

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет

вычислительной техники

Кафедра

МОиПЭВМ

Направление подготовки 09.03.02 «Информационные системы и технологии»

Направленность Информационные системы и технологии

БАКАЛАВРСКАЯ РАБОТА

на тему

**Разработка информационной системы по
организации профориентационной работы
школьников в игровой форме**

Студент

_____ Керимов Ильяс Якубович
(подпись, дата) (ФИО полностью)

Руководитель

_____ Попова Н. А.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр

_____ Такташкин Д.В.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой

_____ Козлов А.Ю.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК

_____ Попова Н.А.
(подпись) (фамилия, инициалы)

Пенза, 2025

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
Вычислительной техники

Кафедра
МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____ А.Ю. Козлов
«__» _____ 2025 г

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА

1. Студент Керимов Ильяс Якубович гр. 21ВИ1 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Разработка информационной системы по организации
профориентационной работы школьников в игровой форме

Тема утверждена приказом ПГУ № _____ от "___" _____ 202_ г.

3. Руководитель работы к.т.н., доцент кафедры «МО и ПЭВМ» Попова Н. А.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

1) *Назначение: разработка информационной системы по организации*
профориентационной работы школьников в игровой форме

2) *Среда выполнения: Windows*

Языковые средства разработки: JavaScript, Java

Инструментальные средства: IntelliJIDEA, VSCode, PostgreSQL, Postman.

3) *Основные функции разрабатываемого программного обеспечения:*

а) войти и зарегистрироваться;

б) пройти тестирование;

в) получить рекомендации по профессии на основе тестирования;

г) сохранение результатов тестирования в личном кабинете;

д) просмотр подробной информации о рекомендуемой профессии;

е) юмористические и визуальные элементы для создания эффекта геймификации;

5. Перечень подлежащих разработке вопросов:

1) Анализ предметной области и постановка задачи;
2) План разработки бюджета;
3) Разработка требований и проектирование;
4) Реализация;
5) Контроль качества программного обеспечения;
6) Оформление пояснительной записки.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	08.04.2025 – 14.04.2025	
2	Проектирование программных средств	20%	15.04.2025 – 25.04.2025	
3	Реализация программных средств	30%	26.04.2025 – 12.05.2025	
4	Тестирование программных средств	20%	13.05.2024 – 22.05.2024	
5	Оформление пояснительной записки	20%	23.05.2025 – 03.06.2025	

Дата выдачи задания «20» декабря 2024г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан ФВТ _____ Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 86 листов, 27 рисунков, 11 таблиц, 12 использованных источников и 2 приложения.

ПРОФОРИЕНТАЦИОННОЕ ТЕСТИРОВАНИЕ, ИГРОВАЯ ФОРМА, КЛИЕНТ- СЕРВЕРНАЯ АРХИТЕКТУРА, MVC, JAVA, JAVASCRIPT, POSTGRESQL, UML, WEB- ПРИЛОЖЕНИЕ, VISUAL STUDIO CODE, INTELLIJIDEA.

Объект исследования – информационная система по организации профориентационной работы школьников в игровой форме на основе клиент – серверной архитектуры.

Целью выпускной квалификационной работы является разработка информационной системы по организации профориентационной работы школьников в игровой форме на основе клиент – серверной архитектуры.

Технологии разработки – среды разработки IntelliJIDEA и Visual Studio Code, языки программирования Java и JavaScript, СУБД PostgreSQL.

Среда исполнения – Windows 10, web-браузер.

Результаты работы – по итогу выполнения выпускной квалификационной работы разработана информационная система по организации профориентационной работы школьников в игровой форме на основе клиент – серверной архитектуры.

Все требования к разработке приложения соблюдены.

					ПГУ 09.03.02 – 08BP202.09 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Разработка информационной системы по организации профориентационной работы школьников в игровой форме Пояснительная записка	Лит.	Лист	Листов
Разраб.		Керимов И. Я.					4	86
Пров.		Попова Н.А.						
Н. Контр.		Такташкин Д.В				Группа 21ВИ1		
Утв.		Козлов А.Ю						

Содержание

Введение.....	7
1 Анализ информационной системы по организации профориентационной работы.....	8
1.1 Анализ предметной области.....	8
1.2 Анализ аналогов систем профориентации для школьников	9
1.3 Анализ требований к профориентационной системе	11
1.4 Постановка задачи.....	16
2 Проектирование информационной системы по организации профориентационной работы.....	18
2.1 Выбор технологий проектирования	18
2.2 Проектирование архитектуры информационной системы по организации профориентационной работы	18
2.3 Проектирование графического интерфейса.....	24
2.4 Оценка бюджета разработки информационной системы.....	29
3 Разработка и реализация информационной системы по организации профориентационной работы школьников	34
3.1 Обоснование выбора средств и технологий разработки.....	34
3.2 Разработка серверной части информационной системы по организации профориентационной работы школьников	35
3.3 Разработка слоя представления приложения	41
4 Тестирование программных средств.....	49
4.1 Тестирование белым ящиком.....	49
4.1.1 Общие сведения	49
4.1.2 Покрытие операторов.....	49
4.1.3 Покрытие решений	50

4.1.4 Покрытие маршрутов	51
4.2 Автоматизированное тестирование.....	52
4.3 Метрика SLOC	56
4.4 Метрика Холстеда	56
Заключение	59
Список использованных источников.....	60
Приложение А (обязательное) Листинг приложения.....	62
Приложение Б (рекомендуемое) Описание вариантов использования	85

Введение

Информационные технологии прочно вошли во все сферы человеческой деятельности, и их влияние с каждым годом только возрастает. Современные информационные системы играют ключевую роль в образовании, здравоохранении, промышленности, бизнесе и других направлениях. Одним из важных направлений их применения становится поддержка и развитие образования, в том числе таких его аспектов, как профориентация.

Профориентационная работа с учащимися – это важнейший этап в формировании осознанного профессионального выбора. От того, насколько эффективно школьник сможет определить свои интересы, склонности и способности, зависит его успешность в будущем образовании и карьере. Однако традиционные методы профориентации нередко оказываются недостаточно мотивирующими и не всегда учитывают интересы современной молодёжи, особенно в условиях стремительного развития цифровой среды.

Интеграция игровых технологий в образовательный процесс позволяет решать сразу несколько задач: повысить вовлечённость учащихся, сделать процесс получения знаний более интересным и эффективным, а также адаптировать его под актуальные интересы школьников. Игровая форма подачи материала стимулирует исследовательскую активность, развивает критическое мышление и помогает глубже усвоить информацию. Именно поэтому создание информационной системы для профориентационной работы в игровой форме представляет собой актуальное и перспективное направление.

Данная дипломная работа посвящена разработке подобной системы, сочетающей в себе возможности информационных технологий и игровые механики для повышения эффективности профориентации среди школьников.

1 Анализ информационной системы по организации профориентационной работы

1.1 Анализ предметной области

Предметная область данной работы – это профориентационная деятельность среди школьников, с применением информационных технологий и игровых методов. Основная цель таких систем – помочь учащимся определить свои интересы, способности и подходящие профессии.

Профориентация как направление в образовании начала активно развиваться в XX веке. Изначально она была частью школьной педагогики и предполагала беседы с учащимися, консультации с психологами и использование бумажных анкет. Позднее стали применяться более научные методы – например, тесты на тип личности, интересы и склонности [1].

С развитием компьютеров и интернета начали появляться электронные тесты и онлайн-опросники, которые частично автоматизировали профориентационный процесс. Однако долгое время такие решения были скучными и малоинтерактивными.

Сегодня профориентация по-прежнему есть в школьной системе, но в большинстве случаев она представлена в виде разовых мероприятий: тестирование, встреча с представителями профессий, консультация с педагогом. При этом большинство школьников воспринимает эти форматы как неинтересные или непонятные, особенно в средней школе (5–8 классы), когда интерес к будущей профессии только начинает формироваться.

Таким образом, предметная область нуждается в новых подходах, сочетающих в себе информационные технологии, психологические методы и элементы игры. Разработка системы, в которой профориентация будет подана в удобной, визуально понятной и игровой форме, отвечает современным требованиям образования и может занять свою нишу в школьной практике

1.2 Анализ аналогов систем профориентации для школьников

В ходе разработки системы по организации профориентации школьников в игровой форме был проведен анализ наиболее близких аналогов данной системы. Далее приведен их обзор.

Сервис «Навигатум»: проект предлагает разнообразные игровые инструменты для профессионального и личностного самоопределения. Среди них – компьютерные игры, такие как «Работа на каникулах», обучающие трудовому законодательству, и «Профи плюс», знакомящие с профессиями и их типами. На рисунке 1 представлен начальный экран данного сервиса.

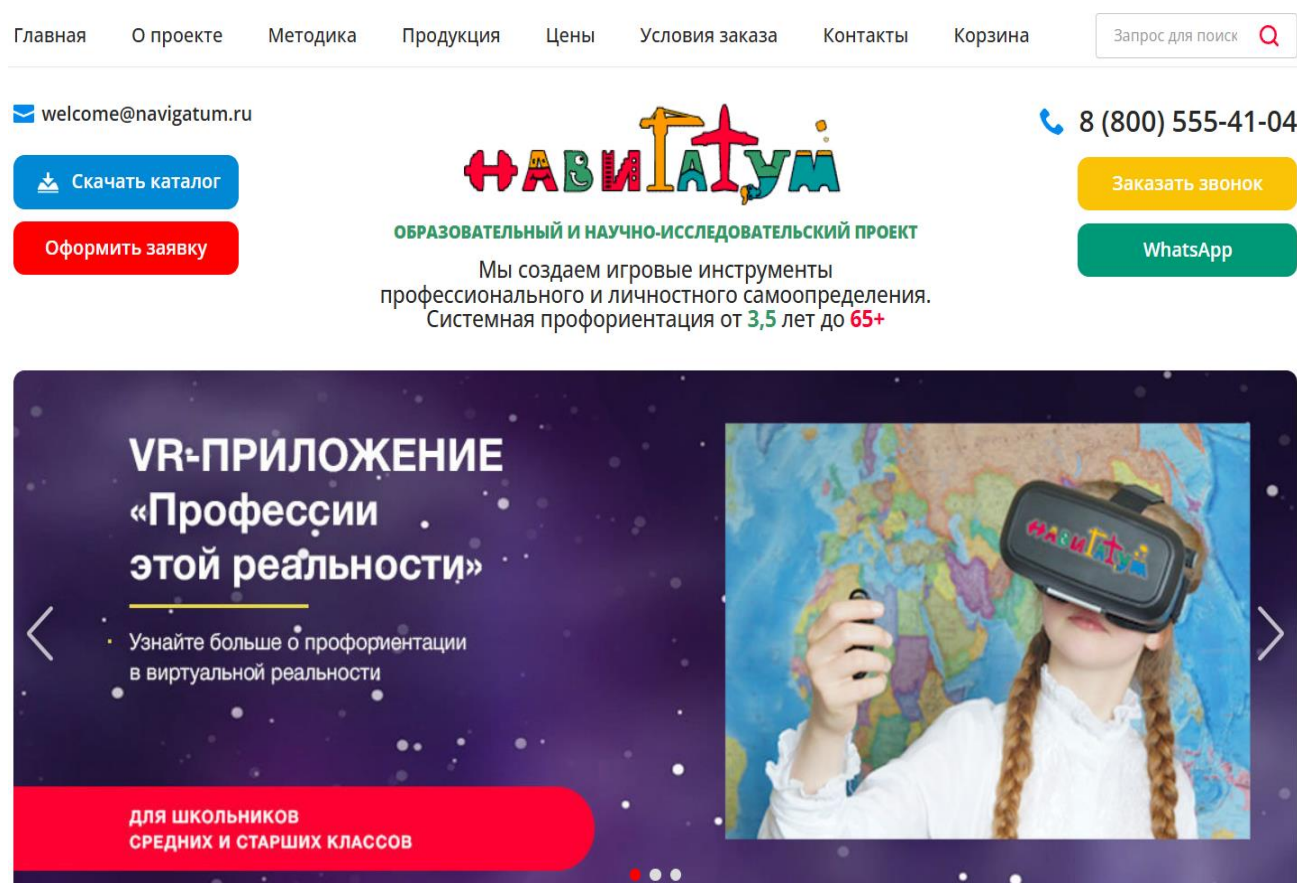


Рисунок 1 – Начальный экран сервиса «Навигатум»

Сервис имеет следующие недостатки:

- перегруженность сервиса;
- коммерческая модель доступа.

Еще одним приложением для контроля успеваемости учащихся является

сайт IPGS (Интегральное игровое профилирование). Вместо традиционных вопросов используются изображения и фантастическая игровая реальность, что делает процесс более увлекательным. Начальный экран «IPGS» представлен на рисунке 2.

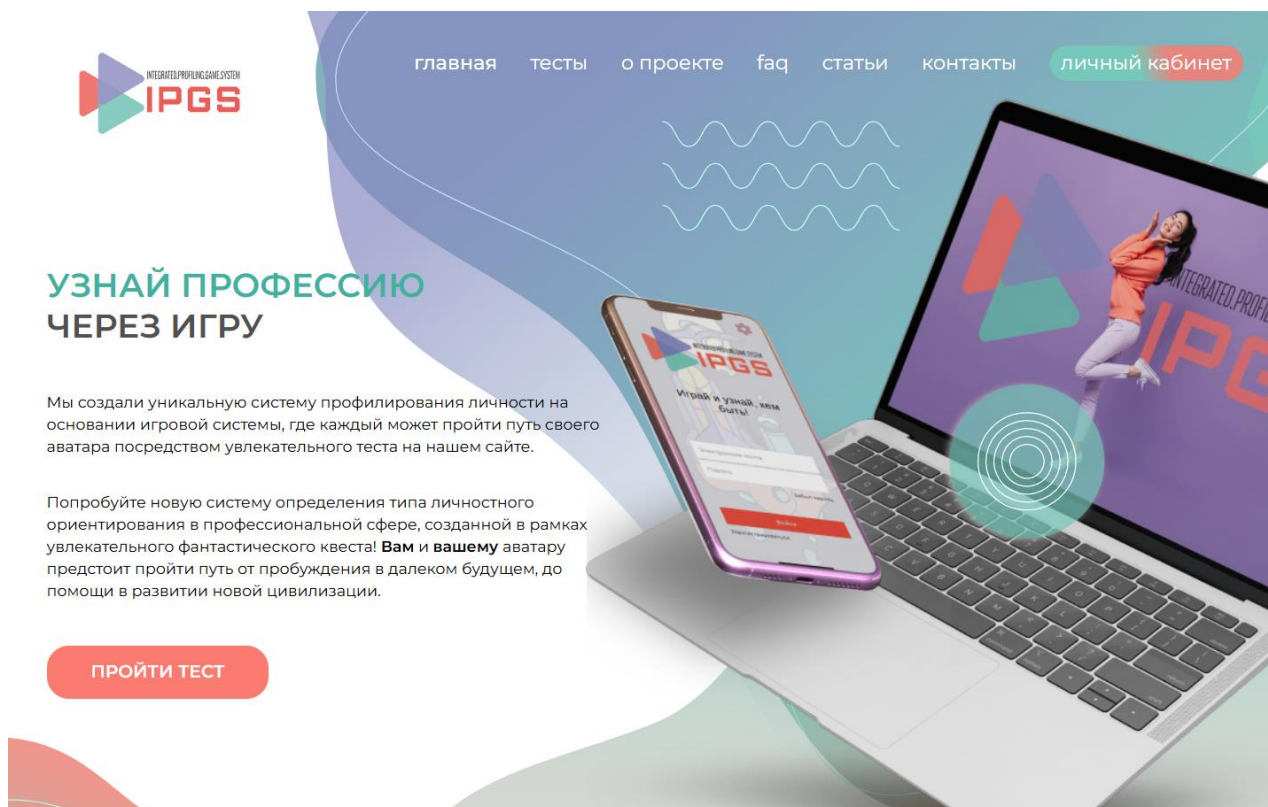


Рисунок 2 – Начальный экран «IPGS»

Сервис имеет следующие недостатки:

- сложность восприятия результатов (в виде таблиц и графиков);
- отсутствует сохранение результатов тестирования;
- нет подробной информации о способе получения рекомендуемой профессии.

В целом, для большинства аналогичных сервисов характерны такие минусы, как коммерческий доступ, перегруженность интерфейса и отсутствие подробной информации о способах получения рекомендуемой профессии. Разрабатываемая информационная система будет лишена данных недостатков.

1.3 Анализ требований к профориентационной системе

В приложении для пользователя должна быть возможность:

- a) зарегистрироваться в системе;
- b) выполнить вход в личный кабинет с использованием логина (email) и пароля;
- c) пройти тестирование, состоящее из 10 вопросов;
- d) выбрать один из четырех вариантов ответа на каждый вопрос;
- e) по завершении теста получить рекомендацию профессии на основе выбранных ответов;
- f) сохранить результаты тестирования в системе;
- g) просматривать историю всех пройденных тестов с указанием:
 - 1) даты;
 - 2) номера попытки;
 - 3) баллов по направлениям (логика, аналитика, креативность);
 - 4) финальной рекомендации;
- h) просматривать значки/бейджи, соответствующие рекомендованным профессиям;
- i) перейти на страницу информации о профессии, где указаны:
 - 1) 5 вузов, в которых можно получить образование по выбранному направлению;
 - 2) проходной балл на бюджет;
 - 3) средняя стоимость обучения за год;
 - 4) средняя заработная плата выпускника;
- j) использовать кнопку «Мне повезёт», открывающую шуточную рекомендацию;
- k) вернуться на главную страницу из любой части приложения.

Регистрация пользователя должна происходить на отдельном экране, на котором необходимо ввести email и пароль. После успешной регистрации пользователь автоматически попадает на главную страницу.

Авторизация осуществляется на специальном экране входа. Для доступа в

систему пользователь должен ввести корректные данные: email и пароль. При успешной авторизации отображается главная страница с доступом к тестированию и личному кабинету. В шапке сайта отображаются кнопки «Личный кабинет» и «Выход», заменяющие «Вход» и «Регистрация».

На главной странице пользователь видит две основные кнопки: «Начать тестирование» и «Мне повезёт». Кнопка «Мне повезёт» открывает отдельный экран с шуточной рекомендацией профессии – «Миллиардер», что добавляет элемент геймификации.

Тестирование начинается с показа первого вопроса. Вопросы загружаются динамически из базы данных. Пользователь видит только один вопрос за раз и выбирает один из четырёх вариантов ответа. После выбора доступна кнопка «Следующий вопрос». На десятом вопросе появляется кнопка «Завершить тестирование».

В личном кабинете пользователь может просматривать историю прохождения тестов. Для каждой попытки отображаются: номер попытки, дата, баллы по категориям, рекомендованная профессия и соответствующий значок. При клике на значок профессии пользователь переходит на страницу с информацией о 5 российских вузах, подходящих для обучения по данному направлению.

Диаграмма вариантов использования (или Use Case Diagram) – это один из видов диаграмм в языке моделирования UML (Unified Modeling Language), который используется для отображения взаимодействия пользователей (акторов) с системой через различные сценарии использования (варианты использования) [2].

На основе требований, которые были сформулированы выше, была разработана диаграмма вариантов использования для пользователя профориентационной платформы, представленная на рисунке 3.

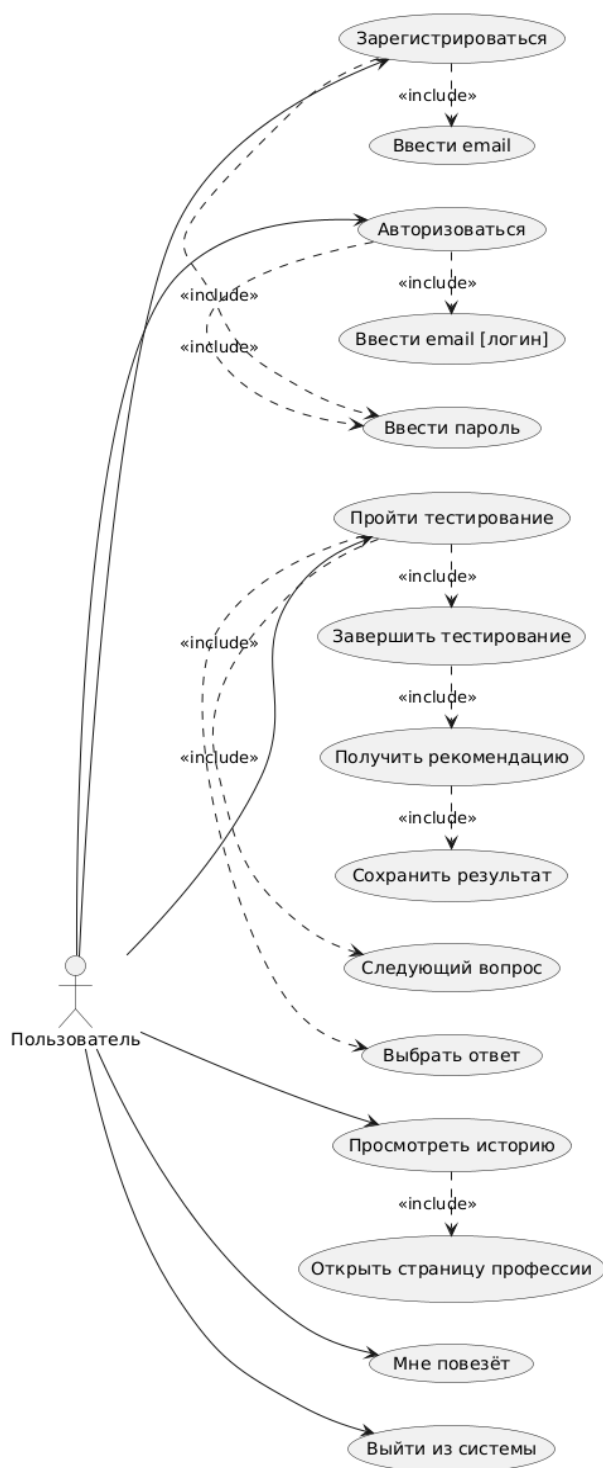


Рисунок 3 – Диаграмма вариантов использования пользователя платформы

Далее описаны основные варианты использования приложения для актора «Пользователь приложения».

В таблице 1 проведено описание варианта использования «Пройти тестирование».

Таблица 1 – Вариант использования «Пройти тестирование»

Вариант использования	Пройти тестирование
ID	1
Краткое описание	Пользователь проходит тестирование, состоящее из 10 вопросов по 4 варианта ответа в каждом для определения будущей профессии
Главные актёры	Пользователь
Второстепенные актёры	Нет
Предусловия	Нет
Основной поток	<ol style="list-style-type: none"> 1. Пользователь начинает тестирование 2. Пользователь читает вопрос в тесте 3. Пользователь выбирает один из четырех вариантов ответа 4. Пользователь переходит к следующему вопросу
Постусловия	Пользователь прошел тестирование

Данная таблица отображает список последовательности действий пользователя, которые ему необходимо совершить, чтобы пройти тестирование.

Таблица 2 – Вариант использования «Получить рекомендацию профессии»

Вариант использования	Получить рекомендацию профессии
ID	2
Краткое описание	Пользователь получает рекомендацию по будущей профессии на основе своих навыков
Главные актёры	Пользователь
Второстепенные актёры	Нет
Предусловия	Пользователь прошел тестирование
Основной поток	<ol style="list-style-type: none"> 1. Пользователь завершает тестирование 2. Пользователь получает рекомендацию по будущей профессии на основе одного из вариантов, наиболее ему подходящих
Постусловия	Пользователь получил рекомендацию по профессии

В данном описании вариантов использования приложения отображен порядок действий, совершаемых пользователем для получения рекомендации по профессии.

Таблица 3 – Вариант использования «Просмотреть историю результатов прохождения тестирования»

Вариант использования	Просмотреть историю результатов прохождения тестирования
ID	3
Краткое описание	Пользователь просматривает историю своих попыток прохождения теста

Продолжение таблицы 3 – Вариант использования «Просмотреть историю результатов прохождения тестирования»

Вариант использования	Просмотреть историю результатов прохождения тестирования
Главные актёры	Пользователь
Второстепенные актёры	Нет
Предусловия	Нет
Основной поток	<ol style="list-style-type: none"> 1. Пользователь переходит на главную страницу 2. Пользователь нажимает кнопку «личный кабинет» в правом верхнем углу. 3. Пользователь видит пронумерованные результаты своих попыток прохождения тестирования
Постусловия	Пользователь просмотрел историю результатов прохождения тестирования

Данная таблица показывает набор действий, совершаемых пользователем для просмотра истории результатов тестирования.

Таблица 4 – Вариант использования «Перейти на страницу профессии»

Вариант использования	Перейти на страницу профессии
ИД	4
Краткое описание	Пользователь переходит на страницу предложенной по итогам тестирования профессии
Главные актёры	Пользователь
Второстепенные актёры	Нет
Предусловия	Пользователь находится в личном кабинете
Основной поток	<ol style="list-style-type: none"> 1. Пользователь просматривает варианты предложенных профессий по итогам каждого из тестирований 2. Пользователь нажимает на название профессии 3. Пользователь переходит на страницу профессии, где указаны ВУЗы для ее получения и т. д.
Постусловия	Пользователь просмотрел информацию о своей профессии

Данный пример использования приложения показывает набор действий, совершаемых пользователем для того, чтобы посмотреть подробное описание рекомендуемой ему после прохождения тестирования профессии.

Выше были предоставлены основные виды использования вариантов web – приложения профориентационного тестирования, необходимые для его проектирования. С остальными вариантами можно ознакомиться в приложении А.

Для корректной и эффективной работы системы профориентационного

тестирования не предъявляются высокие требования к аппаратно-техническому обеспечению.

Для использования системы необходим современный браузер (Google Chrome, Mozilla Firefox, Яндекс и др.) и стабильное интернет-соединение. Backend-сервер может быть развернут на машине с минимальной конфигурацией:

- двухъядерный процессор (1.5+ ГГц);
- не менее 2 ГБ оперативной памяти;
- поддержка Java 17+;
- установленная PostgreSQL;
- Node.js и npm/yarn на фронте для разработки.

Таким образом, в ходе анализа были определены функциональные и нефункциональные требования, выделены и описаны основные варианты использования, которые необходимо реализовать в рамках разработки текстовой ролевой игры.

1.4 Постановка задачи

В результате анализа предметной области была определена цель разработать информационную систему по организации профориентационной работы школьников в игровой форме. Система должна тестировать пользователя на его склонность к различным профессиям, выдавать результат в зависимости от его предпочтений, сохранять полученный результат, а также иметь различные элементы геймификации.

Для достижения поставленной цели требуется решить следующие задачи:

- выполнить анализ предметной области;
- выполнить анализ аналогов;
- определить требования к разрабатываемой информационной системе;
- спроектировать архитектуру системы;
- выполнить оценку бюджета разработки;

- реализовать спроектированное приложение;
- протестировать разработанное приложение.

В результате выполнения данных задач должен быть выполнен анализ проведённой работы и сделаны выводы о достижении поставленной цели.

2 Проектирование информационной системы по организации профорientационной работы

2.1 Выбор технологий проектирования

При выборе инструментов для проектирования системы основной акцент был сделан на простоту, наглядность и отсутствие финансовых ограничений. Визуализация структуры приложения, сценариев взаимодействия и этапов разработки велась с помощью таких онлайн-инструментов, как Visual Paradigm, Draw.io и Online Gantt. Каждый из них оказался удобен по-своему: Visual Paradigm – привычный инструмент для построения UML-диаграмм, Draw.io – за счёт гибкого интерфейса и быстрого доступа, а Online Gantt – как эффективное средство планирования задач во времени.

В качестве языка моделирования использовалась нотация UML, поскольку она давно зарекомендовала себя как универсальный и понятный способ представления архитектуры и логики систем [3].

Для макетирования пользовательского интерфейса был использован FigJam – упрощённая версия Figma, идеально подходящая для быстрой визуализации интерфейсных решений. Он предоставляет ровно столько инструментов, сколько нужно для продумывания логики экранов, не отвлекая избыточным функционалом.

Ключевым преимуществом всех выбранных средств стала их бесплатность и доступность в вебе, что позволило сосредоточиться не на инструментах, а на содержании проекта.

2.2 Проектирование архитектуры информационной системы по организации профорientационной работы

Разрабатываемое приложение будет построено по принципу клиент-серверной архитектуры, что обеспечивает чёткое разделение пользовательского интерфейса и бизнес-логики [4].

Клиентская часть будет реализована с использованием библиотеки React

и представляет собой одностраничное веб-приложение (SPA), отвечающее за отображение экранов, обработку действий пользователя, маршрутизацию и взаимодействие с сервером через REST API.

Серверная часть будет построена на платформе Spring Boot и реализует всю прикладную логику: управление пользователями, обработку ответов на тест, подсчёт баллов и формирование рекомендаций.

Все данные будут храниться в реляционной базе данных PostgreSQL, с которой сервер взаимодействует через ORM-библиотеку Hibernate. Такое решение обеспечивает масштабируемость, расширяемость и удобство в сопровождении приложения. Кроме того, использование REST-архитектуры позволяет при необходимости разработать мобильную версию клиента или интеграцию с другими сервисами без изменения серверной логики.

Рассмотрим отдельно клиентскую и серверную часть приложения для того, чтобы детальнее исследовать их основные составляющие, а также взаимосвязь данных частей между собой.

Для того, чтобы наиболее удобным образом визуализировать архитектурные составляющие, необходимо использовать диаграмму компонентов.

Диаграмма компонентов является важным средством визуализации архитектуры программного обеспечения и позволяет наглядно представить, из каких основных блоков состоит система и как эти блоки взаимодействуют между собой. Она фокусируется на логической структуре приложения, отображая ключевые компоненты (например, клиент, сервер, база данных, API) и связи между ними.

Полезность диаграммы компонентов заключается в том, что она помогает лучше понять модули и зависимости внутри системы. Особенно это актуально в проектах с разделением на фронтенд и бэкенд, как в данном случае. С помощью такой диаграммы можно отразить, какие части кода отвечают за обработку данных, какие – за представление информации пользователю, а какие – за хранение данных. Это облегчает анализ, сопровождение и расширение

приложения. Диаграмма компонентов клиентской части представлена на рисунке 4.

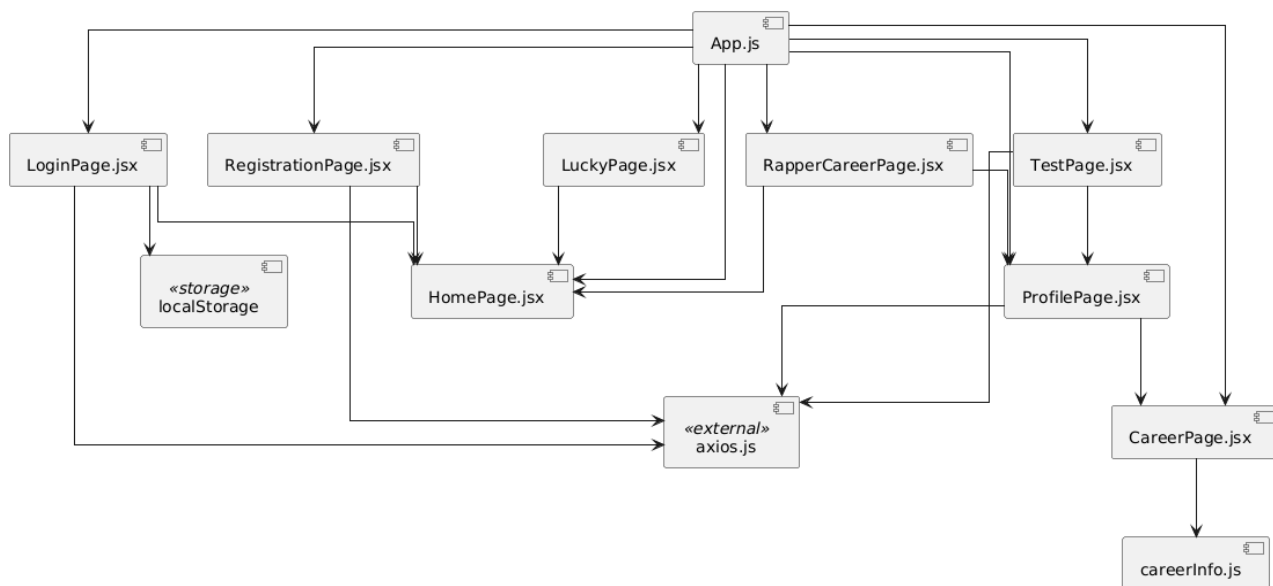


Рисунок 4 – Диаграмма компонентов клиентской части

Эта диаграмма отражает структуру и взаимодействие основных JavaScript- и JSX-файлов проекта.

Перейдем к разбору серверной части проекта. Для отображения ее основных структурных частей снова отлично подойдет диаграмма компонентов. На рисунке 5 представлена диаграмма компонентов серверной части профориентационного сервиса.

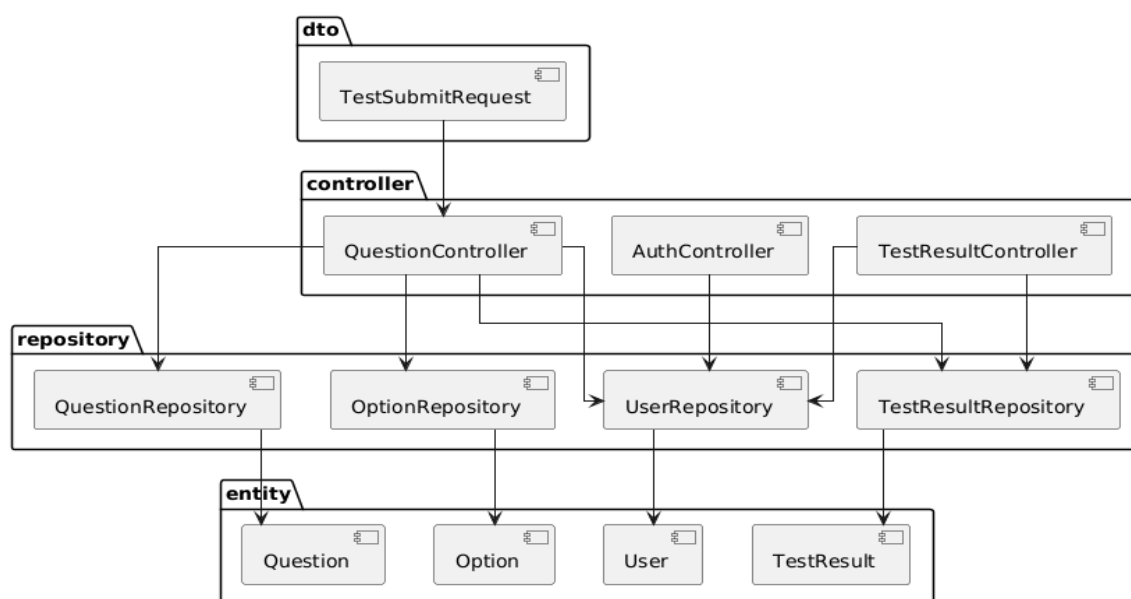


Рисунок 5 – Диаграмма компонентов серверной части

Диаграмма компонентов серверной части информационной системы профориентации отражает взаимодействие между ключевыми модулями, сгруппированными в соответствии с архитектурным подходом MVC (Model–View–Controller). Контроллеры AuthController, QuestionController и TestResultController принимают HTTP-запросы от клиента и направляют их к соответствующим слоям. Они взаимодействуют с репозиториями, такими как UserRepository, QuestionRepository, OptionRepository и TestResultRepository, для извлечения и записи данных. Модели User, Question, Option и TestResult представляют сущности, соответствующие таблицам базы данных, обеспечивая связь с хранимыми данными.

Компоненты DTO (TestSubmitRequest) используются для структурированной передачи данных между клиентом и сервером, в частности при отправке результатов теста.

Помимо компонентов клиентской и серверной части профориентационной системы была спроектирована логическая схема база данных, представленная на рисунке 6.

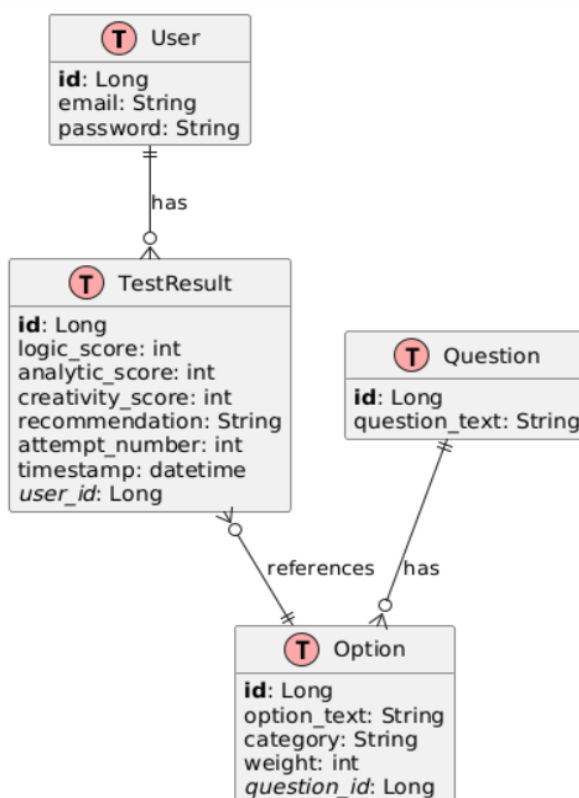


Рисунок 6 – Логическая схема базы данных

Данная схема отображает основные сущности приложения, через которые и реализуются механизм профориентационной деятельности для подбора профессии для конечного пользователя. Речь идет о хранении данных пользователя, результатов тестирований, а также вопросов и вариантов ответа для них.

Далее перейдем к описанию того, какие основные действия пользователь может совершать в приложении. Удобнее всего визуально это можно отобразить через диаграмму деятельности.

Диаграмма деятельности представляет собой один из видов UML-диаграмм, предназначенный для описания логики выполнения процессов в системе [5]. Она визуализирует последовательность действий, которые выполняются в рамках определённого сценария, включая разветвления, условия, параллельные потоки и взаимодействие между различными участниками. В отличие от диаграммы последовательности, диаграмма деятельности делает акцент не на передаче сообщений, а на описании потока управления и переходов между действиями, что делает её особенно полезной для моделирования бизнес-логики и пользовательских сценариев.

В контексте разрабатываемого приложения диаграмма деятельности отражает процесс взаимодействия между пользователем, клиентской и серверной частями системы. Ключевыми этапами в нашем случае являются: регистрация пользователя, авторизация, запуск тестирования, прохождение теста, получение и отображение результатов, а также просмотр истории и рекомендованной профессии в личном кабинете. Диаграмма чётко разделена по ролям (пользователь – клиент – сервер), что позволяет отследить, какие действия выполняются на каждом уровне и как между ними происходит обмен данными. Это позволяет формализовать и упростить понимание логики работы приложения как для разработчиков, так и для заинтересованных сторон. Данная диаграмма представлена на рисунке 7.

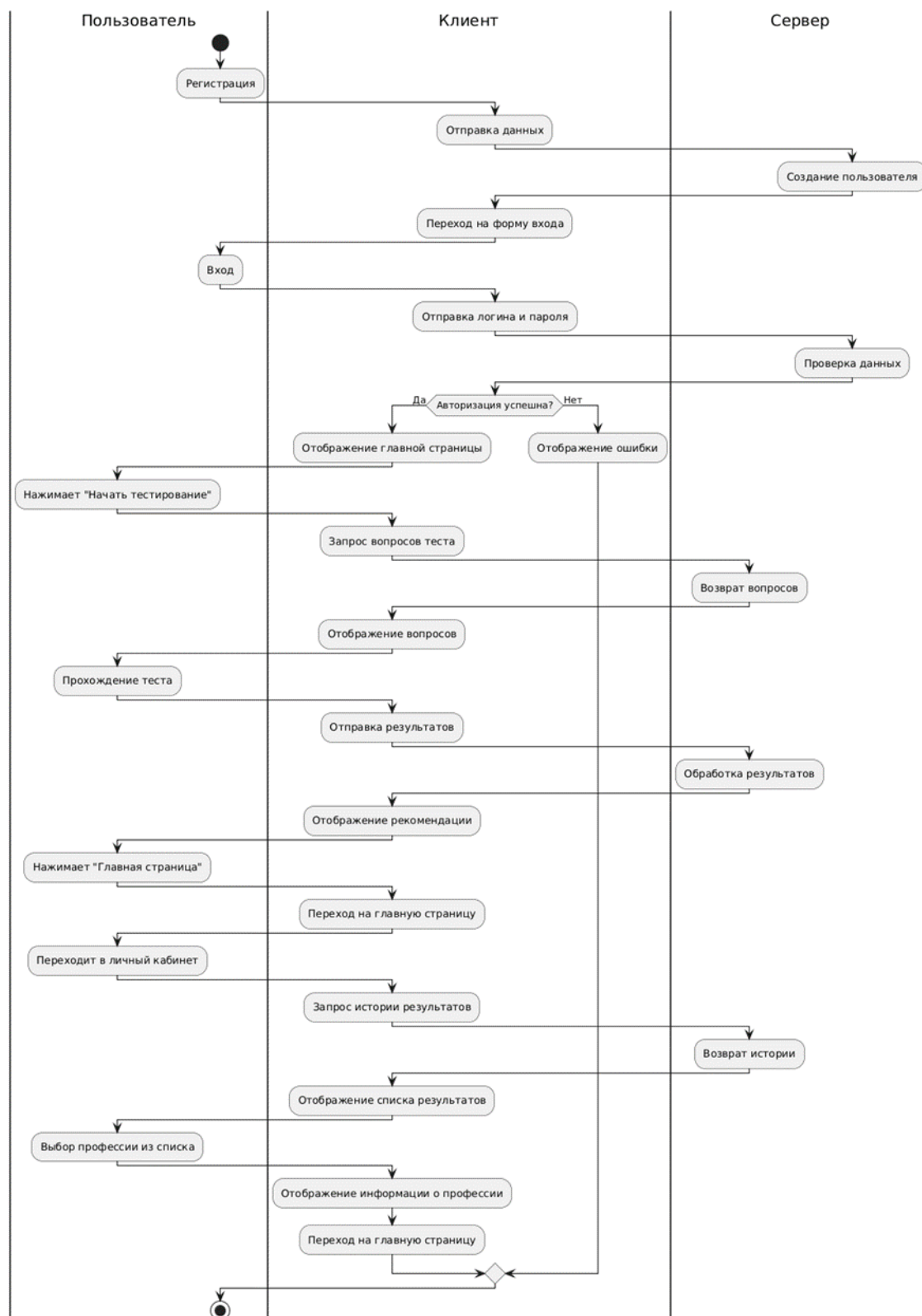


Рисунок 7 – Диаграмма деятельности алгоритма использования приложения

Таким образом, для разработки информационной системы профориентации была выбрана клиент-серверная архитектура, которая наилучшим образом соответствует задачам приложения. Данный тип

архитектуры обеспечивает достаточную гибкость и масштабируемость при относительно невысокой сложности реализации, что особенно актуально для учебных и прикладных проектов с ограниченным числом функций и пользователей.

В отличие от микросервисной архитектуры, клиент-серверная модель позволяет упростить развертывание, сопровождение и тестирование приложения, избегая избыточного распределения по сервисам [6]. Микросервисный подход актуален преимущественно для высоконагруженных систем со множеством изолированных модулей и командами разработки, тогда как в данной системе все модули тесно связаны, а функциональность централизована. Это делает клиент-серверную архитектуру наиболее рациональным и эффективным решением для реализации профориентационного веб-приложения.

2.3 Проектирование графического интерфейса

Проектирование графического интерфейса пользователя (GUI) является одним из ключевых этапов создания web-приложения, особенно в случае систем, ориентированных на широкую аудиторию, включая школьников, как в данном проекте. От качества интерфейса напрямую зависит удобство взаимодействия с системой, вовлечённость пользователя и общее впечатление от работы с приложением. Особенно в условиях игровой формы подачи материала важно не только обеспечить функциональность, но и создать интуитивно понятную, привлекательную и дружелюбную визуальную среду.

Для эффективного проектирования GUI необходимо придерживаться ряда фундаментальных принципов [7]:

- принцип понятности – заключается в минимизации когнитивной нагрузки на пользователя и обеспечении логичной структуры экранов;
- принцип единообразия – предполагает использование унифицированных элементов управления и стилей во всем интерфейсе;

– принцип доступности – обеспечивает комфортную работу для всех категорий пользователей, включая тех, кто может иметь ограничения в восприятии информации.

Следование этим принципам позволяет разработать интерфейс, который не только выполняет свою функцию, но и способствует мотивации пользователя проходить тестирование и взаимодействовать с системой.

В результате проектирования были созданы следующие макеты экранов.

На рисунке 8 изображен макет окна регистрации, в котором пользователь может осуществить регистрацию в сервисе.

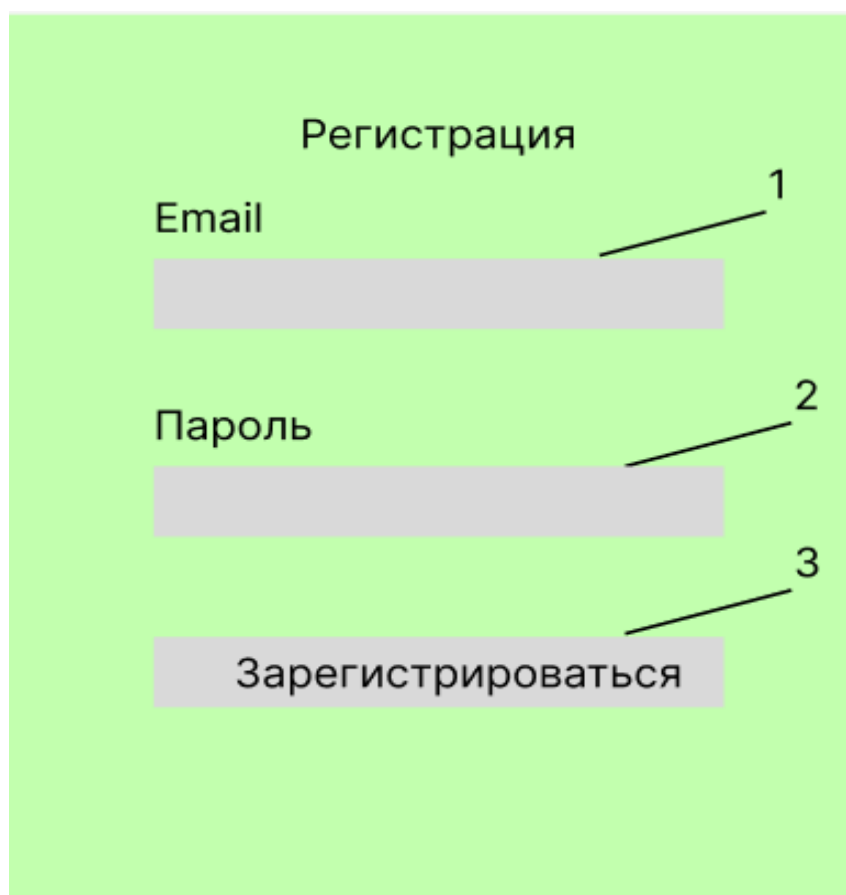
Макет окна регистрации на светло-зеленом фоне. В центре расположены три элемента: текстовое поле для Email, текстовое поле для Пароля и кнопка «Зарегистрироваться». Каждый элемент имеет указательный номер: 1 для Email, 2 для Пароля и 3 для кнопки. Все элементы имеют серый фон и черную обводку. Над элементами, выровненными по центру, находится заголовок «Регистрация».

Рисунок 8 – Макет окна регистрации

Макет окна регистрации содержит следующие элементы:

- 1) – поле для ввода email(логина);
- 2) – поле для ввода пароля;
- 3) – кнопка «Зарегистрироваться», после нажатия на которую происходит регистрация и переход на окно авторизации.

Вход

Email 1

Пароль 2

Войти 3

Рисунок 9 – Макет окна авторизации

На рисунке 9 изображен макет окна авторизации, в котором пользователь может войти в систему.

Макет окна авторизации содержит следующие элементы:

- 1) – поле для ввода email(логина);
- 2) – поле для ввода пароля;
- 3) – кнопка «Войти», после нажатия на которую происходит аутентификация и переход на главный экран сервиса.

На рисунке 10 изображен макет главной страницы web – портала профорientационной работы школьников.

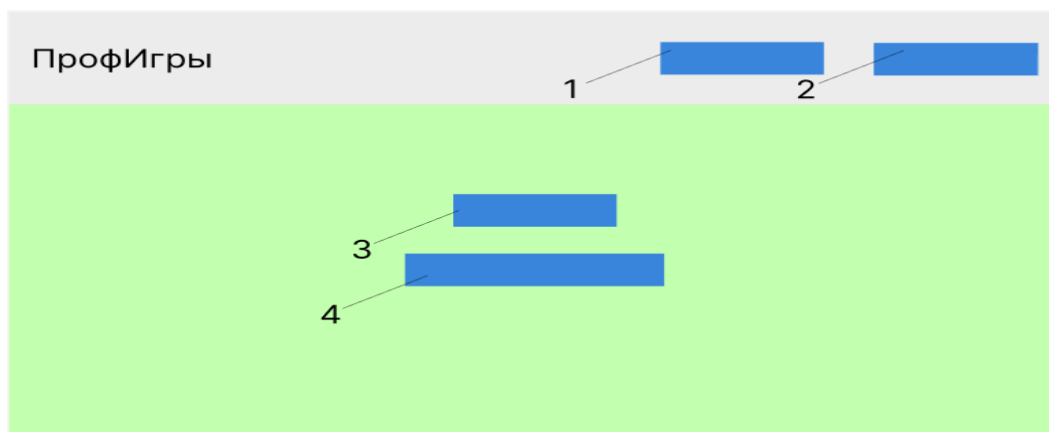


Рисунок 10 – Макет главной страницы

Макет главной страницы содержит следующие элементы:

- 1) – кнопка «Личный кабинет»;
- 2) – кнопка «Выйти»;
- 3) – кнопка «Мне повезёт»;
- 4) – кнопка «Начать тестирование»;

На рисунке 11 изображен макет страницы тестирования, на которой пользователь должен ответить на десять вопросов для того, чтобы узнать, какую профессию ему порекомендует сервис.



Рисунок 11 – Макет страницы тестирования

Макет окна страницы тестирования содержит следующие элементы:

- – заголовок окна «Профориентационный тест»;
- – окно с вопросом и выбором одного из четырех вариантов ответа;
- – кнопка «Следующий вопрос».

На рисунке 12 изображен макет страницы «Личный кабинет».

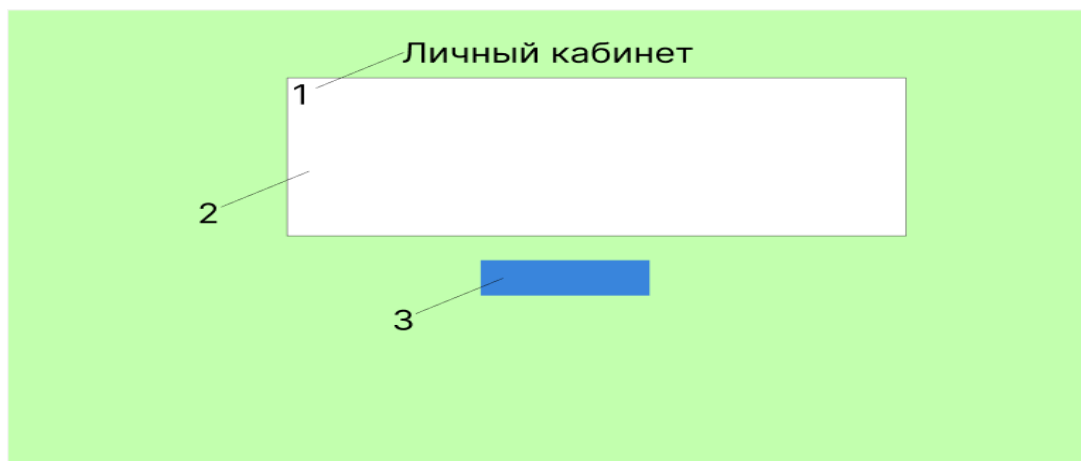


Рисунок 12 – Макет страницы «Личный кабинет»

Макет страницы «Личный кабинет» содержит следующие элементы:

- 1) – заголовок окна «Личный кабинет»;
- 2) – окно с результатами прохождений тестирований;
- 3) – кнопка «На главную».

На рисунке 13 изображен макет страницы с описанием профессии.

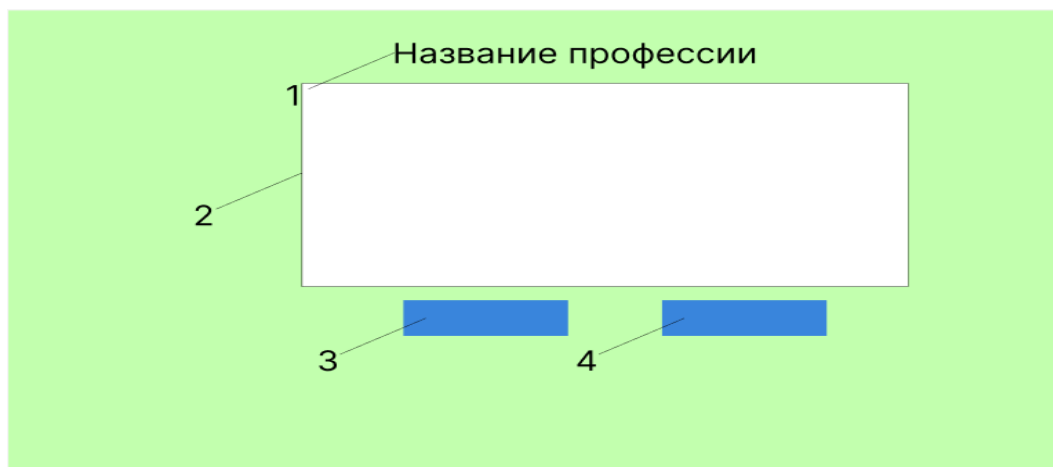


Рисунок 13 – Макет страницы с описанием профессии

Макет страницы с описанием профессии содержит следующие элементы:

- 1) – название профессии, рекомендуемой пользователю;
- 2) – характеристика профессии (рекомендуемые ВУЗы и т. д.);
- 3) – кнопка «В личный кабинет»;
- 4) – кнопка «На главную»;

Таким образом, были спроектированы все окна web – портала профориентационной работы школьников в игровой форме. На основе данных макетов будет разработан пользовательский интерфейс.

2.4 Оценка бюджета разработки информационной системы

Оценка бюджета разработки информационной системы является важным этапом проектирования, так как она позволяет заранее определить объём необходимых ресурсов, обеспечить реалистичное планирование проекта и избежать перерасхода времени и средств [8]. Финансовая оценка даёт возможность соотнести цели и задачи разработки с доступными возможностями, а также аргументированно обосновать затраты перед заказчиком или инвестором. Кроме того, понимание бюджета помогает принимать управленческие решения, связанные с выбором технологий, организацией команды и масштабом функционала.

Рассмотрим основные факторы, влияющие на размер бюджета при разработке приложения:

- объём и сложность функциональности: чем больше возможностей реализуется в системе, тем выше трудозатраты;
- уровень квалификации команды;
- технические решения: выбор архитектуры, используемые технологии и платформа;
- продолжительность разработки;
- расходы на хостинг, безопасность, тестирование и поддержку.

Так как влияние данных факторов варьируется от проекта к проекту в зависимости от того, насколько большую роль играет каждый фактор в конкретном разрабатываемом проекте, то и итоговую оценку бюджета для каждого проекта нужно производить индивидуально.

В таблице 5 показано описание ключевых задач проекта, которые требуется решить для успешной разработки информационной системы профориентации школьников, а также определены их последовательность и

заложено время на выполнение каждой задачи.

Таблица 5 – Список задач проекта

№	Название задачи	Длительность
1	Начало проекта	-
2	Анализ информационных систем профориентации	5 дня
3	Анализ предметной области «Информационные системы для организации профориентации школьников»	2 дня
4	Обзор аналогичных информационных систем профориентации школьников в игровой форме	3 дня
5	Анализ требований к системам профориентации школьников	2 дня
6	Постановка задачи	1 день
7	Проектирование системы по организации профориентации школьников	-
8	Проектирование архитектуры системы по организации профориентации для школьников	3 дня
9	Проектирование графического интерфейса	1 день
10	Оценка бюджета системы по организации профориентации для школьников	1 день
11	Проектирование системы по организации профориентации для школьников завершено	-
12	Разработка и реализация системы по организации профориентации для школьников	-
13	Разработка слоя модели системы по организации профориентации для школьников	5 дней
14	Разработка слоя представления системы по организации профориентации для школьников	3 дня
15	Разработка слоя контроллера системы по организации профориентации для школьников	7 дней
16	Разработка интерфейса системы по организации профориентации для школьников	4 дня
17	Разработка и реализация системы по организации профориентации для школьников	-
18	Тестирование программных средств	-

Продолжение таблицы 5 – Список задач проекта

№	Название задачи	Длительность
19	Тестирование белым ящиком	2 дня
20	Модульное тестирование	2 дня
21	Метрика SLOC	1 день
22	Метрика Холстеда	1 день
23	Тестирование программных средств завершено	-
24	Сдача проекта	-
25	Составление документации	6 дней
26	Оформление и печать документации	2 дня
27	Сдача проекта завершена	-
28	Конец проекта	-

Результат разбиения проекта на задачи показал, что для реализации потребуется выполнить 28 задачу. В таблице 6 детализированы затрачиваемые ресурсы.

Таблица 6 – Описание затрачиваемых ресурсов

Название ресурса	Тип	Стандартная ставка	Начисление
Аналитик	Трудовой	500 Р/ч	Пропорциональное
Программист	Трудовой	650 Р/ч	Пропорциональное
Дизайнер	Трудовой	450 Р/ч	Пропорциональное
Тестировщик	Трудовой	600 Р/ч	Пропорциональное
Компьютер	Материальный	90 000 Р	В начале
Электричество	Затраты	1000 Р/м	Пропорциональное
Интернет	Затраты	600 Р/м	Пропорциональное

Для визуализации этапов разработки системы по организации профориентации для школьников и оценки ресурсов и времени, которые займет каждый этап, была разработана диаграмма Ганта, представленная на рисунке 14.

На диаграмме отражены ресурсы, основные этапы разработки, последовательность и сроки выполнения каждой задачи. Выполнение задач имеет последовательный характер, так как доступные трудовые ресурсы ограничены, и каждый этап зависит от успешного завершения предыдущего. Это позволяет эффективно организовать рабочий процесс и обеспечить

прогрессивное развитие проекта.

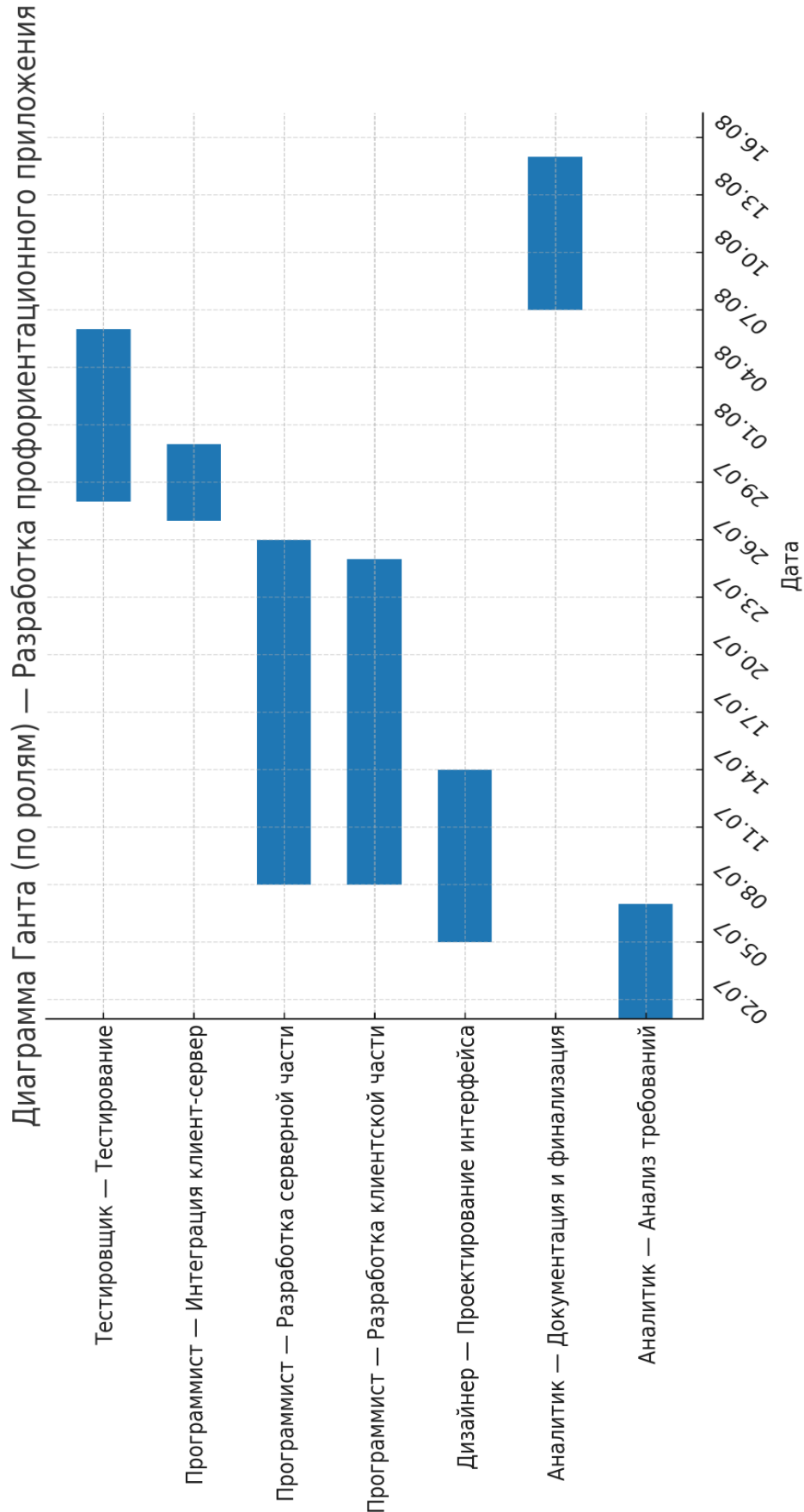


Рисунок 14 – Диаграмма Ганта

После проведения анализа и подготовки отчета о бюджете была вычислена общая стоимость разработки программных средств – 444 400 рублей. 10% были заложены на непредвиденные расходы, итого на разработку необходимо 484 440 рублей.

Общий срок выполнения всех задач, необходимых для разработки, по результатам анализа составил 51 день.

3 Разработка и реализация информационной системы по организации профориентационной работы школьников

3.1 Обоснование выбора средств и технологий разработки

В качестве основного языка программирования для разработки клиентской части приложения был выбран JavaScript с использованием библиотеки React. React предоставляет мощные средства для построения динамических и отзывчивых пользовательских интерфейсов, благодаря компонентному подходу, виртуальному DOM и богатой экосистеме. Этот выбор обусловлен необходимостью создать современное одностраничное приложение (SPA), обеспечивающее быструю и плавную навигацию без перезагрузки страниц, что особенно важно в условиях игровой формы подачи профориентационного контента для школьников.

Для разработки клиентской части использовалась среда программирования Visual Studio Code (VS Code). Этот редактор предоставляет удобный и настраиваемый интерфейс, поддержку подсветки синтаксиса, автодополнение, встроенный терминал, а также расширения для React, Tailwind CSS и работы с Git [9]. Благодаря лёгкости и расширяемости, VS Code является одним из наиболее популярных инструментов среди фронтенд-разработчиков и позволяет эффективно организовать работу над клиентской частью даже в небольших командах.

Серверная часть приложения была реализована на языке Java с использованием фреймворка Spring Boot. Этот выбор обусловлен широкими возможностями Spring Boot в построении REST API, высокой производительностью, масштабируемостью, а также богатой экосистемой, включающей модули для работы с базами данных, безопасности и тестирования. Java в связке со Spring Boot позволяет быстро разрабатывать надёжные серверные приложения с чётким разделением логики и высокой степенью повторного использования кода.

В качестве среды разработки для серверной части была выбрана IntelliJ

IDEA, одна из самых мощных IDE для Java. Она предоставляет расширенную поддержку Spring, удобную навигацию по коду, отладку, рефакторинг и автоматическую генерацию конфигураций. Использование IntelliJ IDEA позволило ускорить процесс разработки и упростить сопровождение серверной части благодаря встроенным инструментам анализа кода и интеграции с системами контроля версий.

Для хранения данных использовалась реляционная система управления базами данных PostgreSQL. Этот выбор обусловлен её надёжностью, поддержкой ACID-транзакций, высокой производительностью и широкими возможностями по работе с типами данных, включая UUID, что удобно для идентификации сущностей. PostgreSQL также хорошо масштабируется и поддерживает расширенные SQL-функции, что делает её подходящим решением как для учебных, так и для промышленных проектов.

3.2 Разработка серверной части информационной системы по организации профориентационной работы школьников

Серверная часть приложения построена на основе архитектурного шаблона MVC (Model – View – Controller), с добавлением слоя DTO (Data Transfer Object) и слоя конфигурации. Такая структура позволяет чётко разделить ответственность между компонентами: контроллеры обрабатывают входящие HTTP-запросы, сервисы реализуют бизнес-логику, модели (сущности) представляют структуру данных, а репозитории обеспечивают доступ к базе данных с помощью Spring Data JPA.

В составе серверной части можно выделить несколько ключевых модулей:

- модельный слой (entity) – включает классы User, Question, Option и TestResult, эти классы описывают структуры таблицы базы данных и определяют взаимосвязи между сущностями, такие как «вопрос имеет множество вариантов» или «пользователь имеет множество результатов тестов»;
- контроллеры (controller) – реализованы в виде классов AuthController, QuestionController, TestResultController, они принимают запросы

от клиента (например, регистрация пользователя или отправка ответов на тест), вызывают соответствующие сервисы и формируют ответы;

- репозитории (repository) – содержат интерфейсы UserRepository, QuestionRepository, OptionRepository, TestResultRepository, которые расширяют стандартные JPA-интерфейсы и предоставляют доступ к данным без необходимости ручного написания SQL-запросов;

- DTO (dto) – используются для переноса данных между клиентом и сервером, пример – класс TestSubmitRequest, который принимает массив ID выбранных пользователем ответов.

В данном проекте используется библиотека Hibernate, которая выполняет ключевую роль как ORM-фреймворк (Object-Relational Mapping), используемый для взаимодействия между объектами Java и таблицами реляционной базы данных PostgreSQL. Hibernate упрощает работу с базой данных, позволяя разработчику работать с сущностями как с обычными Java-объектами, а не писать вручную SQL-запросы для операций CRUD (создание, чтение, обновление, удаление).

С помощью Hibernate можно легко сохранять, извлекать и обновлять данные, при этом он автоматически управляет транзакциями, связями между сущностями (например, @OneToMany, @ManyToOne) и генерацией SQL-кода. Кроме того, Hibernate используется в связке с Spring Data JPA, что позволяет создавать репозитории без реализации методов вручную – достаточно лишь расширить базовый интерфейс JpaRepository.

Далее рассмотрим структуру MVC на примере группы классов, связанных в проекте с вопросами для профориентационного тестирования.

Класс Question в серверной части проекта представляет собой сущность (Entity), связанную с таблицей вопросов в базе данных. Он используется для хранения и обработки данных, относящихся к каждому тестовому вопросу, отображаемому пользователю в процессе прохождения тестирования. Фрагмент кода файла, в котором описаны импорты, зависимости и поля сущности Question, представлен на листинге 1.

Листинг 1 – Фрагмент кода класса Question

```
public class Question {

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

    private String questionText;

    @OneToMany(mappedBy = "question", cascade = CascadeType.ALL,
fetch = FetchType.LAZY)
    private List<Option> options;

    // геттеры и сеттеры
    public Long getId() { return id; }

    public void setId(Long id) { this.id = id; }

    public String getQuestionText() { return questionText; }

    public void setQuestionText(String questionText) {
this.questionText = questionText; }

    public List<Option> getOptions() { return options; }

    public void setOptions(List<Option> options) {
        this.options = options;
        // также можно связать обратно:
        for (Option option : options) {
            option.setQuestion(this);
        }
    }
}
```

Уровень View в данной архитектуре представлен на клиентской части и будет рассмотрен позже.

QuestionController – это REST-контроллер серверной части приложения, отвечающий за обработку HTTP-запросов, связанных с тестированием пользователя. Он играет ключевую роль в организации профориентационного процесса, предоставляя клиенту данные о вопросах и вариантах ответов, а также принимая и обрабатывая результаты теста. Его основными функциями являются получение вопросов и сохранение результатов прохождения теста. Фрагмент кода. Файла, в котором указаны эти методы, представлен на листинге 2.

Листинг 2 – Фрагмент кода класса QuestionController

```

@CrossOrigin(origins = "http://localhost:3000")
@RestController
@RequestMapping("/api")
public class QuestionController {
    @Autowired
    private QuestionRepository questionRepository;
    @Autowired
    private OptionRepository optionRepository;
    @Autowired
    private TestResultRepository testResultRepository;
    @Autowired
    private UserRepository userRepository;

    @GetMapping("/questions")
    public List<Question> getAllQuestions() {
        return questionRepository.findAllWithOptions();
    }

    @PostMapping("/submit-test")
    public String submitTest(@RequestBody TestSubmitRequest
request) {
        List<Long> optionIds = request.getSelectedOptionIds();
        User user =
userRepository.findById(request.getUserId()).orElseThrow();

        Map<String, Integer> scores = new HashMap<>();
        scores.put("logic", 0);
        scores.put("analytics", 0);
        scores.put("creativity", 0);

        for (Long id : optionIds) {
            Option opt =
optionRepository.findById(id).orElse(null);
            if (opt != null && opt.getCategory() != null) {
                scores.put(opt.getCategory(),
scores.get(opt.getCategory()) + opt.getWeight());
            }
        }

        // Простая логика рекомендаций
        int logic = scores.get("logic");
        int analytics = scores.get("analytics");
        int creativity = scores.get("creativity");

        String recommendation;
        if (logic >= 8 || (logic >= 5 && analytics >= 3)) {
            recommendation = "Рекомендуемая профессия:
Программист";
        } else if (analytics >= 5) {
            recommendation = "Рекомендуемая профессия: Финансовый
аналитик";

```

```

    } else if (creativity >= 5) {
        recommendation = "Рекомендуемая профессия: Дизайнер";
    } else {
        recommendation = "Рекомендуемая профессия: Рэпер";
    }

    // сохраняем результат в БД
    TestResult result = new TestResult();
    result.setUser(user);
    result.setLogicScore(logic);
    result.setAnalyticsScore(analytics);
    result.setCreativityScore(creativity);
    result.setRecommendation(recommendation);
    result.setTimestamp(LocalDateTime.now());

    result.setAttemptNumber(testResultRepository.countByUser(user) +
1);

    testResultRepository.save(result);

    return recommendation;
}
}

```

TestSubmitRequest – это DTO-класс (Data Transfer Object), используемый в серверной части приложения для приёма данных от клиента при отправке результатов тестирования. Код класса, в котором указаны поля и методы их получения и изменения, представлен в листинге 3.

Листинг 3 – Код файла TestSubmitRequest

```

@Getter
@Setter
public class TestSubmitRequest {
    private Long userId;
    private List<Long> selectedOptionIds;
}

```

QuestionRepository – это интерфейс в серверной части приложения, который обеспечивает доступ к данным сущности **Question** в базе данных. Он является частью слоя **repository** и использует возможности **Spring Data JPA**, что позволяет автоматически создавать методы для работы с таблицами без написания SQL-запросов вручную. Код класса представлен в листинге 4.

Листинг 4 – Код класса Repository

```
@Repository
public interface QuestionRepository extends
JpaRepository<Question, Long> {
    @Query("SELECT q FROM Question q LEFT JOIN FETCH q.options")
    List<Question> findAllWithOptions();
}
```

Структура взаимодействия модуля вопросов, представленная диаграммой классов, изображена на рисунке 15.

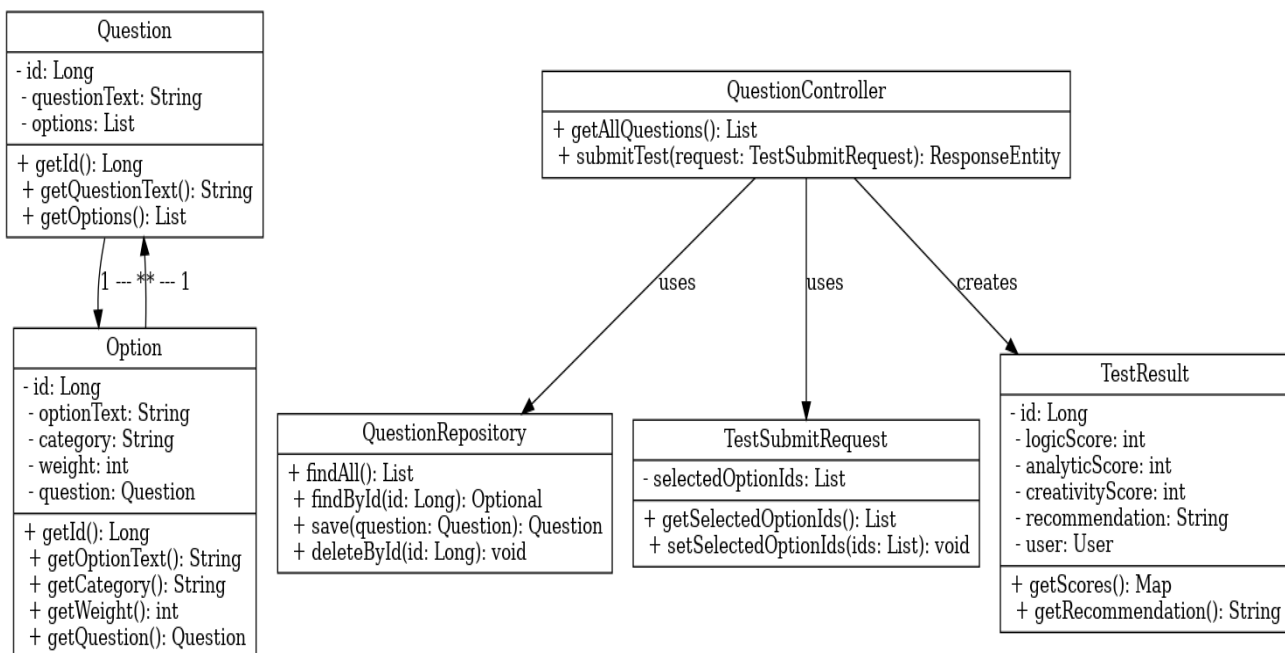


Рисунок 15 – Диаграмма классов модуля вопросов

Код всех файлов серверной части представлен в приложении Б.

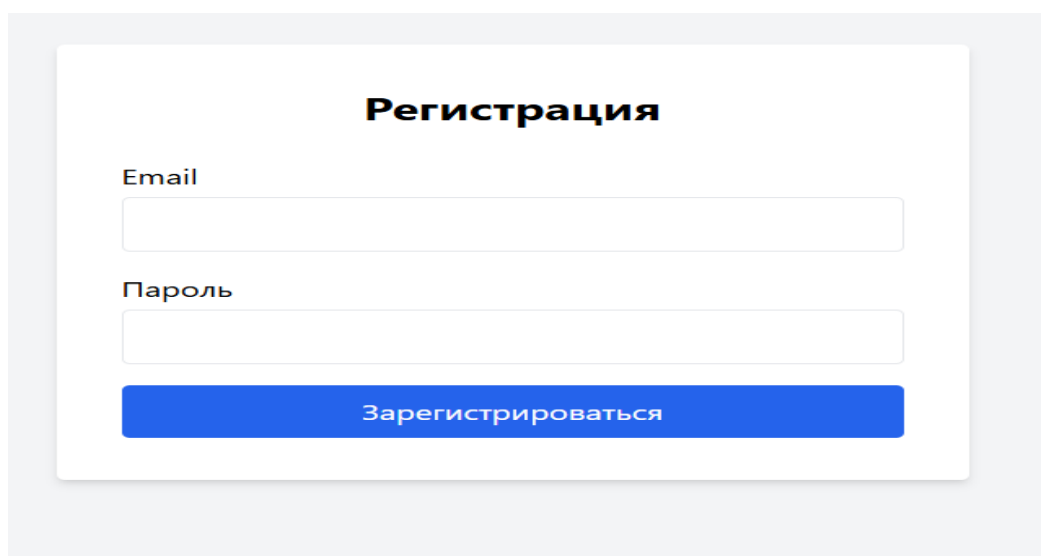
Таким образом, в ходе разработки серверной части была реализована надёжная и расширяемая архитектура на основе шаблона MVC с использованием фреймворка Spring Boot. Были созданы модели данных, обеспечивающие структуру хранимой информации, а также REST-контроллеры, реализующие основные сценарии взаимодействия с клиентом.

Дополнительно внедрены DTO-классы для безопасной передачи данных и репозитории на основе Spring Data JPA для удобного доступа к базе данных. Такая архитектура обеспечивает чёткое разделение логики, удобство тестирования, возможность масштабирования и простоту сопровождения проекта в будущем.

3.3 Разработка слоя представления приложения

Разработка графического интерфейса проводилась на фреймворке React языка JavaScript. React обладает рядом преимуществ, делающих его удобным для создания интерактивных веб-приложений. Он использует компонентный подход, что позволяет переиспользовать интерфейсные элементы, и виртуальный DOM, обеспечивающий высокую производительность при обновлении данных. Кроме того, React легко интегрируется с REST API и имеет широкую экосистему для поддержки маршрутизации, стилизации и управления состоянием [10]. Были разработаны следующие основные страницы: авторизация, регистрация, главная страница, страница тестирования, страница с результатами, личный кабинет.

При открытии сайта пользователь, не прошедший авторизацию, автоматически перенаправляется на страницу входа или может вручную перейти на страницу регистрации. Графический интерфейс страницы регистрации представлен в виде формы с полями для ввода адреса электронной почты и пароля, а также кнопкой подтверждения. После ввода данных и нажатия кнопки «Зарегистрироваться» осуществляется проверка и отправка информации на сервер. В случае успешной регистрации пользователь автоматически попадает на главную страницу приложения. Страница регистрации представлена на рисунке 16.



The image shows a registration form titled "Регистрация" (Registration). It contains two input fields: "Email" and "Пароль" (Password). Below the password field is a blue button labeled "Зарегистрироваться" (Register).

Рисунок 16 – Страница регистрации

Если ранее пользователь уже был зарегистрирован, он может войти в систему через страницу авторизации. Графически эта страница представляет собой простую форму с двумя полями: для ввода email и пароля, а также кнопкой «Войти». В случае успешного ввода данных пользователь перенаправляется на главную страницу приложения. Интерфейс страницы авторизации выполнен в едином стиле с остальными разделами и обеспечивает минималистичный, интуитивно понятный пользовательский опыт. Страница авторизации представлена на рисунке 17.

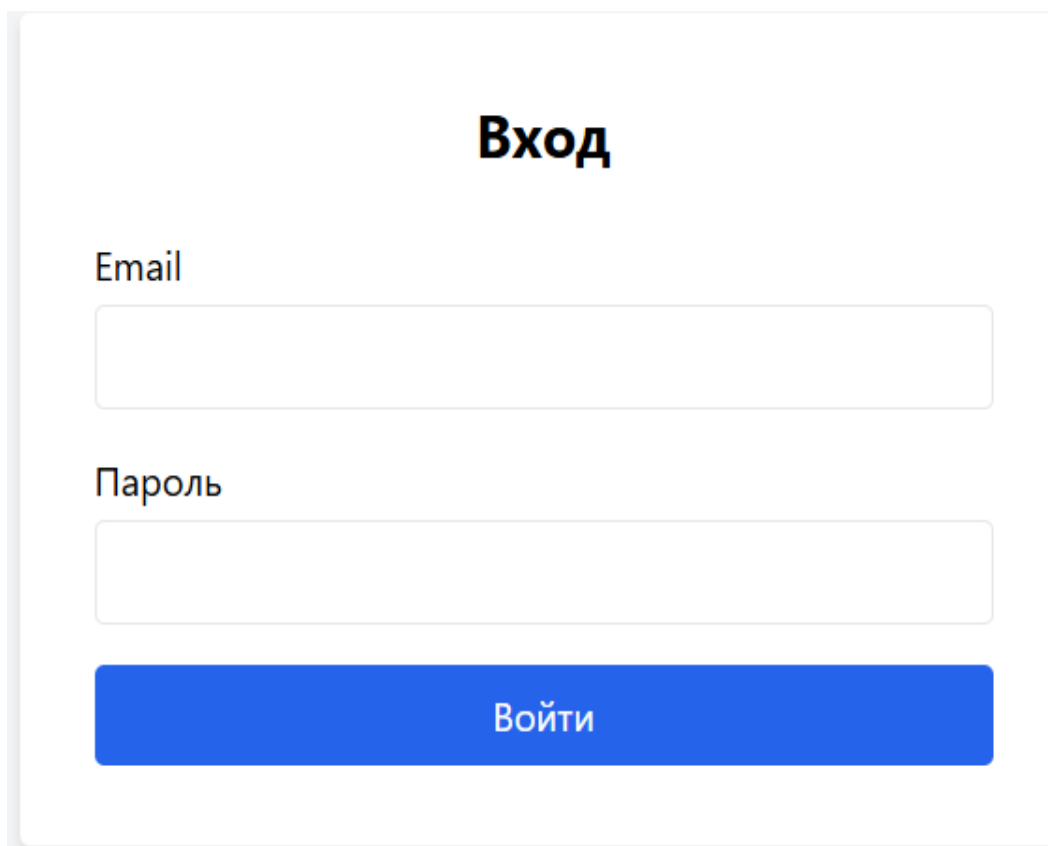
The image shows a login form titled "Вход" (Login) in a bold, black font. Below the title, there are two input fields: the first is labeled "Email" and the second is labeled "Пароль" (Password). Both fields are empty and have a light gray border. Below the password field is a blue button with the text "Войти" (Login) in white. The entire form is enclosed in a light gray border.

Рисунок 17 – Страница авторизации

Если пользователь нажал на кнопку «Войти» и успешно прошёл авторизацию, он попадает на главную страницу приложения, где отображается персонализированная навигация. В шапке страницы для авторизованного пользователя доступны кнопки «Начать тестирование», «Личный кабинет» и «Выход», а сам контент ориентирован на дальнейшее взаимодействие с системой – прохождение теста, просмотр результатов и рекомендации.

Для неавторизованного пользователя главная страница также доступна, но

в урезанном виде. В шапке отображаются кнопки «Войти» и «Зарегистрироваться», а доступ к тестированию и личному кабинету ограничен. Вместо персонализированного контента может выводиться краткая информация о назначении сервиса и призыв зарегистрироваться. Таким образом, визуальное оформление главной страницы адаптируется в зависимости от статуса пользователя, обеспечивая удобную навигацию и логичное разграничение доступа. На рисунке 18 представлена страница для неавторизованного пользователя. На рисунке 19 для авторизованного пользователя.

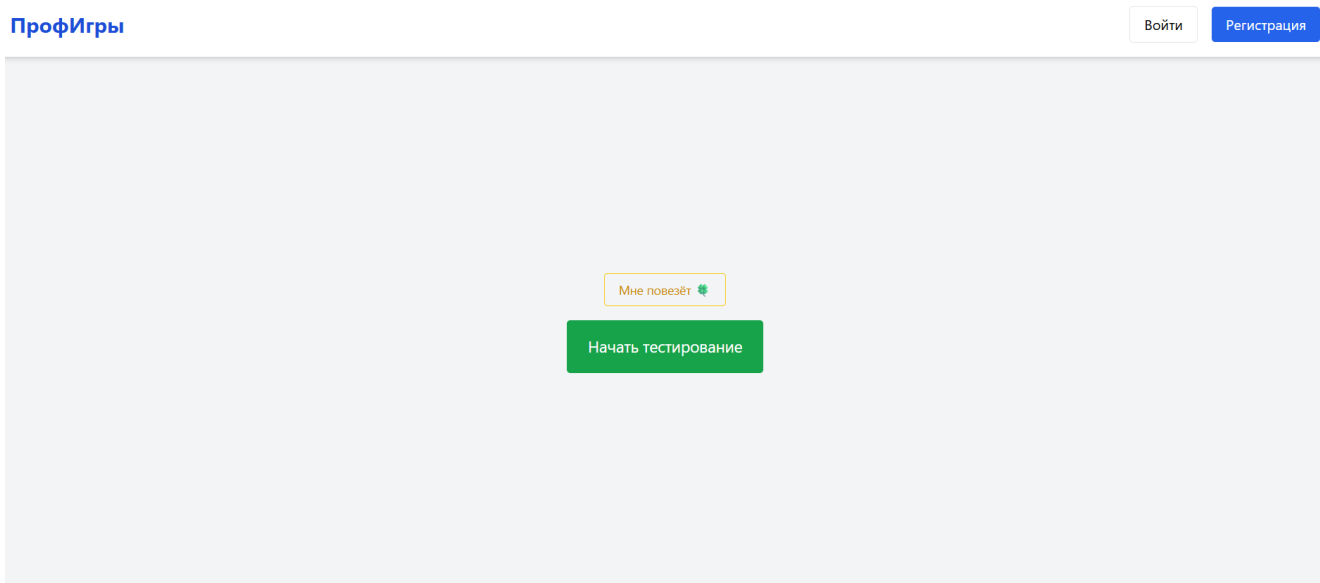


Рисунок 18 – Главная страница для неавторизованного пользователя

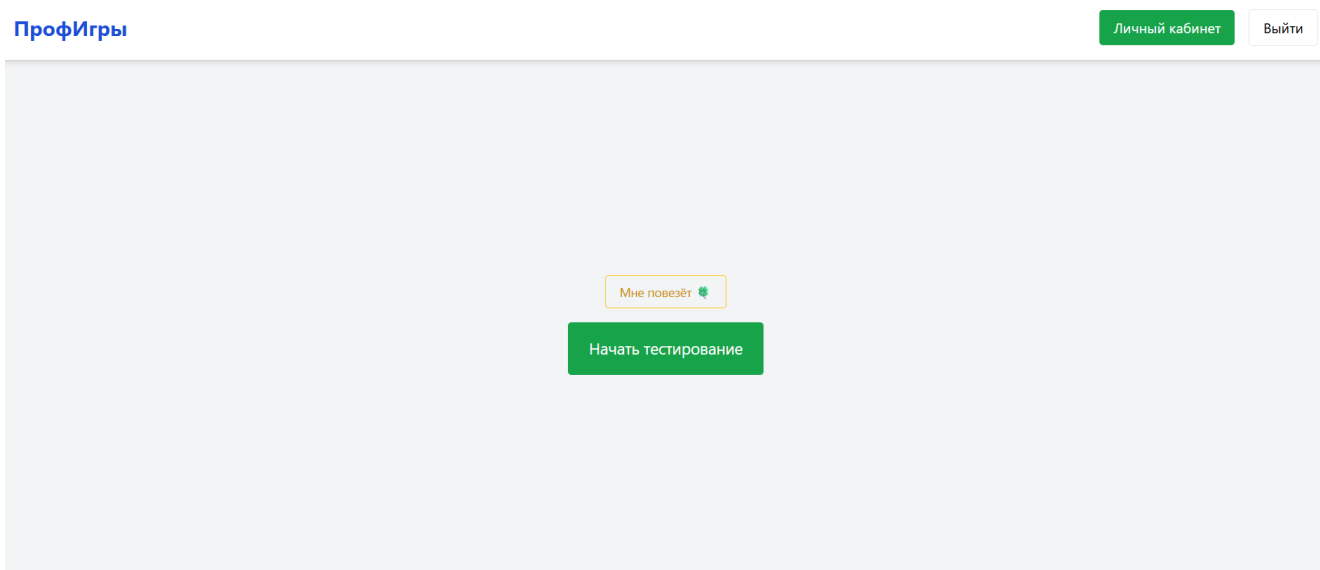


Рисунок 19 – Главная страница для авторизованного пользователя

При нажатии на кнопку «Мне повезёт» пользователь попадает на шуточную страницу с сообщением о том, что его профессия – миллиардер. Страница оформлена в игровом стиле и содержит кнопку для возврата на главную. Это элемент юмора, призванный развеселить пользователя и мотивировать пройти тест всерьёз. С этой страницы можно обратно перейти на главную страницу, нажав соответствующую кнопку. Данная страница представлена на рисунке 20.

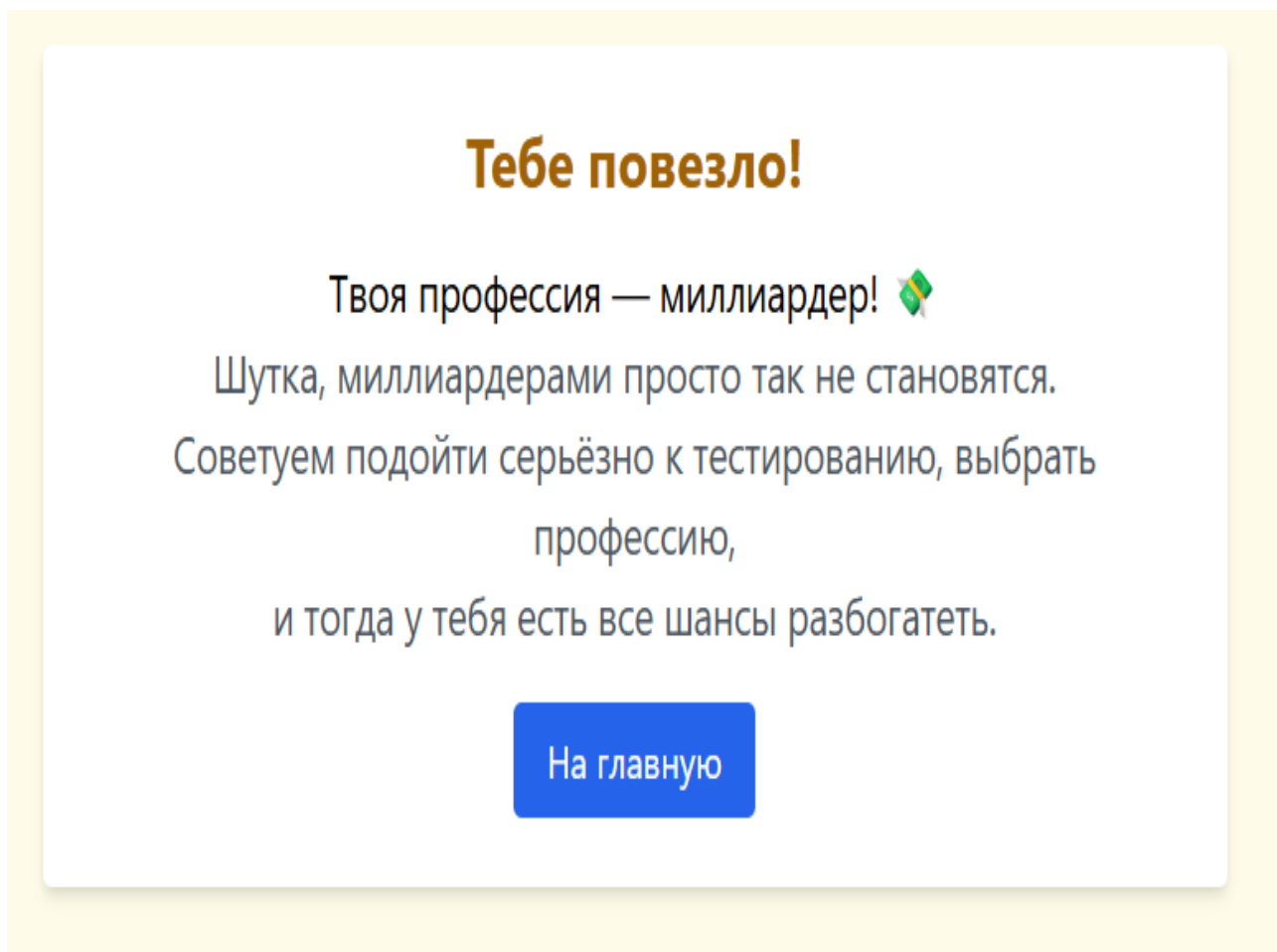


Рисунок 20 – Страница «Мне повезёт»

При нажатии на кнопку «Начать тестирование» пользователь попадает на страницу с пошаговым прохождением теста. Страница состоит из 9 экранов, каждый из которых содержит один вопрос и 4 варианта ответа. После выбора варианта становится активной кнопка «Далее», позволяющая перейти к следующему вопросу. В конце отображается результат с рекомендацией профессии. Страница прохождения тестирования представлена на рисунке 21.

Профориентационный тест

Вопрос 1 из 10:

Что ты предпочёл бы делать в свободное время?

- ☐ Смотреть сериалы
- ☐ Решать головоломки
- ☐ Рисовать
- ☐ Анализировать данные

Следующий вопрос

Рисунок 21 – Страница прохождения тестирования

Когда пользователь доходит до последнего вопроса, появляется кнопка «Завершить». При нажатии на неё тест отправляется на сервер, после чего происходит перенаправление на страницу с результатом. На этой странице отображается подходящая пользователю профессия и краткое описание, а также кнопка для возврата на главную. Страница с результатом тестирования представлена на рисунке 22.

Профориентационный тест

Результат теста

Рекомендуемая профессия: Финансовый аналитик

На главную

Рисунок 22 – Страница результата тестирования

После прохождения тестирования на главной странице можно нажать на кнопку «Личный кабинет», чтобы перейти к просмотру результатов. В личном кабинете отображается список всех ранее пройденных тестов с указанием рекомендованных профессий, а также разбивкой по баллам в зависимости от ответов в тесте с красочной иллюстрацией. Для каждого результата можно открыть подробное описание, а также вернуться обратно на главную страницу с помощью соответствующей кнопки. Страница личного кабинета представлена на рисунке 23.

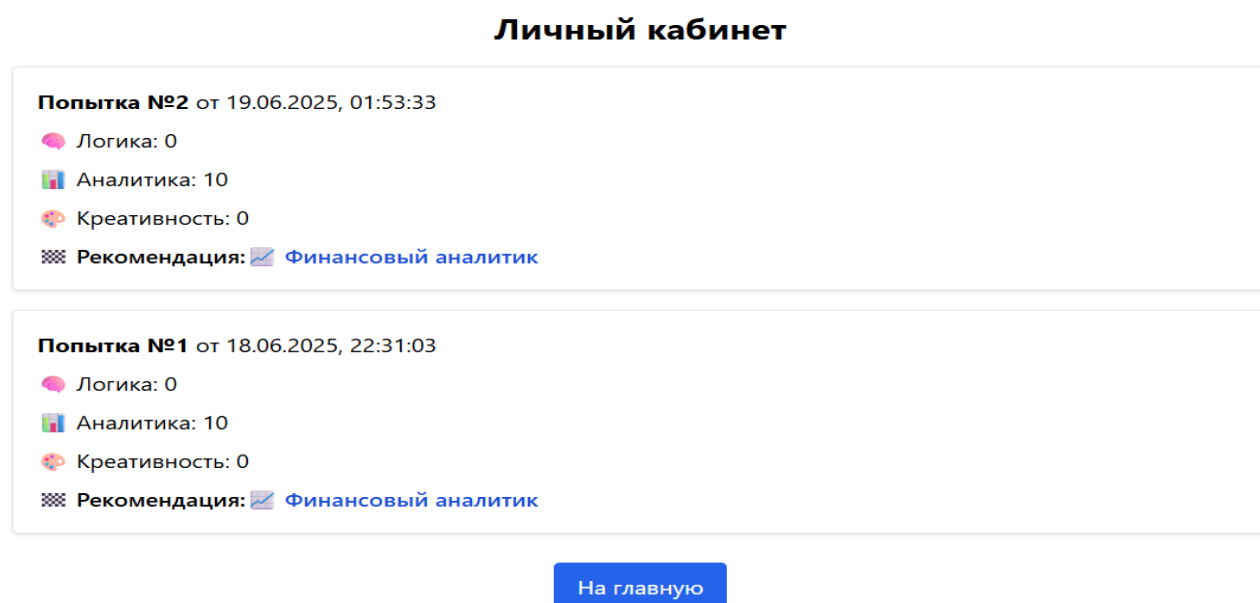


Рисунок 23 – Страница личного кабинета

При нажатии на название профессии на странице с результатами открывается отдельная страница с подробной информацией о выбранной профессии. На этой странице отображаются название, краткое описание и ключевые характеристики профессии. Интерфейс выполнен в виде карточки с информацией и кнопками «В личный кабинет» и «На главную» для возврата к предыдущему экрану. Интерфейс данной страницы представлен на рисунке 24.

Финансовый аналитик

ВУЗ	Проходной балл	Цена / год	ЗП выпускника
Финансовый университет при Правительстве РФ	260	270 000 ₽	140 000 ₽
РЭУ им. Плеханова	250	230 000 ₽	130 000 ₽
МГУ (экономический факультет)	285	380 000 ₽	160 000 ₽
СПбГУ (менеджмент)	270	310 000 ₽	145 000 ₽
НИУ ВШЭ	275	330 000 ₽	150 000 ₽

[В личный кабинет](#)
[На главную](#)

Рисунок 24 – Страница с описанием профессии

В качестве элемента геймификации помимо страницы «Мне повезет» существует страница с шуточной профессией «Рэпер», она открывается если в тесте выбрать самые пассивные варианты ответа, такие как «Люблю лежать на диване» и т. д. После того, как будет дано определенное количество таких ответов данная профессия также появится в личном кабинете. На рисунке 25 показана как выглядит эта страница.

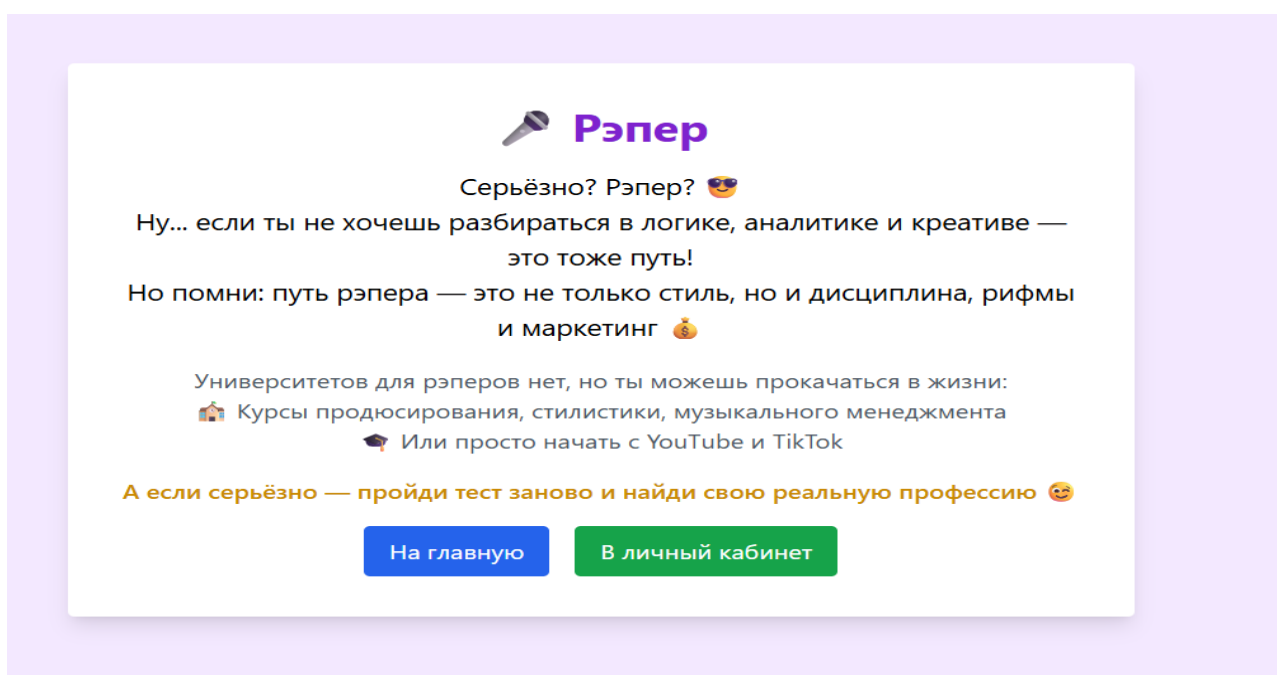


Рисунок 25 – Страница сражения

Таким образом, при разработке графического интерфейса веб-приложения использовались компоненты библиотеки React в сочетании с утилитами Tailwind CSS для стилизации. Интерфейс реализован по принципу одностраничного приложения с использованием маршрутизации, обеспечивающей плавный переход между страницами без перезагрузки. Были разработаны основные пользовательские экраны: регистрация, авторизация, главная страница, тестирование, отображение результатов и личный кабинет, каждый из которых адаптирован под различное состояние пользователя.

4 Тестирование программных средств

4.1 Тестирование белым ящиком

4.1.1 Общие сведения

В рамках ВКР было проведено тестирование белым ящиком алгоритма получения вопросов для тестирования по следующим критериям покрытия кода:

- покрытие операторов;
- покрытие решений;
- покрытие маршрутов.

Ход и результаты тестирования представлены ниже.

4.1.2 Покрытие операторов

Покрытие операторов – один из критериев покрытия кода, который показывает, была ли выполнена и протестирована каждая строка исходного кода. Он предполагает, что каждый оператор (инструкция) в программе должен быть выполнен хотя бы один раз во время выполнения тестов.

Для оценки покрытия операторов был выбран метод получения всех вопросов из класса `QuestionController`. Блок – схема алгоритма получения вопросов представлена на рисунке 26.

Чтобы каждый оператор метода был выполнен хотя бы один раз, был составлен единичный тест-кейс с входными и ожидаемыми выходными данными, указанными в таблице 7.

Таблица 7 – Тест-кейс покрытия операторов

№	Операторы	Входные данные	Выходные данные
1	a, b, c	<code>questionRepository.findAll()</code> , вызов контроллера	JSON-массив с одним вопросом

В данной таблице операторам a, b и c соответствуют инструкции создания объекта `QuestionController`, вызов метода контроллера `getAllQuestions` и вызов у класса репозитория `QuestionRepository` метода `findAll`. На выходе ожидается

JSON – массив с одним вопросом.

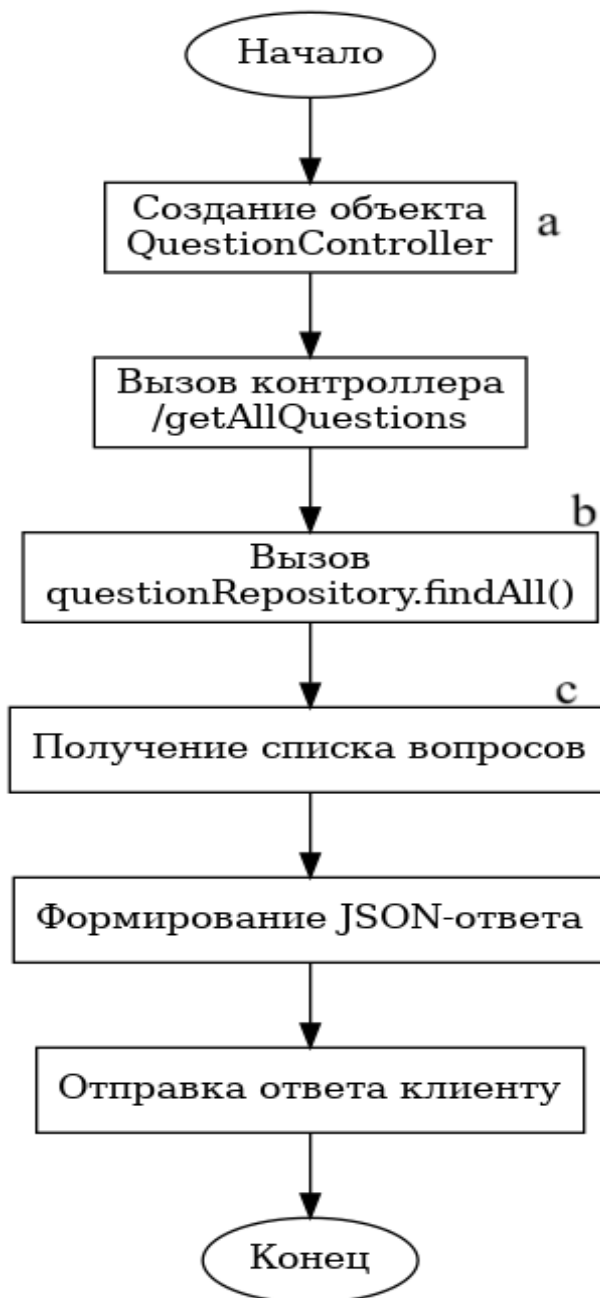


Рисунок 26 – Блок-схема алгоритма получения списка вопросов

Таким образом, тестирование критерия покрытия операторов привело к положительным результатам.

4.1.3 Покрытие решений

Покрытие решений измеряет, насколько хорошо тесты охватывают все возможные решения, то есть ветвления, в программном коде. Покрытие решений означает, что каждая логическая ветвь программы была выполнена в ходе тестов

не менее одного раза.

В рамках тестирования этого критерия стояла задача – написать достаточное количество тестов для прохождения всех логических ветвей, определяющих результат тестирования и выдачу соответствующей рекомендации.

Чтобы выполнить каждую ветвь из каждой точки принятия решения хотя бы один раз, были составлены три тест-кейса, представленные ниже в таблице 8.

Таблица 8 – Тест-кейс покрытия решений

№	Решения	Входные данные	Выходные данные
1	b – нет	Низкие значения по всем категориям (логика, аналитика, креативность)	Рекомендация: «Ваша профессия: рэпер»
2	b – да, e – нет	Максимальный балл только по аналитике	Рекомендация: «Вам подходит профессия аналитика»
3	b – да, e – да	Высокие баллы сразу по двум или трем категориям (например, логика и креатив)	Рекомендация: «Профессия будет определена по доминирующей категории»

Таким образом, покрытие решений позволило протестировать все возможные пути в логике формирования итоговой рекомендации. Это подтвердило корректность реализации условий и обеспечило надёжность подсказок, выдаваемых пользователю по результатам тестирования.

4.1.4 Покрытие маршрутов

Покрытие маршрутов – один из критериев покрытия кода, который показывает, была ли выполнена и протестирована каждая возможная ветвь из каждой точки принятия решения и каждый оператор в исходном коде.

В ходе тестирования этого критерия необходимо было написать такое количество тестов, которое обеспечивало бы прохождение каждого условия как в случае истинного, так и ложного значения, а также выполнение всех

операторов хотя бы один раз.

Для примера рассматривался путь обработки результатов тестирования, в ходе которого система определяет доминирующую категорию ответов (логика, аналитика, креативность) и формирует рекомендацию на основе полученных баллов.

Чтобы пройти все возможные логические пути обработки и задействовать все участки кода, был составлен один полный тест-кейс, представленный в таблице 9.

Таблица 9 – Тест-кейс покрытия маршрутов

№	Условие	Входные данные	Выходные данные
1	a, b, c, d (да), e, f (нет), g (да), h, i, j	Ответы пользователя: преобладает логика, затем креатив, аналитика низкая	Выдана рекомендация: «Подходит профессия, связанная с логикой»

Таким образом, тестирование критерия покрытия маршрутов привело к положительным результатам.

4.2 Автоматизированное тестирование

Для автоматического тестирования были разработаны автотесты, проверяющие корректность работы основных функций веб-приложения профориентационной направленности. В качестве инструмента для написания тестов использовался фреймворк Spring Boot Test совместно с библиотекой MockMvc, которая позволяет выполнять HTTP-запросы к контроллерам без запуска полноценного сервера.

Сборка и управление зависимостями проекта осуществлялись с помощью Maven, что обеспечило удобное подключение всех необходимых библиотек для тестирования и поддержку жизненного цикла проекта. Использование Spring Boot в сочетании с MockMvc позволило построить надёжную, модульную и легко расширяемую систему автоматического тестирования для клиент-серверного приложения.

Автоматизированное тестирование охватило ключевые сценарии взаимодействия с системой, включая регистрацию, авторизацию и обработку

результатов тестирования. Благодаря покрытию базовых маршрутов и бизнес-логики, удалось убедиться в корректности работы контроллеров и устойчивости приложения к некорректным входным данным. Код класса контроллера регистрации и авторизации представлен в приложении В. Код тест – кейсов для него представлен на листинге 9.

Листинг 9 – Код тест – кейсов проверки аутентификации и регистрации

```
@Autowired
private MockMvc mockMvc;
@Autowired
private UserRepository userRepository;

@BeforeEach
void setup() {
    userRepository.deleteAll(); // очищаем БД перед каждым тестом
}

@Test
void successfulRegistration() throws Exception {
    String json = """
        {
            "email": "newuser@example.com",
            "password": "secure123"
        }
        """;

    mockMvc.perform(post("/api/auth/register")
        .contentType(MediaType.APPLICATION_JSON)
        .content(json)
        .andExpect(status().isOk())
        .andExpect(content().string("Регистрация успешна")));
}

@Test
void login() throws Exception {
    String json = """
        {
            "email": "user@example.com",
            "password": "correct123"
        }
        """;

    mockMvc.perform(post("/api/auth/login")
        .contentType(MediaType.APPLICATION_JSON)
        .content(json)
        .andExpect(status().isOk())); // если в ответе токен
}
```

На листинге 10 будет представлен код контроллера вопросов.

Листинг 10 – Код тест – кейсов проверки поиска вопросов и сохранения результатов пользователя

```

@SpringBootTest
@AutoConfigureMockMvc
class QuestionControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @MockBean
    private QuestionRepository questionRepository;

    @MockBean
    private OptionRepository optionRepository;

    @MockBean
    private TestResultRepository testResultRepository;

    @MockBean
    private UserRepository userRepository;

    @Test
    void testGetAllQuestionsReturnsQuestions() throws Exception {
        Question q = new Question();
        q.setId(1L);
        q.setQuestionText("Что тебе больше всего интересно?");
        Option opt = new Option();
        opt.setId(1L);
        opt.setOptionText("Логика");
        opt.setCategory("logic");
        opt.setWeight(3);
        q.setOptions(List.of(opt));

        Mockito.when(questionRepository.findAllWithOptions()).thenReturn(List.of(q));

        mockMvc.perform(get("/api/questions"))
            .andExpect(status().isOk())
            .andExpect(jsonPath("$.questionText").value("Что тебе больше всего интересно?"))
            .andExpect(jsonPath("$.options[0].category").value("logic"));
    }

    @Test
    void testSubmitTestReturnsRecommendation() throws Exception {
        // создаем моки
        Long userId = 1L;
        Long optionId = 100L;

        Option opt = new Option();
    }

```

```

    opt.setId(optionId);
    opt.setCategory("logic");
    opt.setWeight(5);

    User user = new User();
    user.setId(userId);
    user.setEmail("test@example.com");

    Mockito.when(optionRepository.findById(optionId)).thenReturn(Optional.of(opt));

    Mockito.when(userRepository.findById(userId)).thenReturn(Optional.of(user));

    Mockito.when(testResultRepository.countByUser(user)).thenReturn(0);

    String json = String.format("""
        {
            "userId": %d,
            "selectedOptionIds": [%d]
        }
    """, userId, optionId);

    mockMvc.perform(post("/api/submit-test")
        .contentType(MediaType.APPLICATION_JSON)
        .content(json)
        .andExpect(status().isOk())
        .andExpect(content().string("Рекомендуемая
профессия: Рэпер"))); // т.к. логика 5, но не >=8
    }
}

```

Результат тестирования всех тест – кейсов представлен на рисунке 27.

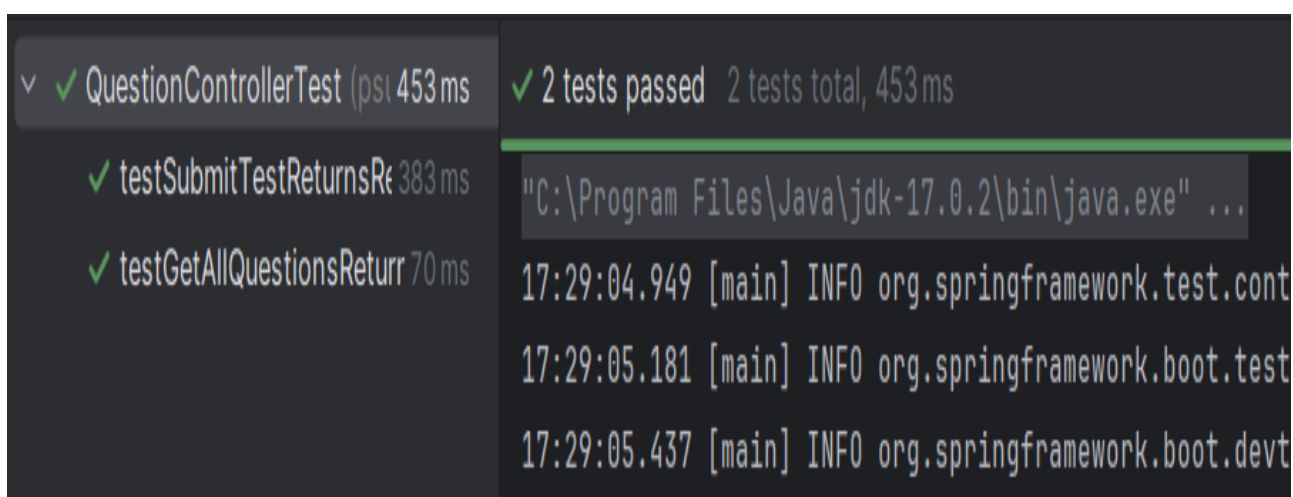


Рисунок 27 – Результат автоматизированного тестирования системы
профориентационного ориентирования

На основе полученных положительных результатов автотестирования можно сделать вывод о корректности работы информационной системы профориентационной работы школьников.

4.3 Метрика SLOC

Метрика SLOC представляет собой метрику количества строк кода. Подсчет данной метрики был проведен для модуля, отвечающего за получение вопросов и обработки результатов тестирования. Код модуля приведен в листинге Б.2.

В данной программе имеем:

- 64 физических строк кода;
- 40 логических строк кода;
- 0 строк комментариев.

Эта оценка позволяет судить о средней сложности контроллера и его логической насыщенности, что важно при анализе покрытия тестами и расчёте трудозатрат на сопровождение.

4.4 Метрика Холстеда

Для страницы отображения рейтинга были подсчитаны метрики Холстеда. Метрики Холстеда относятся к метрикам, вычисляемым на основании анализа числа строк и синтаксических элементов исходного кода программы [13].

Для расчёта метрик Холстеда были взяты функции, создающие на клиенте таблицу рейтинга, заполняющие ее данными и отображающие необходимые кнопки. Код данного модуля представлен в листинге Б.12.

Количество используемых операндов и операторов, а также их общее число в программе приведены в таблицах 10 и 11 соответственно.

Таблица 10 – Операторы

Операторы	Число вхождений
=	48
:	10
=>	8

Продолжение таблицы 10 – Операторы

Операторы	Число вхождений
useState	6
useEffect	2
async	1
await	1
try	1
catch	2
if	2
return	3
const	9
import	3
export	1
function	1
==	1
===	1
...	2
&&	2
?	1
.	31
n1=21	N1=136

Ключевыми параметрами данной таблицы является параметр n1, отображающий количество уникальных операторов, и N1, отображающий общее число уникальных операторов.

Таблица 11 – Операнды

Операнды	Число вхождений
questions	6
setQuestion	2
currentIndex	5
setCurrentIndex	2
answers	6
setAnswers	2
loading	2
setLoading	3
result	4
setResult	2
navigate	2
axios	4
res	2
err	4
response	2
data	2
selectedOptionIds	1

Продолжение таблицы 11 – Операнды

Операнды	Число вхождений
currentQuestion	8
opt	6
id	9
optionText	1
handleFinish	2
useEffect	2
useState	6
useNavigate	2
1	7
0	15
true	1
null	1
http://localhost:8080/api/	2
/	29
n2 = 34	N2=143

Ключевыми параметрами данной таблицы является параметр n2, отображающий количество уникальных операторов, и N2, отображающий общее число уникальных операндов.

Длина программы рассчитывается по формуле:

$$N = N1 + N2 = 136 + 143 = 279.$$

Словарь программы рассчитывается по формуле:

$$N = n1 + n2 = 21 + 34 = 55.$$

Объем программы рассчитывается по формуле:

$$N_u = N * \log_2(n) = 279 * 6,17 = 1612,99.$$

Сложность программы рассчитывается по формуле:

$$HD = (n1 / 2) * (N2 / n2) = (21 / 2) * (143 / 34) = 44,161$$

Полученные показатели объема и сложности программы удовлетворяют требованиям по метрикам программного кода.

Заключение

По итогам выполнения выпускной квалификационной работы была спроектирована и разработана информационная система по организации профориентационной работы школьников в игровой форме.

В ходе работы были выполнены следующие задачи:

- проведен анализ предметной области;
- проведен анализ аналогов;
- проведен анализ требований к разрабатываемой системе;
- выполнена постановка задачи;
- спроектирована архитектура приложения;
- спроектирован графический интерфейс;
- выполнена оценка бюджета разработки;
- разработано спроектированное приложение;
- выполнено тестирование белым ящиком;
- выполнено автоматизированное тестирование;
- посчитана метрика SLOC;
- посчитана метрика Холстеда.

Разработка информационной системы по организации профориентационной работы школьников в игровой форме состояла из двух основных частей: разработка клиентской части и разработка серверной части. Клиентская часть была разработана в среде разработки Visual Studio Code с использованием языка программирования JavaScript, серверная же часть была разработана в среде разработки IntelliJIDEA на языке программирования Java.

Все этапы сопровождались диаграммами и пояснительным текстом к ним. Разработанное программное обеспечение отвечает всем требованиям технического задания.

Список использованных источников

1. Профориентация: как помочь школьнику выбрать профессию. [Электронный ресурс]. URL: <https://infourok.ru/proforientaciya-kak-pomoch-shkolniku-vybrat-professiyu-6782439.html> (Дата обращения: 19.05.2025);
2. Википедия. Диаграмма прецедентов (Use Case Diagram). [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Диаграмма_прецедентов (дата обращения: 14.05.2025)
3. Для чего нужны UML диаграммы? [Электронный ресурс]. URL: <https://www.run-it.com.ua/ru/dlia-cheho-nuzhny-uml-dyahrammy/> (Дата обращения: 11.05.2025);
4. Популярные шаблоны архитектуры программного обеспечения. [Электронный ресурс]. URL: <https://tavtechsolutions.com/ru/> (Дата обращения: 28.04.2025);
5. Диаграмма деятельности – UML-диаграмма, отражающая деятельность и потоки управления в системе. [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Диаграмма_деятельности (Дата обращения: 9.05.2025);
6. What are Microservices? How monolithic architecture compares. Blog article. [Электронный ресурс]. URL: <https://middleware.io/blog/microservices-architecture/> (Дата обращения: 1.05.2025);
7. Ключевые принципы UI-дизайна: Clarity, Consistency, Accessibility. [Электронный ресурс]. URL: <https://maze.co/collections/ux-ui-design/ui-design-principles/> (Дата обращения: 15.05.2025);
8. NEKLO. Software Development Cost Estimation: How Do It Right. [Электронный ресурс]. URL: <https://neklo.com/blog/software-development-cost-estimation> (Дата обращения: 28.05.2025);
9. Tailwind CSS. Editor setup – официальный гайд по интеграции с редакторами кода. [Электронный ресурс]. URL: <https://tailwindcss.com/docs/editor-setup> (Дата обращения: 30.05.2025);

10. Benefits of Using Reactjs for Your Next Project. CodeWalnut. [Электронный ресурс]. URL: <https://www.codewalnut.com/learn/11-benefits-of-using-reactjs-for-your-next-project> (Дата обращения: 01.06.2025).

Приложение А

(обязательное)

Листинг приложения

Листинг А.1 – Файл «AuthController.java»

```

@RestController
@RequestMapping("/api/auth")
@CrossOrigin(origins = "*") // разрешаем запросы с React
public class AuthController {

    @Autowired
    private UserRepository userRepository;

    @PostMapping("/register")
    public String registerUser(@RequestBody User user) {
        if (userRepository.existsByEmail(user.getEmail())) {
            return "Пользователь с таким email уже существует";
        }
        System.out.println(user.getEmail());
        System.out.println(user.getPassword());

        // Пароль пока сохраняется как есть – позже добавим
        шифрование (например, BCrypt)
        userRepository.save(user);
        return "Регистрация успешна";
    }

    @PostMapping("/login")
    public String login(@RequestBody User user) {
        Optional<User> found =
userRepository.findByEmail(user.getEmail());
        if (found.isEmpty()) return "Пользователь не найден";

        if (!found.get().getPassword().equals(user.getPassword()))
        {
            return "Неверный пароль";
        }

        return "Вход успешно выполнен";
    }
}

```

Листинг А.2 – Файл «QuestionController.java»

```

@CrossOrigin(origins = "http://localhost:3000")
@RestController
@RequestMapping("/api")
public class QuestionController {
    @Autowired
    private QuestionRepository questionRepository;
    @Autowired
    private OptionRepository optionRepository;
    @Autowired
    private TestResultRepository testResultRepository;
    @Autowired
    private UserRepository userRepository;
}

```

```

@GetMapping("/questions")
public List<Question> getAllQuestions() {
    return questionRepository.findAllWithOptions();
}

@PostMapping("/submit-test")
public String submitTest(@RequestBody TestSubmitRequest
request) {
    List<Long> optionIds = request.getSelectedOptionIds();
    User user =
userRepository.findById(request.getUserId()).orElseThrow();

    Map<String, Integer> scores = new HashMap<>();
    scores.put("logic", 0);
    scores.put("analytics", 0);
    scores.put("creativity", 0);

    for (Long id : optionIds) {
        Option opt =
optionRepository.findById(id).orElse(null);
        if (opt != null && opt.getCategory() != null) {
            scores.put(opt.getCategory(),
scores.get(opt.getCategory()) + opt.getWeight());
        }
    }

    // Простая логика рекомендаций
    int logic = scores.get("logic");
    int analytics = scores.get("analytics");
    int creativity = scores.get("creativity");

    String recommendation;
    if (logic >= 8 || (logic >= 5 && analytics >= 3)) {
        recommendation = "Рекомендуемая профессия:
Программист";
    } else if (analytics >= 5) {
        recommendation = "Рекомендуемая профессия: Финансовый
аналитик";
    } else if (creativity >= 5) {
        recommendation = "Рекомендуемая профессия: Дизайнер";
    } else {
        recommendation = "Рекомендуемая профессия: Рэпер";
    }

    // сохраняем результат в БД
    TestResult result = new TestResult();
    result.setUser(user);
    result.setLogicScore(logic);
    result.setAnalyticsScore(analytics);
    result.setCreativityScore(creativity);
    result.setRecommendation(recommendation);
    result.setTimestamp(LocalDateTime.now());
}

```

```

result.setAttemptNumber(testResultRepository.countByUser(user) +
1);

        testResultRepository.save(result);

        return recommendation;
    }

}

```

Листинг А.3 – Файл «TestResultController.java»

```

@RestController
@RequestMapping("/api/profile")
@CrossOrigin(origins = "http://localhost:3000")
public class TestResultController {
    @Autowired
    private TestResultRepository testResultRepository;

    @Autowired
    private UserRepository userRepository;

    @GetMapping("/{userId}/results")
    public List<TestResult> getUserResults(@PathVariable Long
userId) {
        User user = userRepository.findById(userId).orElseThrow();
        return
testResultRepository.findByUserOrderByTimestampDesc(user);
    }

}

```

Листинг А.4 – «Файл TestSubmitRequest.java»

```

@Getter
@Setter
public class TestSubmitRequest {
    private Long userId;
    private List<Long> selectedOptionIds;
}

```

Листинг А.5 – Файл «Option.java»

```

@Entity
public class Option {

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

    private String optionText;

    private String category; // Например: "logic", "analytics",
"creativity"
}

```

```

private int weight; // Например: 1

@ManyToOne
@JoinColumn(name = "question_id")
@JsonIgnore
private Question question;

// геттеры и сеттеры
public Long getId() { return id; }

public void setId(Long id) { this.id = id; }

public String getOptionText() { return optionText; }

public void setOptionText(String optionText) { this.optionText
= optionText; }

public String getCategory() { return category; }

public void setCategory(String category) { this.category =
category; }

public int getWeight() { return weight; }

public void setWeight(int weight) { this.weight = weight; }

public Question getQuestion() { return question; }

public void setQuestion(Question question) { this.question =
question; }
}

```

Листинг Б.6 - Файл Question.java

```

@Entity
public class Question {

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

    private String questionText;

    @OneToMany(mappedBy = "question", cascade = CascadeType.ALL,
fetch = FetchType.LAZY)
    private List<Option> options;

    // геттеры и сеттеры
    public Long getId() { return id; }

    public void setId(Long id) { this.id = id; }

    public String getQuestionText() { return questionText; }
}

```

```

    public void setQuestionText(String questionText) {
        this.questionText = questionText; }

    public List<Option> getOptions() { return options; }

    public void setOptions(List<Option> options) {
        this.options = options;
        // также можно связать обратно:
        for (Option option : options) {
            option.setQuestion(this);
        }
    }
}

```

Листинг А.6 – Файл «TestResult.java»

```

@Entity
@Getter
@Setter
public class TestResult {

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

    private int logicScore;
    private int analyticsScore;
    private int creativityScore;

    private String recommendation;

    private int attemptNumber;

    private LocalDateTime timestamp;

    @ManyToOne
    @JoinColumn(name = "user_id")
    private User user;
}

```

Листинг А.7 – Файл «User.java»

```

@Entity
@Table(name = "users") // можно задать любое имя таблицы
public class User {

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

    @Column(unique = true, nullable = false)
    private String email;
}

```

```

@Column(nullable = false)
private String password;

// Геттеры и сеттеры

public Long getId() {
    return id;
}

public void setId(Long id) {
    this.id = id;
}

public String getEmail() {
    return email;
}

public void setEmail(String email) {
    this.email = email;
}

public String getPassword() {
    return password;
}

public void setPassword(String password) {
    this.password = password;
}
}

```

Листинг A.8 – Файл «OptionRepository.java»

```

public interface OptionRepository extends JpaRepository<Option,
Long> {
}

```

Листинг A.9 – Файл «QuestionRepository.java»

```

@Repository
public interface QuestionRepository extends
JpaRepository<Question, Long> {
    @Query("SELECT q FROM Question q LEFTJOIN FETCH q.options")
    List<Question> findAllWithOptions();
}

```

Листинг A.10 – Файл «TestResultRepository.java»

```

public interface TestResultRepository extends
JpaRepository<TestResult, Long> {
    List<TestResult> findByUserOrderByTimestampDesc(User user);
    int countByUser(User user);
}

```

Листинг А.11 – Файл «UserRepository.java»

```
public interface UserRepository extends JpaRepository<User, Long>
{
    boolean existsByEmail(String email);
    Optional<User> findByEmail(String email);
}
```

Листинг А.12 – Файл «DiplomApplication.java»

```
@SpringBootApplication
public class DiplomApplication {

    public static void main(String[] args) {
        SpringApplication.run(DiplomApplication.class, args);
    }

}
```

Листинг А.13 – Файл «careerData.js»

```
export const careerInfo = {
  "Программист": [
    {
      university: "МФТИ",
      budgetScore: 285,
      pricePerYear: "350 000 ₽",
      avgSalary: "180 000 ₽"
    },
    {
      university: "ВШЭ",
      budgetScore: 280,
      pricePerYear: "360 000 ₽",
      avgSalary: "160 000 ₽"
    },
    {
      university: "ИТМО",
      budgetScore: 270,
      pricePerYear: "250 000 ₽",
      avgSalary: "150 000 ₽"
    },
    {
      university: "МГУ (ВМК)",
      budgetScore: 290,
      pricePerYear: "410 000 ₽",
      avgSalary: "190 000 ₽"
    },
    {
      university: "СПбГУ",
      budgetScore: 275,
      pricePerYear: "300 000 ₽",
      avgSalary: "155 000 ₽"
    }
  ],
}
```

```

"Финансовый аналитик": [
  {
    university: "Финансовый университет при Правительстве РФ",
    budgetScore: 260,
    pricePerYear: "270 000 ₽",
    avgSalary: "140 000 ₽"
  },
  {
    university: "РЭУ им. Плеханова",
    budgetScore: 250,
    pricePerYear: "230 000 ₽",
    avgSalary: "130 000 ₽"
  },
  {
    university: "МГУ (экономический факультет)",
    budgetScore: 285,
    pricePerYear: "380 000 ₽",
    avgSalary: "160 000 ₽"
  },
  {
    university: "СПбГУ (менеджмент)",
    budgetScore: 270,
    pricePerYear: "310 000 ₽",
    avgSalary: "145 000 ₽"
  },
  {
    university: "НИУ ВШЭ",
    budgetScore: 275,
    pricePerYear: "330 000 ₽",
    avgSalary: "150 000 ₽"
  }
],
"Дизайнер": [
  {
    university: "Британская высшая школа дизайна",
    budgetScore: null,
    pricePerYear: "500 000 ₽",
    avgSalary: "120 000 ₽"
  },
  {
    university: "СПбГУ (искусства и культуры)",
    budgetScore: 240,
    pricePerYear: "250 000 ₽",
    avgSalary: "100 000 ₽"
  },
  {
    university: "МИГХИ",
    budgetScore: 230,
    pricePerYear: "220 000 ₽",
    avgSalary: "90 000 ₽"
  },
  {
    university: "МГХПА им. Строганова",

```

```

        budgetScore: 245,
        pricePerYear: "260 000 ₺",
        avgSalary: "110 000 ₺"
      },
      {
        university: "НИУ ВШЭ (дизайн)",
        budgetScore: 270,
        pricePerYear: "340 000 ₺",
        avgSalary: "115 000 ₺"
      }
    ]
  };

```

Листинг А.14 – Файл «CareerPage.jsx»

```

import React from "react";
import { useParams, useNavigate } from "react-router-dom";
import { careerInfo } from "../data/careerData";

function CareerPage() {
  const { profession } = useParams();
  const navigate = useNavigate();

  const data = careerInfo[profession];

  if (!data) {
    return <div className="p-4">Информация по профессии не
    найдена.</div>;
  }

  return (
    <div className="max-w-4xl mx-auto p-6">
      <h1 className="text-2xl font-bold mb-4 text-
      center">{profession}</h1>

      <table className="w-full border-collapse">
        <thead>
          <tr className="bg-gray-200">
            <th className="border px-4 py-2 text-left">ВУЗ</th>
            <th className="border px-4 py-2">Проходной балл</th>
            <th className="border px-4 py-2">Цена / год</th>
            <th className="border px-4 py-2">ЗП выпускника</th>
          </tr>
        </thead>
        <tbody>
          {data.map((item, index) => (
            <tr key={index} className="text-center">
              <td className="border px-4 py-2 text-
              left">{item.university}</td>
              <td className="border px-4 py-2">{item.budgetScore
              || "-"}</td>
              <td className="border px-4 py-
              2">{item.pricePerYear}</td>
              <td className="border px-4 py-

```

```

2">{item.avgSalary}</td>
    </tr>
  )})
</tbody>
</table>

<div className="mt-6 flex gap-4 justify-center">
  <button
    onClick={() => navigate("/profile")}
    className="bg-green-600 text-white px-4 py-2 rounded"
  >
    В личный кабинет
  </button>
  <button
    onClick={() => navigate("/") }
    className="bg-blue-600 text-white px-4 py-2 rounded"
  >
    На главную
  </button>
</div>
</div>
);
}

```

```
export default CareerPage;
```

Листинг A.15 – Файл «HomePage.jsx»

```

import React, { useEffect, useState } from "react";
import { useNavigate } from "react-router-dom";

function HomePage() {
  const navigate = useNavigate();
  const [isLoggedIn, setIsLoggedIn] = useState(false);

  useEffect(() => {
    // Проверяем флаг при загрузке
    const loggedIn = localStorage.getItem("isLoggedIn") ===
"true";
    setIsLoggedIn(loggedIn);
  }, []);

  const handleLogout = () => {
    localStorage.removeItem("isLoggedIn");
    setIsLoggedIn(false);
    navigate("/");
  };

  return (
    <div className="min-h-screen bg-gray-100 flex flex-col">
      { /* Шапка */ }
      <header className="bg-white shadow-md p-4 flex justify-
between items-center">
        <h1 className="text-2xl font-bold text-blue-

```

```

700">ПрофИгры</h1>
    <div className="space-x-4">
        {isLoggedIn ? (
            <>
                <button
                    onClick={() => navigate("/profile")}
                    className="bg-green-600 text-white px-4 py-2
rounded"
                >
                    Личный кабинет
                </button>
                <button
                    onClick={handleLogout}
                    className="border px-4 py-2 rounded"
                >
                    Выйти
                </button>
            </>
        ) : (
            <>
                <button
                    onClick={() => navigate("/login")}
                    className="border px-4 py-2 rounded"
                >
                    Войти
                </button>
                <button
                    onClick={() => navigate("/register")}
                    className="bg-blue-600 text-white px-4 py-2
rounded"
                >
                    Регистрация
                </button>
            </>
        )}
    </div>
</header>

{/* КОНТЕНТ */}
<main className="flex-grow flex items-center justify-
center">
    <div className="flex flex-col items-center space-y-4">
        <button
            onClick={() => navigate("/lucky")}
            className="text-sm px-4 py-2 border border-yellow-400
text-yellow-600 rounded hover:bg-yellow-100"
        >
            Мне повезёт 🍀
        </button>
        <button
            onClick={() => navigate("/test")}
            className="text-lg px-6 py-4 bg-green-600 text-white
rounded"

```

```

        >
        Начать тестирование
    </button>
</div>
</main>

    { /* Футер */ }
    <footer className="bg-white text-center p-4 shadow-inner
text-sm text-gray-500">
        &copy; 2025 ПроФИгры. Все права защищены.
    </footer>
</div>
);
}

export default HomePage;

```

Листинг А.16 – Файл «LoginPage.jsx»

```

import React, { useState } from "react";
import { useNavigate } from "react-router-dom";
import axios from "axios";

function LoginPage() {
    const navigate = useNavigate();
    const [email, setEmail] = useState("");
    const [password, setPassword] = useState("");
    const [error, setError] = useState("");

    const handleLogin = async (e) => {
        e.preventDefault();

        try {
            const response = await
            axios.post("http://localhost:8080/api/auth/login", {
                email,
                password,
            });

            if (response.data.includes("успешно")) {
                localStorage.setItem("isLoggedIn", "true");
                navigate("/");
            } else {
                setError(response.data); // Показываем ошибку от backend
            }
        } catch (err) {
            console.error(err);
            setError("Ошибка сервера или сеть недоступна");
        }
    };

    return (
        <div className="min-h-screen flex flex-col items-center
justify-center bg-gray-100">

```

```

    <div className="bg-white p-8 rounded shadow-md w-full max-w-
md">
      <h2 className="text-2xl font-bold mb-6 text-
center">Вход</h2>
      <form onSubmit={handleLogin} className="space-y-4">
        {error && <div className="text-red-600 text-
sm">{error}</div>}
        <div>
          <label className="block mb-1">Email</label>
          <input
            type="email"
            required
            value={email}
            onChange={(e) => setEmail(e.target.value)}
            className="w-full border rounded px-3 py-2
focus:outline-none focus:ring"
          />
        </div>
        <div>
          <label className="block mb-1">Пароль</label>
          <input
            type="password"
            required
            value={password}
            onChange={(e) => setPassword(e.target.value)}
            className="w-full border rounded px-3 py-2
focus:outline-none focus:ring"
          />
        </div>
        <button
          type="submit"
          className="bg-blue-600 text-white w-full py-2 rounded"
        >
          Войти
        </button>
      </form>
    </div>
  </div>
);
}

```

```
export default LoginPage;
```

Листинг А.17 – Файл «LuckyPage.jsx»

```

import React from "react";
import { useNavigate } from "react-router-dom";

function LuckyPage() {
  const navigate = useNavigate();

  return (
    <div className="min-h-screen flex items-center justify-center
bg-yellow-50 p-6">

```

```

    <div className="bg-white p-6 rounded shadow-md max-w-xl
text-center space-y-4">
      <h1 className="text-2xl font-bold text-yellow-700">Тебе
повезло!</h1>
      <p className="text-lg">
        Твоя профессия - миллиардер! 🤖
      <br />
      <span className="text-gray-600">
        Шутка, миллиардерами просто так не становятся.
      <br />
        Советуем подойти серьёзно к тестированию, выбрать
профессию,
      <br />и тогда у тебя есть все шансы разбогатеть.
      </span>
    </p>
    <button
      onClick={() => navigate("/")}
      className="mt-4 bg-blue-600 text-white px-4 py-2
rounded"
    >
      На главную
    </button>
  </div>
</div>
);
}

```

```
export default LuckyPage;
```

Листинг А.18 – Файл «ProfilePage.jsx»

```

import React, { useEffect, useState } from "react";
import axios from "axios";
import { useNavigate } from "react-router-dom";
import { Link } from "react-router-dom";

function ProfilePage() {
  const [results, setResults] = useState([]);
  const [loading, setLoading] = useState(true);

  const navigate = useNavigate();
  const userId = 1; // пока захардкожено, позже заменим на
  реального пользователя

  useEffect(() => {
    axios
      .get(`http://localhost:8080/api/profile/${userId}/results`)
      .then((res) => {
        setResults(res.data);
        setLoading(false);
      })
      .catch((err) => {
        console.error("Ошибка загрузки истории тестов", err);
      });
  });
}

```

```

        setLoading(false);
    });
}, []);

if (loading) return <div className="p-4">Загрузка
истории...</div>;

const getProfessionBadge = (recommendation) => {
    if (recommendation.includes("Программист")) {
        return (
            <span className="inline-flex items-center gap-2 text-green-
700 font-semibold">
                🖥 Программист
            </span>
        );
    }
    if (recommendation.includes("Финансовый аналитик")) {
        return (
            <span className="inline-flex items-center gap-2 text-blue-
700 font-semibold">
                📊 Финансовый аналитик
            </span>
        );
    }
    if (recommendation.includes("Дизайнер")) {
        return (
            <span className="inline-flex items-center gap-2 text-pink-
700 font-semibold">
                🎨 Дизайнер
            </span>
        );
    }
    return (
        <span className="inline-flex items-center gap-2 text-gray-
500">
            🖋 Рэпер
        </span>
    );
};

return (
    <div className="max-w-4xl mx-auto p-6">
        <h1 className="text-2xl font-bold mb-4 text-center">Личный
кабинет</h1>

        {results.length === 0 ? (
            <p className="text-center">Вы ещё не проходили тест.</p>
        ) : (
            <div className="space-y-4">
                {results.map((res) => (
                    <div
                        key={res.id}

```

```

        className="border rounded p-4 shadow bg-white space-
y-2"
      >
        <p>
          <strong>Попытка №{res.attemptNumber}</strong>{" "}
            от {new Date(res.timestamp).toLocaleString()}
        </p>
        <p>🧠 Логика: {res.logicScore}</p>
        <p>📊 Аналитика: {res.analyticsScore}</p>
        <p>🎨 Креативность: {res.creativityScore}</p>
        <p className="font-semibold">
          🏆 Рекомендация:
          <Link
            to={` /career/${encodeURIComponent(res.recommendation.replace("Реко-
            мендуемая профессия: ", ""))}`}>
            {getProfessionBadge(res.recommendation)}
          </Link>
        </p>
      </div>
    )}
  </div>
)}

<div className="mt-6 text-center">
  <button
    onClick={() => navigate("/") }
    className="bg-blue-600 text-white px-4 py-2 rounded"
  >
    На главную
  </button>
</div>
</div>
);
}

```

```
export default ProfilePage;
```

Листинг А.19 – Файл «RapperCareerPage.jsx»

```

import React from "react";
import { useNavigate } from "react-router-dom";

function RapperCareerPage() {
  const navigate = useNavigate();

  return (
    <div className="min-h-screen bg-purple-100 flex items-center
    justify-center p-6">
      <div className="bg-white rounded shadow-lg p-8 max-w-2xl
      text-center space-y-4">
        <h1 className="text-3xl font-bold text-purple-700">🎤
        Рэпер</h1>
        <p className="text-lg">

```

```

        Серьёзно? Рэпер? 🤔<br />
        Ну... если ты не хочешь разбираться в логике, аналитике
и креативе – это тоже путь!<br />
        Но помни: путь рэпера – это не только стиль, но и
дисциплина, рифмы и маркетинг 💰
    </p>
    <p className="text-gray-600">
        Университетов для рэперов нет, но ты можешь прокачаться
в жизни:<br />
        🎓! Курсы продюсирования, стилистики, музыкального
менеджмента<br />
        📺 Или просто начать с YouTube и TikTok
    </p>
    <p className="text-yellow-600 font-semibold">
        А если серьёзно – пройди тест заново и найди свою
реальную профессию 😊
    </p>
    <div className="mt-6 flex gap-4 justify-center">
        <button
            onClick={() => navigate("/")}
            className="bg-blue-600 text-white px-4 py-2 rounded"
        >
            На главную
        </button>
        <button
            onClick={() => navigate("/profile")}
            className="bg-green-600 text-white px-4 py-2 rounded"
        >
            В личный кабинет
        </button>
    </div>
</div>
</div>
);
}

```

```
export default RapperCareerPage;
```

Листинг A.20 – Файл «RegistrarionPage.jsx»

```

import React, { useState } from "react";
import { useNavigate } from "react-router-dom";
import axios from "axios";

function RegistrationPage() {
    const navigate = useNavigate();
    const [email, setEmail] = useState("");
    const [password, setPassword] = useState("");
    const [error, setError] = useState("");

    const handleRegister = async (e) => {
        try {

```

```

    const response = await
    axios.post("http://localhost:8080/api/auth/register", {
      email,
      password,
    });

    if (response.data.includes("успешна")) {
      navigate("/"); // Перенаправление на главную при успехе
    } else {
      setError(response.data); // Показать сообщение от backend
    }
  } catch (err) {
    console.error("Ошибка при регистрации:", err);
    setError("Ошибка сервера или сеть недоступна");
  }
};

return (
  <div className="min-h-screen flex flex-col items-center
justify-center bg-gray-100">
    <div className="bg-white p-8 rounded shadow-md w-full max-w-
md">
      <h2 className="text-2xl font-bold mb-6 text-
center">Регистрация</h2>
      <form onSubmit={handleRegister} className="space-y-4">
        <div>
          <label className="block mb-1">Email</label>
          <input
            type="email"
            required
            value={email}
            onChange={(e) => setEmail(e.target.value)}
            className="w-full border rounded px-3 py-2
focus:outline-none focus:ring"
          />
        </div>
        <div>
          <label className="block mb-1">Пароль</label>
          <input
            type="password"
            required
            value={password}
            onChange={(e) => setPassword(e.target.value)}
            className="w-full border rounded px-3 py-2
focus:outline-none focus:ring"
          />
        </div>
        <button
          type="submit"
          className="bg-blue-600 text-white w-full py-2 rounded"
        >
          Зарегистрироваться
        </button>

```

```

        </form>
      </div>
    </div>
  );
}

export default RegistrationPage;

```

Листинг А.21 – Файл «TestPage.jsx»

```

import React, { useEffect, useState } from "react";
import axios from "axios";
import { useNavigate } from "react-router-dom";

function TestPage() {
  const [questions, setQuestions] = useState([]);
  const [currentIndex, setCurrentIndex] = useState(0);
  const [answers, setAnswers] = useState({});
  const [loading, setLoading] = useState(true);
  const [result, setResult] = useState(null);

  const navigate = useNavigate();

  useEffect(() => {
    axios
      .get("http://localhost:8080/api/questions")
      .then((res) => {
        setQuestions(res.data);
        setLoading(false);
      })
      .catch((err) => {
        console.error("Ошибка загрузки вопросов", err);
        setLoading(false);
      });
  }, []);

  const handleFinish = async () => {
    try {
      const response = await
    axios.post("http://localhost:8080/api/submit-test", {
      userId: 1, // пока захардкодим, позже подставим реального
пользователя
      selectedOptionIds: Object.values(answers),
    });
    setResult(response.data);
  } catch (err) {
    console.error("Ошибка отправки результата:", err);
    alert("Ошибка при отправке теста");
  }
};

  if (loading) return <div className="p-4">Загрузка
вопросов...</div>;
  if (!questions.length) return <div className="p-4">Вопросы не

```

```

найденны</div>;

const currentQuestion = questions[currentIndex];

return (
  <div className="max-w-3xl mx-auto p-4">
    <h1 className="text-2xl font-bold mb-4 text-center">Профориентационный тест</h1>

    {!result && (
      <>
        <div className="mb-6 p-4 border rounded shadow">
          <p className="font-semibold mb-2">
            Вопрос {currentIndex + 1} из {questions.length}:
          </p>
          <p className="mb-4">{currentQuestion.questionText}</p>
          <ul>
            {currentQuestion.options.map((opt) => (
              <li key={opt.id} className="mb-2">
                <label>
                  <input
                    type="radio"
                    name={`question-${currentQuestion.id}`}
                    value={opt.id}
                    checked={answers[currentQuestion.id] ===
opt.id}
                    onChange={() =>
setAnswers({ ...answers,
[currentQuestion.id]: opt.id })
                  }
                  className="mr-2"
                />
                {opt.optionText}
              </label>
            </li>
          </ul>
        </div>

        <div className="text-center">
          {currentIndex < questions.length - 1 ? (
            <button
              className="bg-blue-600 text-white px-6 py-2
rounded"
              onClick={() => setCurrentIndex(currentIndex + 1)}
              disabled={!answers[currentQuestion.id]} // не даём
перейти без ответа
            >
              Следующий вопрос
            </button>
          ) : (
            <button
              className="bg-green-600 text-white px-6 py-2

```

```

rounded"
        onClick={handleFinish}
        disabled={!answers[currentQuestion.id]} // не даём
завершить без ответа
    >
        Завершить тестирование
    </button>
    )}
</div>
</>
)}

{result && (
    <div className="mt-8 p-6 bg-green-100 border border-green-
400 rounded shadow">
        <h2 className="text-xl font-bold mb-3">Результат
теста</h2>
        <p className="text-lg">{result}</p>
        <div className="mt-4 text-center">
            <button
                onClick={() => navigate("/")}
                className="bg-blue-600 text-white px-4 py-2 rounded"
            >
                На главную
            </button>
        </div>
    </div>
)}
</div>
);
}

```

```
export default TestPage;
```

Листинг А.22 – Файл «App.js»

```

import React from "react";
import { BrowserRouter as Router, Routes, Route } from "react-
router-dom";
import HomePage from "../pages/HomePage";
import RegistrationPage from "../pages/RegistrationPage";
import LoginPage from "../pages/LoginPage";
import TestPage from "../pages/TestPage";
import ProfilePage from "../pages/ProfilePage";
import CareerPage from "../pages/CareerPage";
import LuckyPage from "../pages/LuckyPage";
import RapperCareerPage from "../pages/RapperCareerPage";

function App() {
    return (
        <Router>
            <Routes>
                <Route path="/" element={<HomePage />} />
            </Routes>
        </Router>
    );
}

```

```

    <Route path="/register" element={<RegistrationPage />} />
    <Route path="/login" element={<LoginPage />} />
    <Route path="/test" element={<TestPage />} />
    <Route path="/profile" element={<ProfilePage />} />
    <Route path="/career/:profession" element={<CareerPage />}
  />
    <Route path="/lucky" element={<LuckyPage />} />
    <Route path="/career/Пәнеп" element={<RapperCareerPage />}
  />
  </Routes>
</Router>
);
}

export default App;

```

Приложение

Б (рекомендуемое)

Описание вариантов использования

Таблица Б.1 – Вариант использования «Зарегистрироваться»

Вариант использования	Зарегистрироваться
ID	5
Краткое описание	Пользователь указывает email и пароль, которые будут его идентифицировать.
Главные актёры	Пользователь
Второстепенные актёры	Нет
Предусловия	1. Логин не занят; 2. Логин проходит валидацию; 3. Пароль проходит валидацию.
Основной поток	1. Пользователь переходит на экран регистрации; 2. Пользователь указывает email; 3. Пользователь указывает пароль; 4. Пользователь жмет кнопку «Зарегистрироваться».
Постусловия	Пользователь зарегистрирован

Таблица Б.2 – Вариант использования «Авторизоваться»

Вариант использования	Авторизоваться
ID	6
Краткое описание	Пользователь указывает email и пароль, которые он использовал при регистрации
Главные актёры	Пользователь
Второстепенные актёры	Нет
Предусловия	1. Email ранее был зарегистрирован; 2. Пароль ранее был зарегистрирован.
Основной поток	1. Пользователь указывает email; 2. Пользователь указывает пароль.
Постусловия	Пользователь авторизировался

Таблица Б.3 – Вариант использования «Перейти на страницу «Мне повезёт»»

Вариант использования	Перейти на страницу «Мне повезёт»
ID	7
Краткое описание	Пользователь переходит на страницу «Мне повезёт»
Главные актёры	Пользователь
Второстепенные актёры	Нет
Предусловия	Пользователь находится на главной странице
Основной поток	1. Пользователь жмет на кнопку «Мне повезёт»; 2. Пользователь читает шуточное описание профессии;
Постусловия	Нет