String gameId = currentGame.get("gameId").getAsString();
                    String questionId = currentGame.get("questionId").getAsString();

                    GameQuestion gameQuestion = new GameQuestion(id, gameId,
questionId);

                    resultList.add(gameQuestion);

                }
            }
            return resultList;
        }

        public List<GameQuestion> getGameQuestionsById(String gameId, String
questionId){
            List<GameQuestion> resultList = new ArrayList<>();


            String URL = "http://localhost:8080/game_questions";
            String response = HttpRequest.sendGET(URL);
            System.out.println(response);
            if (response != null) {
                JsonArray jsonResult =
JsonParser.parseString(response).getAsJsonArray();

                for (int i = 0; i < jsonResult.size(); i++) {
                    JsonObject currentGame = jsonResult.get(i).getAsJsonObject();

                    String id = currentGame.get("id").getAsString();
                    String currentGameId = currentGame.get("gameId").getAsString();
                    String currentQuestionId =
currentGame.get("questionId").getAsString();
                    if (currentGameId.equals(gameId) &&
currentQuestionId.equals(questionId)) {
                        GameQuestion gameQuestion = new GameQuestion(id, gameId,
questionId);
                        resultList.add(gameQuestion);
                    }
                }
            }
            return resultList;
        }
```

```
    public Game createGame(String points) {
        String URL = "http://localhost:8080/games";

        Map<String, String> params = new HashMap<>();
        params.put("points", points);

        String response = HttpRequest.sendPOST(URL, params);
        if (response != null) {
            JsonObject jsonResult =
JsonParser.parseString(response).getAsJsonObject();
            String id = jsonResult.get("id").getAsString();
            String currentPoints = jsonResult.get("points").getAsString();

            Game game = new Game(id, currentPoints);
            return game;
        } else {
            return null;
        }
    }

    public void updateGame(Game game){
        String URL = "http://localhost:8080/games/" + game.getId();
        String response = HttpRequest.sendGET(URL);
        if (response != null) {
            JsonObject jsonResult =
JsonParser.parseString(response).getAsJsonObject();

            jsonResult.addProperty("points", game.getPoints());

            System.out.println(jsonResult);

            String jsonString = jsonResult.toString();
            String newResponse = HttpRequest.sendPUT(URL, jsonString);

        }

    }
}
```

Game.java

```
package sample.model;


import javafx.beans.property.*;

import java.util.List;

public class Game {

    private StringProperty id;
    private StringProperty points;

    public Game(String id, String points) {
        this.id = new SimpleStringProperty(id);
        this.points = new SimpleStringProperty(points);
    }

    public String getId() {
        return id.get();
    }
```

```
    public StringProperty idProperty() {
        return id;
    }

    public void setId(String id) {
        this.id.set(id);
    }

    public String getPoints() {
        return points.get();
    }

    public StringProperty pointsProperty() {
        return points;
    }

    public void setPoints(String points) {
        this.points.set(points);
    }

    public String getName() {
        return "Игра №" + getId();
    }
}
```

Question.java

```
package sample.model;

import com.google.gson.annotations.Expose;
import com.google.gson.annotations.SerializedName;
import javafx.beans.property.SimpleStringProperty;
import javafx.beans.property.StringProperty;

import java.util.List;

public class Question {

    private StringProperty id;
    private StringProperty level;
    private StringProperty text;
    private StringProperty firstAnswer;
    private StringProperty secondAnswer;
    private StringProperty thirdAnswer;
    private StringProperty fourthAnswer;
    private StringProperty correctAnswer;

    public Question(String id, String level, String text, String firstAnswer,
String secondAnswer, String thirdAnswer, String fourthAnswer, String
correctAnswer) {
        this.id = new SimpleStringProperty(id);
        this.level = new SimpleStringProperty(level);
        this.text = new SimpleStringProperty(text);
        this.firstAnswer = new SimpleStringProperty(firstAnswer);
        this.secondAnswer = new SimpleStringProperty(secondAnswer);
        this.thirdAnswer = new SimpleStringProperty(thirdAnswer);
        this.fourthAnswer = new SimpleStringProperty(fourthAnswer);
        this.correctAnswer = new SimpleStringProperty(correctAnswer);
    }
```

```
public String getId() {
    return id.get();
}

public StringProperty idProperty() {
    return id;
}

public void setId(String id) {
    this.id.set(id);
}

public String getLevel() {
    return level.get();
}

public StringProperty levelProperty() {
    return level;
}

public void setLevel(String level) {
    this.level.set(level);
}

public String getText() {
    return text.get();
}

public StringProperty textProperty() {
    return text;
}

public void setText(String text) {
    this.text.set(text);
}

public String getFirstAnswer() {
    return firstAnswer.get();
}

public StringProperty firstAnswerProperty() {
    return firstAnswer;
}

public void setFirstAnswer(String firstAnswer) {
    this.firstAnswer.set(firstAnswer);
}

public String getSecondAnswer() {
    return secondAnswer.get();
}

public StringProperty secondAnswerProperty() {
    return secondAnswer;
}

public void setSecondAnswer(String secondAnswer) {
    this.secondAnswer.set(secondAnswer);
}

public String getThirdAnswer() {
    return thirdAnswer.get();
}
```

```
    public StringProperty thirdAnswerProperty() {
        return thirdAnswer;
    }

    public void setThirdAnswer(String thirdAnswer) {
        this.thirdAnswer.set(thirdAnswer);
    }

    public String getFourthAnswer() {
        return fourthAnswer.get();
    }

    public StringProperty fourthAnswerProperty() {
        return fourthAnswer;
    }

    public void setFourthAnswer(String fourthAnswer) {
        this.fourthAnswer.set(fourthAnswer);
    }

    public String getCorrectAnswer() {
        return correctAnswer.get();
    }

    public StringProperty correctAnswerProperty() {
        return correctAnswer;
    }

    public void setCorrectAnswer(String correctAnswer) {
        this.correctAnswer.set(correctAnswer);
    }
}
```

GameQuestion.java

```
package sample.model;
import javafx.beans.property.SimpleStringProperty;
import javafx.beans.property.StringProperty;

public class GameQuestion {
    private StringProperty id;
    private StringProperty game_id;
    private StringProperty question_id;

    public GameQuestion(String id, String game_id, String question_id) {
        this.id = new SimpleStringProperty(id);
        this.game_id = new SimpleStringProperty(game_id);
        this.question_id = new SimpleStringProperty(question_id);
    }

    public String getId() {
        return id.get();
    }

    public StringProperty idProperty() {
        return id;
    }

    public void setId(String id) {
        this.id.set(id);
    }
```

```
    public String getGame_id() {
        return game_id.get();
    }

    public StringProperty game_idProperty() {
        return game_id;
    }

    public void setGame_id(String game_id) {
        this.game_id.set(game_id);
    }

    public String getQuestion_id() {
        return question_id.get();
    }

    public StringProperty question_idProperty() {
        return question_id;
    }

    public void setQuestion_id(String question_id) {
        this.question_id.set(question_id);
    }
}
```

Main.java

```
package sample;

import javafx.application.Application;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;
import sample.api.MyApi;
import sample.controller.MainMenuViewController;
import javafx.scene.layout.AnchorPane;
import sample.model.Game;
import sample.model.GameQuestion;
import sample.model.Question;

import java.util.List;

public class Main extends Application {

    private ObservableList<Game> games = FXCollections.observableArrayList();
    private ObservableList<Question> questions =
FXCollections.observableArrayList();
    private ObservableList<GameQuestion> gameQuestions =
FXCollections.observableArrayList();
    private MyApi api = new MyApi();

    public ObservableList<Game> getGames() {
        return games;
    }

    public void setGames(ObservableList<Game> games) {
        this.games = games;
    }
```

```
    public void setGames() {
        this.games.addAll(api.getGames());
    }

    public ObservableList<Question> getQuestions() {
        return questions;
    }

    public void setQuestions(ObservableList<Question> questions) {
        this.questions = questions;
    }

    public ObservableList<GameQuestion> getGameQuestions() {
        return gameQuestions;
    }

    public void setGameQuestions(ObservableList<GameQuestion> gameQuestions) {
        this.gameQuestions = gameQuestions;
    }



    public Stage getPrimaryStage() {
        return primaryStage;
    }

    public void setPrimaryStage(Stage primaryStage) {
        this.primaryStage = primaryStage;
    }

    private Stage primaryStage;

    public Main(){
        this.games.addAll(api.getGames());
        this.questions.addAll(api.getQuestions());
        this.gameQuestions.addAll(api.getGameQuestions());
    }

    @Override
    public void start(Stage primaryStage) throws Exception{
        setPrimaryStage(primaryStage);
        FXMLLoader loader = new FXMLLoader();
        loader.setLocation(Main.class.getResource("view/MainMenuView.fxml"));
        AnchorPane mainMenu = loader.load();

        Scene scene = new Scene(mainMenu);
        primaryStage.setScene(scene);
        primaryStage.show();

        MainMenuViewController mainMenuViewController = loader.getController();
        mainMenuViewController.setMainApp(this);
        mainMenuViewController.setRootLayout(mainMenu);
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

MainMenuViewController.java

```
package sample.controller;
```

```
import javafx.collections.ObservableList;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.layout.AnchorPane;
import javafx.scene.layout.BorderPane;
import sample.Main;
import sample.model.Game;

import java.io.IOException;
import java.util.List;

public class MainMenuViewController {

    private Main mainApp;
    private AnchorPane rootLayout;

    public void setMainApp(Main mainApp) {
        this.mainApp = mainApp;
    }

    public void setRootLayout(AnchorPane rootLayout) {
        this.rootLayout = rootLayout;
    }


    @FXML
    public void handleAboutMe() throws IOException {
        FXMLLoader loader = new FXMLLoader();
        System.out.println(Main.class.getResource("view/AboutMeView.fxml"));
        loader.setLocation(Main.class.getResource("view/AboutMeView.fxml"));

        AnchorPane aboutMe = loader.load();

        Scene scene = new Scene(aboutMe);
        mainApp.getPrimaryStage().setScene(scene);
        mainApp.getPrimaryStage().show();

        AboutMeViewController aboutMeViewController = loader.getController();
        aboutMeViewController.setRootLayout(aboutMe);
        aboutMeViewController.setMainApp(mainApp);
    }

    @FXML
    public void handleGamesStats() throws IOException {
        FXMLLoader loader = new FXMLLoader();
        System.out.println(Main.class.getResource("view/GamesStatView.fxml"));
        loader.setLocation(Main.class.getResource("view/GamesStatView.fxml"));

        AnchorPane gamesStats = loader.load();

        Scene scene = new Scene(gamesStats);
        mainApp.getPrimaryStage().setScene(scene);
        mainApp.getPrimaryStage().show();

        GamesStatViewController gamesStatViewController =
loader.getController();
        gamesStatViewController.setRootLayout(gamesStats);
        gamesStatViewController.setMainApp(mainApp);
    }

    @FXML
    public void handleStartGame() throws IOException {
        FXMLLoader loader = new FXMLLoader();
```

```
            System.out.println(Main.class.getResource("view/QuestionView.fxml"));
            loader.setLocation(Main.class.getResource("view/QuestionView.fxml"));

            AnchorPane newGame = loader.load();

            Scene scene = new Scene(newGame);
            mainApp.getPrimaryStage().setScene(scene);
            mainApp.getPrimaryStage().show();

            QuestionViewController questionViewController = loader.getController();
            questionViewController.setRootLayout(newGame);
            questionViewController.setMainApp(mainApp);
        }

}
```

QuestionViewController.java

```
package sample.controller;

import javafx.collections.ObservableList;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.layout.AnchorPane;
import sample.Main;
import sample.api.MyApi;
import sample.model.Game;
import sample.model.Question;

import java.io.IOException;
import java.util.ArrayList;

public class QuestionViewController {
    private Main mainApp;
    private AnchorPane rootLayout;
    private Integer level;
    private Integer points;
    private ObservableList<Question> questions;
    private MyApi api = new MyApi();
    private String currentCorrect;

    @FXML
    private Label pointsLabel;

    @FXML
    private Label questionLabel;

    @FXML
    private Button buttonA;
    @FXML
    private Button buttonB;
    @FXML
    private Button buttonC;
    @FXML
    private Button buttonD;


    public void setRootLayout(AnchorPane rootLayout) {
        this.rootLayout = rootLayout;
    }
```

```
public void setMainApp(Main mainApp) {
    this.mainApp = mainApp;
    this.questions = mainApp.getQuestions();
    System.out.println(questions.get(0).getLevel());
    this.pointsLabel.setText("Баллы: " + this.points.toString());
    nextQuestion();
}

@FXML
public void initialize(){
    this.level = 1;
    this.points = 0;
}

public Question getQuestion(){
    ArrayList<Question> levelQuestions = new ArrayList<>();
    for (Question question : questions){
        if (question.getLevel().equals(this.level.toString())) {
            levelQuestions.add(question);
        }
    }
    int a = (int) (Math.random() * (levelQuestions.size()) );
    return levelQuestions.get(a);
}

public void nextQuestion(){
    Question currentQuestion = getQuestion();
    questionLabel.setText(currentQuestion.getText());
    buttonA.setText("A:" + currentQuestion.getFirstAnswer());
    buttonB.setText("B:" + currentQuestion.getSecondAnswer());
    buttonC.setText("C:" + currentQuestion.getThirdAnswer());
    buttonD.setText("D:" + currentQuestion.getFourthAnswer());
    this.currentCorrect = currentQuestion.getCorrectAnswer();

}

public void resultScreen(boolean win, Integer points) throws IOException {
    this.api.createGame(this.points.toString());
    FXMLLoader loader = new FXMLLoader();
    System.out.println(Main.class.getResource("view/ResultView.fxml"));
    loader.setLocation(Main.class.getResource("view/ResultView.fxml"));

    AnchorPane result = loader.load();

    Scene scene = new Scene(result);
    mainApp.getPrimaryStage().setScene(scene);
    mainApp.getPrimaryStage().show();

    ResultViewController resultViewController = loader.getController();
    resultViewController.setRootLayout(result);
    resultViewController.setMainApp(mainApp, win, this.points);
}

@FXML
public void checkFirstAnswer() throws IOException {
    if (this.currentCorrect.equals("1")){
        this.points += this.level;
        this.level += 1;
        this.pointsLabel.setText("Баллы: " + this.points.toString());
        if (level <= 10) {
            nextQuestion();
        } else {
            resultScreen(true, this.points);
```

```
            }
        } else {
            resultScreen(false, this.points);
        }
    }

    @FXML
    public void checkSecondAnswer() throws IOException {
        if (this.currentCorrect.equals("2")){
            this.points += this.level;
            this.level += 1;
            this.pointsLabel.setText("Баллы: " + this.points.toString());
            if (level <= 10) {
                nextQuestion();
            } else {
                resultScreen(true, this.points);
            }
        } else {
            resultScreen(false, this.points);
        }
    }

    @FXML
    public void checkThirdAnswer() throws IOException {
        if (this.currentCorrect.equals("3")){
            this.points += this.level;
            this.level += 1;
            this.pointsLabel.setText("Баллы: " + this.points.toString());
            if (level <= 10) {
                nextQuestion();
            } else {
                resultScreen(true, this.points);
            }
        } else {
            resultScreen(false, this.points);
        }
    }

    @FXML
    public void checkFourthAnswer() throws IOException {
        if (this.currentCorrect.equals("4")){
            this.points += this.level;
            this.level += 1;
            this.pointsLabel.setText("Баллы: " + this.points.toString());
            if (level <= 10) {
                nextQuestion();
            } else {
                resultScreen(true, this.points);
            }
        } else {
            resultScreen(false, this.points);
        }
    }

}
```

GamesStatViewController.java

```
package sample.controller;

import javafx.beans.property.SimpleStringProperty;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
```

```
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.layout.AnchorPane;
import sample.Main;
import sample.model.Game;

import java.io.IOException;

public class GamesStatViewController {

    private Main mainApp;
    private AnchorPane rootLayout;

    public void setRootLayout(AnchorPane rootLayout) {
        this.rootLayout = rootLayout;
    }

    @FXML
    private TableView<Game> gamesTableView;
    @FXML
    private TableColumn<Game, String> gameColumn;
    @FXML
    private Label gameLabel;
    @FXML
    private Label pointsLabel;

    public void setMainApp(Main mainApp) {
        this.mainApp = mainApp;
        mainApp.setGames();
        gamesTableView.setItems(mainApp.getGames());
    }

    @FXML
    private void initialize() {
        gameColumn.setCellValueFactory(
                c -> new SimpleStringProperty(
                        c.getValue().getName()
                )
        );

        showGameDetails(null);
        gamesTableView.getSelectionModel().selectedItemProperty().addListener(
                ((observableValue, oldValue, newValue) ->
showGameDetails(newValue))

        );
    }

    private void showGameDetails(Game game) {
        if (game != null) {
            gameLabel.setText(game.getName());
            pointsLabel.setText(game.getPoints());

        }
    }

    @FXML
    public void closeScene() throws IOException {
        FXMLLoader loader = new FXMLLoader();
        System.out.println(Main.class.getResource("view/MainMenuView.fxml"));
        loader.setLocation(Main.class.getResource("view/MainMenuView.fxml"));
```

```
            AnchorPane mainMenu = loader.load();

            Scene scene = new Scene(mainMenu);
            mainApp.getPrimaryStage().setScene(scene);
            mainApp.getPrimaryStage().show();

            MainMenuViewController mainMenuViewController = loader.getController();
            mainMenuViewController.setRootLayout(mainMenu);
            mainMenuViewController.setMainApp(mainApp);
        }
    }
```