

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 09.03.04 «Программная инженерия»

Направленность Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

**Информационная система обработки заявок на
стажировки в ИТ-компаниях**

Студент

(подпись, дата)

Крутяков Даниил Вениаминович

(ФИО полностью)

Руководитель

(подпись, дата)

Такташкин Д.В.

(фамилия, инициалы)

Нормоконтролёр

(подпись, дата)

Такташкин Д.В.

(фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от 03.06.2026г. № 12)

Заведующий кафедрой

(подпись)

Козлов А.Ю.

(фамилия, инициалы)

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

Секретарь ГЭК

(подпись)

Попова Н.А.

(фамилия, инициалы)

Пенза, 2026

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
«26» декабря 2025 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Крутяков Даниил Вениаминович гр. 22ВП2 факультета ФВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Информационная система обработки заявок на стажировки в
ИТ-компаниях

Тема утверждена приказом ПГУ №1583/о от "30" декабря 2025 г.

3. Руководитель работы доцент кафедры «МОиПЭВМ», доцент, к.т.н., Такташкин Д.В.

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

1) *Назначение: автоматизированная информационная система отбора стажеров*

2) *Среда выполнения: Windows 11.*

Языковые средства разработки: Java, HTML, SQL, UML.

Инструментальные средства: IntelliJ IDEA, Visual Studio Code,

Spring Boot, PostgreSQL, Docker, Visual Paradigm 18.

*Технологии разработки: монолитная архитектура, REST API,
компонентный подход.*

3) *Основные функции приложения:*

а) регистрация и авторизация пользователей;

б) разграничение прав доступа на основе ролевой модели;

в) управление профилями кандидатов и выбор направления стажировки;

г) формирование и сопровождение многоэтапной заявки;

д) мониторинг поведенческих метрик кандидата;

е) централизованное управлениями заявками;

ж) фиксация кадровых решений.

4. Перечень подлежащих разработке вопросов:

1) Анализ предметной области и постановка задачи;

2) Проектирование;

3) Реализация;

4) Контроль качества программного обеспечения;

5) 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%	08.04.2026 – 13.04.2026	
2	Проектирование программных средств	20%	15.04.2026 – 25.04.2026	
3	Реализация программных средств	30%	26.04.2026 – 12.05.2026	
4	Тестирование программных средств	20%	13.05.2026 – 22.05.2026	
5	Оформление пояснительной записки	20%	23.05.2026 – 03.06.2026	

Дата выдачи задания «26» декабря 2025г.

Руководитель бакалаврской работы _____ 26.12.2025 г.
(подпись, дата)

Задание к исполнению принял студент _____ 26.12.2025 г.
(подпись, дата)

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

Реферат

Пояснительная записка содержит 86 страниц, 18 рисунков, 13 таблиц, 21 использованный источник и 3 приложения.

ИНФОРМАЦИОННАЯ СИСТЕМА, ОБРАБОТКИ ЗАЯВОК, УПРАВЛЕНИЯ ПРОФИЛЯМИ КАНДИДАТОВ, ФОРМИРОВАНИЯ ЗАЯВОК, КАДРОВОЕ РЕШЕНИЕ, JAVA, SPRING BOOT, HTML, POSTGRESQL, UML

Объект исследования – информационная система обработки заявок.

Цель выпускной квалификационной работы – спроектировать и разработать информационную систему обработки заявок на стажировку в ИТ-компаниях.

Технологии разработки – язык программирования Java и фреймворк Spring Boot для реализации серверной части, язык программирования JavaScript и HTML для реализации пользовательского интерфейса, система управления базами данных PostgreSQL, средство визуального проектирования Visual Paradigm, среда разработки IntelliJ IDEA и WebStorm.

Среда исполнения – Windows 11, web-браузер.

Результаты работы – разработана информационная система обработки заявок на стажировки в ИТ-компаниях, обеспечивающая автоматизацию многоэтапного отбора кандидатов, эвристический анализ программного кода на предмет использования искусственного интеллекта, а также централизованное управление заявками через панель специалиста.

					ПГУ 09.03.04 – 08ВР222.18 ПЗ					
Изм.	Лист	№ докум.	Подпись	Дата	Информационная система обработки заявок на стажировки в ИТ-компаниях Пояснительная записка		Лит.	Лист	Листов	
Разраб.		Крутяков Д.В.							4	86
Пров.		Такташкин Д.В.								
Реценз.										
Н. Контр.		Такташкин Д.В.								
Утв.		Козлов А.Ю.					Группа 22ВР2			

Содержание

Введение.....	7
1 Анализ предметной области и постановка задачи	8
1.1 Анализ предметной области	8
1.2 Обзор и анализ программных продуктов-аналогов	9
1.3 Постановка задачи на разработку приложения	11
1.4 Анализ требований	12
1.4.1 Анализ функциональных требований	12
1.4.2 Анализ нефункциональных требований	16
1.5 Планирование разработки и оценка бюджета.....	18
2 Проектирование системы	21
2.1 Архитектура системы	21
2.2 Проектирование структур данных	23
2.2.1 Проектирование концептуальной модели базы данных.....	23
2.2.2 Проектирование логической модели базы данных.....	25
2.3 Проектирование серверной части	27
2.4 Проектирование пользовательского интерфейса	29
3 Реализация программных средств информационной системы	32
3.1 Реализация слоя базы данных приложения.....	32
3.2 Реализация серверной части	35
3.3 Реализация слоя представления	38
4 Тестирование информационной системы.....	44
4.1 Обоснование выбора.....	44
4.2 Тестирование методом белого ящика	45
4.2.1 Покрытие операторов.....	45
4.2.2 Покрытие решений и условий.....	47
4.3 Анализ сложности кода	50
4.4 Метрики программного кода	53
4.5 Модульное тестирование	54

Заключение	56
Список использованных источников	58
Приложение А (справочное) Глоссарий	60
Приложение Б (обязательное) Листинг программы	62
Приложение В (рекомендуемое) Листинг модульных тестов	85

Введение

В условиях высокой конкуренции на рынке информационных технологий устойчивое развитие компании в значительной степени зависит от качества отбора профильных специалистов и стажеров. Для компаний важно не просто привлекать большое количество кандидатов, но и обеспечивать объективную оценку их реальных компетенций. Если у компании нет единого инструмента для первичной фильтрации, отбор стажеров превращается в хаотичный процесс. Специалисты теряют прозрачность, а эксперты тратят впустую часы на ручную проверку. Ситуация усугубляется тем, что сегодня соискатели массово используют нейросети для решения заданий. Именно поэтому так важно автоматизировать всю цепочку обработки заявок.

Современный найм состоит из множества шагов. Кандидата нужно направить на правильный трек, собрать его анкету, провести тест и принять написанный код. После этого требуется проверка на плагиат и вынесение итогового вердикта. Объединение всех этих разрозненных шагов в единый цифровой конвейер исключает потерю данных и сводит к нулю вероятность человеческой ошибки при переносе оценок.

В рамках данной выпускной квалификационной работы поставлена цель спроектировать и разработать информационную систему, предназначенную для обработки стажировочных заявок в ИТ-секторе.

В качестве практического результата работы ожидается получение полнофункциональной системы первичного отбора соискателей. Наличие специального модуля призвано гарантировать объективность оценки технических компетенций пользователя. Разработанный инструмент также направлен на упрощение внутреннего взаимодействия между рекрутерами и специалистами, формируя надежную базу для отбора лучших кандидатов на этап собеседования. В конечном итоге автоматизация этого рутинного процесса ощутимо сократит временные и финансовые затраты компании на организацию массовых стажировок.

1 Анализ предметной области и постановка задачи

1.1 Анализ предметной области

Предметная область выпускной квалификационной работы связана с процессами первичного отбора и оценки кандидатов на стажировки в ИТ-компаниях, где формирование кадрового резерва должно опираться на прозрачную систему тестирования, объективную оценку компетенций и строгую верификацию самостоятельности выполнения заданий. В условиях постоянного обновления технологий, инструментов разработки и массового распространения больших языковых моделей (LLM), особую значимость приобретает не только привлекательность компании на рынке труда, но и наличие автоматизированного механизма оценки реального уровня специалиста, фильтрации недобросовестных соискателей и планирования этапов собеседований. Одним из основных инструментов такого отбора выступает многоэтапная система оценки, в рамках которой фиксируются анкетные данные кандидата, результаты теоретического тестирования с контролем времени и метрики эвристического анализа написанного программного кода. В дальнейшем в тексте выпускной квалификационной работы искусственный интеллект может обозначаться сокращением ИИ.

Компетенция – совокупность знаний, умений, навыков и способов деятельности, необходимых для успешного прохождения технического отбора и выполнения профильных задач в определенной ИТ-области [1].

Эвристический анализ кода – метод проверки исходного текста программы, основанный на поиске специфических структурных, лингвистических и поведенческих аномалий (паттернов), применяемый для выявления фактов машинной генерации без использования предварительно обученных полновесных моделей машинного обучения [2].

Первичный ИТ-рекрутинг – начальный этап процесса подбора персонала, направленный на сбор структурированной информации о кандидатах (включая

Hard и Soft skills), автоматизированную оценку их базовых технических навыков и отсеивание нерелевантных соискателей до этапа проведения интервью [3].

Анализ предметной области показывает, что процессы массового отбора кандидатов на ИТ-стажировки являются сложными, высоконагруженными и уязвимыми к современным методам фальсификации результатов (в частности, автоматизированной генерации ответов нейросетями). Это обуславливает необходимость создания единой информационной системы, которая обеспечит централизованное хранение данных соискателей, поддержку многоэтапных бизнес-процессов тестирования, защиту от жульничества, жесткое разграничение прав доступа между участниками и формирование аналитической отчетности для принятия кадровых решений.

1.2 Обзор и анализ программных продуктов-аналогов

Перед разработкой информационной системы обработки заявок на стажировки целесообразно рассмотреть уже существующие программные решения в данной области. Такой анализ позволяет определить, какие функции чаще всего встречаются в подобных платформах технического рекрутинга, какие подходы используются для организации тестирования, написания кода и поддержки управленческих решений со стороны специалистов. Кроме того, сравнение аналогов помогает выделить недостатки (в частности, уязвимость перед нейросетями), которые необходимо учесть при проектировании собственной системы.

Для сравнения были определены критерии, отражающие основные задачи разрабатываемой информационной системы:

- наличие встроенного в браузер редактора программного кода;
- проведение теоретического тестирования с жестким контролем времени;
- автоматический эвристический анализ кода на выявление использования генеративного ИИ;

- поддержка расширенного анкетирования кандидатов (опыт, зарплатные ожидания, готовность к переезду);
- разграничение доступа по ролям;
- наличие сводной панели для специалиста с фиксацией кадровых решений.

Для оценивания аналогов была выбрана следующая шкала:

- функционал полностью отсутствует (0 баллов);
- функционал реализован частично (1 балл);
- функционал полностью реализован (2 балла).

Для сравнения были выбраны следующие системы:

- HackerRank;
- Яндекс.Контест;
- Codility.

Сравнение перечисленных выше систем представлено в таблице 1.

Таблица 1 – Сравнительный анализ

Критерий	HackerRank	Яндекс.Контест	Codility
Наличие встроенного в браузер редактора программного кода	2	2	2
Проведение теоретического тестирования с контролем времени	2	2	2
Поддержка расширенного анкетирования кандидатов	1	0	1
Наличие сводной панели для специалиста	2	1	1
Разграничение доступа по ролевой модели	2	2	2
Автоматический эвристический анализ кода на использование генеративного ИИ	1	0	1

Проведенное сравнение показывает, что существующие решения обладают развитым функционалом в области проверки алгоритмических навыков. Наиболее полно технические критерии представлены в платформе HackerRank,

так как она охватывает среду выполнения кода, базовое профилирование и систему прокторинга. Система «Яндекс.Контекст» и сервис Codility отлично подходят для проведения олимпиад и решения задач, однако в большей степени ориентированы на автопроверку ответов, чем на полный цикл собеседования с первичным анкетированием.

Главным недостатком всех рассмотренных систем является слабое или полное отсутствие защиты от использования современных больших языковых моделей – встроенные антиплагиаты ищут совпадения с уже существующим кодом других участников, но не анализируют эвристические и лингвистические маркеры уникально сгенерированного нейросетью ответа.

Разрабатываемая информационная система должна учитывать сильные стороны рассмотренных аналогов (удобный редактор кода, таймеры), но при этом предоставлять специализированный модуль Анти-ИИ и глубокую интеграцию с процессами HR-отдела ИТ-компаний. Полученные результаты анализа используются при формировании требований к системе и дальнейшем проектировании ее функциональности.

1.3 Постановка задачи на разработку приложения

На основе анализа предметной области сформирована цель: спроектировать и разработать информационную систему обработки заявок на стажировку в ИТ-компаниях. Для её выполнения требуется решить следующие задачи:

- спроектировать архитектуру информационной системы, включая моделирование бизнес-процессов и структуры базы данных;
- осуществить реализацию серверной и клиентской частей системы с внедрением алгоритмов эвристического анализа;
- провести тестирование разработанного программного обеспечения, включая проверку надежности алгоритмов детектирования ИИ;
- подготовить пояснительную записку, содержащую описание этапов разработки и результаты тестирования системы.

С учетом реализуемых бизнес-процессов в системе выделены две основные роли пользователей:

- кандидат (студент) – осуществляет регистрацию, выбирает направление стажировки, заполняет расширенную анкету, проходит тестирование с ограничением по времени, выполняет практическое задание во встроенном редакторе и отслеживает текущий статус своей заявки;
- HR-специалист (администратор) – работает с панелью управления, просматривает сводную таблицу всех заявок, изучает детализированные профили кандидатов (включая лог ответов на тест и развернутый аналитический отчет эвристического модуля оценки кода), а также принимает итоговое кадровое решение по каждому соискателю.

Полученный перечень задач отражает ключевые бизнес-процессы первичного автоматизированного отбора стажеров в ИТ-компании и является фундаментальной основой для дальнейшего проектирования информационной системы.

1.4 Анализ требований

1.4.1 Анализ функциональных требований

На основе определенных ролей пользователя и выполняемых задач была разработана диаграмма вариантов использования.

Диаграмма вариантов использования служит для выявления функциональных требований к системе и иллюстрирует стандартные взаимодействия между пользователями и системой [4]. Она позволяет определить, какие действия будут выполнять пользователи, какие варианты использования являются основными, а какие расширяют или дополняют другие сценарии.

Так как в разрабатываемой системе предусмотрено две категории пользователей с разными правами доступа, диаграммы вариантов использования были подготовлены для каждой роли.

На рисунке 1 представлена диаграмма вариантов использования для роли «Кандидат».



Рисунок 1 – Диаграмма вариантов использования для роли «Кандидат»

Для этой роли главным вариантом является подача заявки на стажировку.

На рисунке 2 изображена диаграмма для роли «Администратор».

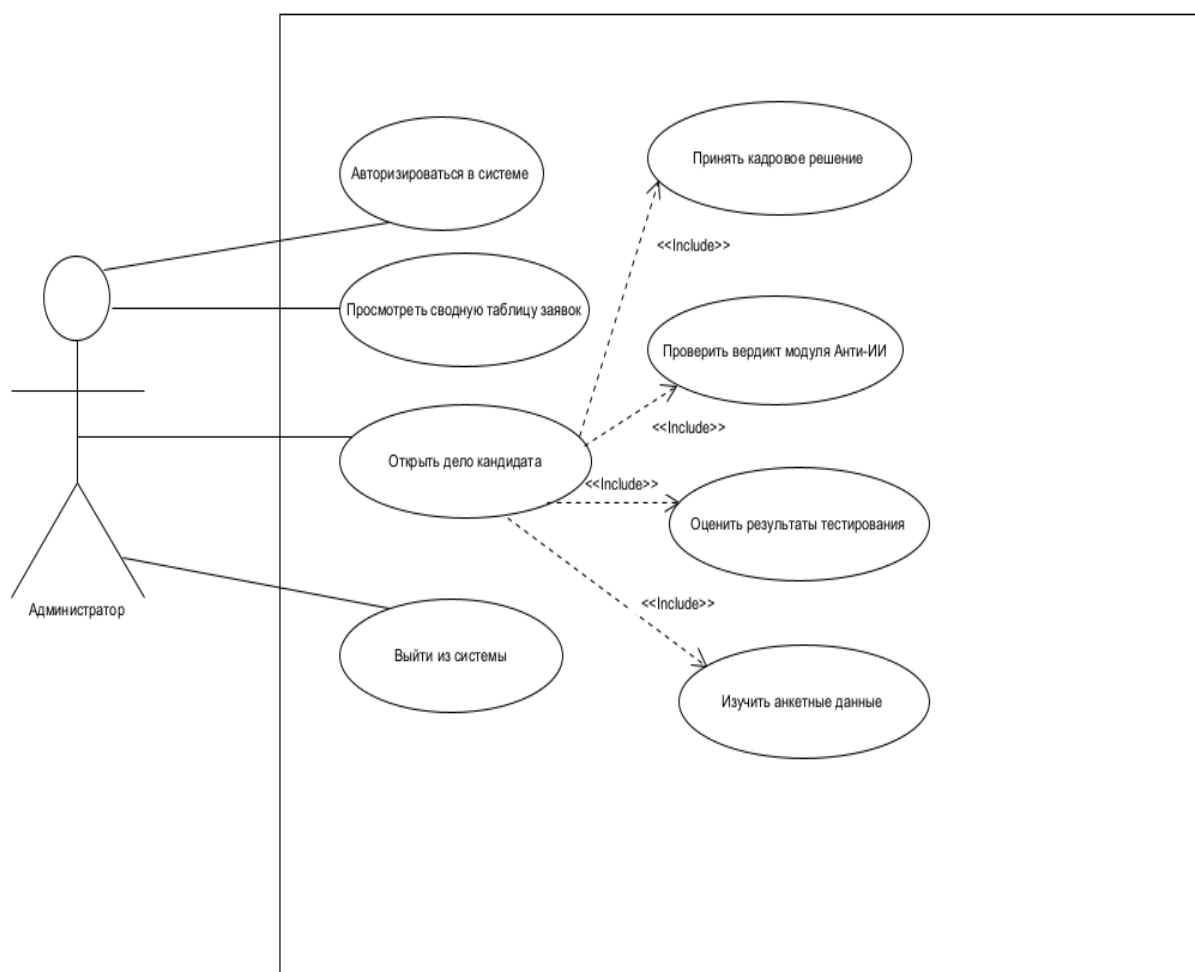


Рисунок 2 – Диаграмма вариантов использования для роли «Администратор»

Для более детального описания ключевых функций наиболее значимых сценариев применения были подготовлены их спецификации.

Сценарий варианта использования «Подать заявку на стажировку» представлен в таблице 2.

Таблица 2 – Вариант использования «Подать заявку на стажировку»

Прецедент: Подать заявку
ID: 1
Краткое описание: Кандидат заполняет расширенную анкету и проходит профильный тест
Главный актер: Кандидат
Второстепенный актер: Нет
Предусловие:
1. Пользователь авторизован в системе;
2. Находиться на главной странице выбора направлений.

Продолжение таблицы 2

Основной поток:
1. Выбирает профильное направление стажировки;
2. Заполняет анкетные данные;
3. Сохраняет анкету и инициирует начало теоретического тестирования;
4. Отвечает на вопросы теста в рамках отведенного времени;
5. Отправляет форму, система рассчитывает и сохраняет итоговый балл.
Постусловие:
1. Анкета и баллы за тест сохранены в базе данных;
2. Переход к следующему этапу

Сценарий варианта использования «Написать программный код» представлен в таблице 3.

Таблица 3 – Вариант использования «Написать программный код»

Прецедент: Написать программный код
ID: 2
Краткое описание: Кандидат пишет программный код, который проверяется эвристическим модулем на генерацию нейросетями
Главный актер: Кандидат
Второстепенный актер: Система
Предусловие: Успешно пройден теоретический этап тестирования
Основной поток:
1. Просматривает условие задачи для выбранного направления;
2. Пишет программный код;
3. Нажимает кнопку «Отправить код на проверку».
Постусловие:
1. Практическое задание сохранено;
2. Сформирован развернутый отчет о проверке на ИИ для администратора.

Сценарий варианта использования «Принять кадровое решение» представлен в таблице 4.

Таблица 4 – Вариант использования «Принять кадровое решение»

Прецедент: Принять кадровое решение
ID: 3
Краткое описание: Администратор анализирует данные кандидата и выносит итоговое решение
Главный актер: Администратор
Второстепенный актер: Нет
Предусловие:
1. Администратор авторизован в системе;
2. Находится в панели управления.

Продолжение таблицы 4

Основной поток: 1. Просматривает сводную таблицу всех поступивших заявок; 2. Открывает детализированный профиль выбранного кандидата; 3. Изучает результаты кандидата; 4. Принимает решение и нажимает кнопку «Пригласить на собеседование».
Постусловие: 1. Анкета кандидата и баллы за тест сохранены в базе данных; 2. Открыт доступ к следующему этапу (практическому заданию).

Спецификации вариантов использования позволяют описать логику взаимодействия пользователя с системой на уровне отдельных сценариев. В них фиксируются наименование и идентификатор прецедента, краткое описание его назначения, действующие лица, предусловия, основной поток выполнения, возможные альтернативные потоки и постусловия. Такое представление дает возможность структурированно описать поведение системы при выполнении ключевых операций (например, при прохождении тестирования или эвристическом анализе кода) и служит основой для последующего проектирования программных компонентов и сценариев тестирования.

В ходе анализа функциональных требований были определены основные варианты использования разрабатываемой системы. Для самых важных сценариев подготовлены спецификации, позволяющие описать последовательность действий пользователя и реакцию системы при выполнении соответствующих операций.

1.4.2 Анализ нефункциональных требований

Нефункциональные требования определяют эксплуатационные характеристики системы и имеют тоже важное критическое значение. Согласно подходу к разработке требований, нефункциональные требования определяют ограничения и характеристики качества, которым должна соответствовать система при выполнении своих функций. Кроме того, при оценке качества программного обеспечения обычно учитываются следующие характеристики:

- безопасность – система должна обеспечивать защищенный доступ к персональным данным кандидатов, результатам тестирования и исходному коду, для этого предусматривается авторизация пользователей;
- производительность – система должна обеспечивать быструю обработку поступающих заявок, бесперебойную работу клиентских таймеров тестирования и быстрый анализ присланного программного кода;
- надежность – система должна правильно сохранять данные и предотвращать потерю информации при прохождении многоэтапного отбора;
- сопровождаемость – архитектура системы обязана позволять дорабатывать отдельные и добавлять новые профильные направления;
- оперативная память (не менее 4 ГБ);
- процессор – не менее 2-х вычислительных ядер;
- операционная система – поддержка семейства Linux или Windows, обеспечивающих стабильную работу JVM;
- дисковое пространство – система должна иметь минимум 5 ГБ свободного места.

Также разработанная система должна корректно воспроизводиться на актуальных версиях следующих браузеров: Opera, Firefox, Safari, Microsoft Edge, Яндекс Браузер.

Серверная часть реализуется на языке Java с использованием фреймворка Spring Boot [5]. Данный фреймворк применяется для создания надежных веб-приложений и позволяет сократить объем типовой настройки проекта за счет готовых механизмов конфигурации.

Для постоянного хранения данных используется СУБД PostgreSQL [6]. Эта система управления базами данных идеально подходит для работы со структурированными данными и поддерживает механизмы обеспечения целостности. Для обеспечения переносимости и упрощения развертывания системы используется технология контейнеризации Docker (с применением конфигурации docker-compose).

Пользовательский интерфейс (клиентская часть) реализован с применением механизма серверного рендеринга на базе шаблонизатора Thymeleaf [7] в связке с современным CSS-фреймворком Tailwind CSS для обеспечения адаптивной и эстетичной верстки. Для реализации функционала написания кода непосредственно в браузере интегрирована специализированная библиотека CodeMirror [8], предоставляющая возможности подсветки синтаксиса, нумерации строк и отслеживания событий буфера обмена (для нужд системы Анти-ИИ). Обмен данными между интерфейсом и сервером выполняется посредством протокола HTTP с использованием стандартных механизмов отправки веб-форм.

Перечисленные нефункциональные требования определяют качественные характеристики разрабатываемой информационной системы и должны строго учитываться при проектировании архитектуры, базы данных, серверной логики и пользовательского интерфейса.

1.5 Планирование разработки и оценка бюджета

Перед началом разработки информационной системы обработки заявок был составлен календарный план работ. Его назначение заключается в том, чтобы определить состав основных этапов проекта, установить примерные сроки их выполнения, распределить трудовые ресурсы и получить предварительную оценку стоимости разработки. Такое планирование позволяет заранее увидеть наиболее трудоемкие части проекта и определить общую продолжительность выполнения выпускной квалификационной работы.

Для подготовки календарного плана использовалась программа Project Libre. В ней были заданы этапы проекта, их длительность, даты начала и завершения, связи между работами и назначенный ресурс. В качестве исполнителя всех этапов указан ресурс «Программист». При планировании учитывалось, что работы выполнялись не в формате полного рабочего дня, а в течение нескольких часов в день, поэтому продолжительность этапов отражает плановые трудозатраты с учетом частичной ежедневной занятости [9]. На

основании этих данных была сформирована диаграмма Ганта, отражающая порядок выполнения работ и временные интервалы каждого этапа [10]. Для построения диаграммы Ганта был заполнен список задач, затем был получен календарный график выполнения работ, который показан на рисунке 3.

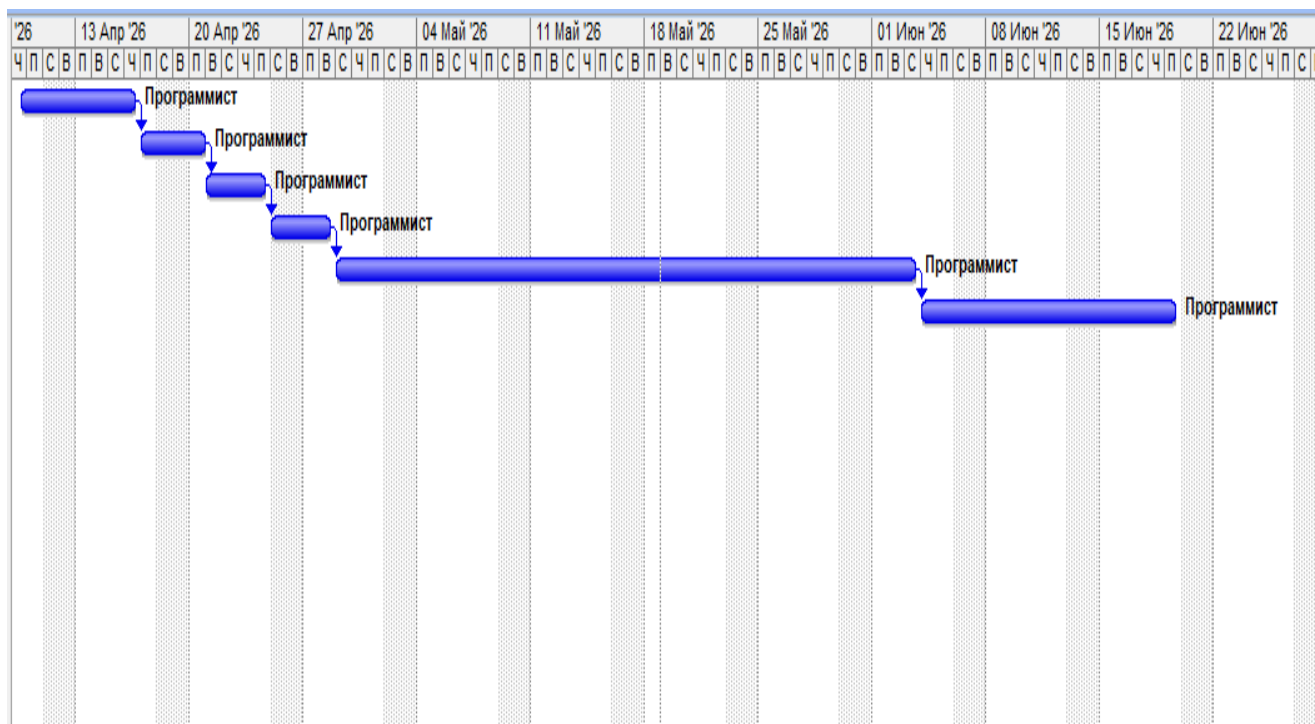


Рисунок 3 – Календарный график выполнения работ

В календарный план были включены работы, связанные как с исследовательской частью, так и с непосредственной разработкой программного продукта. К ним относятся анализ предметной области, определение функциональных и нефункциональных требований, проектирование архитектуры, разработка информационной системы и оформление пояснительной записки. Согласно плану, выполнение проекта начинается 26 марта 2026 года, а завершение работ запланировано на 19 июня 2026 года.

Наиболее продолжительным этапом является разработка информационной системы. Вторым по продолжительности этап является оформление пояснительной записки, так как нужно подробно описать все этапы разработки и проектирования системы.

Для оценки бюджета была использована стоимость одного рабочего дня программиста, равная 5000 рублей. Стоимость каждого этапа определялась

путем умножения его продолжительности на дневную ставку исполнителя. Расчет затрат представлен в таблице 5.

Таблица 5 – Расчет затрат на разработку информационной системы

№	Этап работы	Длительность (рабочее время)	Затраты
1	Анализ предметной области	1,88 дней	9 400 руб.
2	Анализ функциональных требований	1 день	5000 руб.
3	Анализ нефункциональных требований	1 день	5000 руб.
4	Проектирование архитектуры системы	1 день	5000 руб.
5	Разработка информационной системы	9 дней	45000 руб.
6	Оформление пояснительной записки для ВКР	4 дня	20000 руб.

По результатам расчета общая плановая продолжительность выполнения проекта составляет 17,88 рабочих дней, а общая стоимость разработки оценивается в 89 400 рубля.

Таким образом, составленный календарный план позволил определить структуру работ, оценить продолжительность разработки и рассчитать ориентировочный бюджет проекта. Полученные данные используются для обоснования трудоемкости создания информационной системы карьерного развития сотрудников и подтверждают необходимость предварительного планирования при выполнении выпускной квалификационной работы.

2 Проектирование системы

2.1 Архитектура системы

В качестве фундаментального архитектурного паттерна для проектируемой информационной системы выбран монолитный подход на базе концепции MVC (Model-View-Controller) [11]. Использование монолитной архитектуры на данном этапе жизненного цикла программного продукта является наиболее рациональным инженерным решением. Подобный подход исключает сетевые задержки, характерные для межсервисного взаимодействия, кардинально упрощает обеспечение транзакционной целостности при записи результатов многоэтапного тестирования и снижает накладные расходы на развертывание инфраструктуры.

В рамках паттерна MVC приложение логически разделяется на три взаимосвязанных уровня. Уровень моделей (Model) инкапсулирует в себе основную бизнес-логику системы, алгоритмы эвристического анализа и правила доступа к базе данных. В контексте разработанной платформы уровень моделей представлен программными сущностями (кандидаты, заявки, технологические треки) и сервисными классами, отвечающими за обработку поведенческих метрик. Уровень представления (View) отвечает за генерацию пользовательского интерфейса. В данном проекте он реализован с использованием технологии серверной шаблонизации, обеспечивающей динамическую отрисовку интерактивных веб-страниц (включая веб-редактор кода и таймеры) с подстановкой актуальных данных из моделей. Контроллер (Controller) выступает связующим звеном: он маршрутизирует входящие HTTP-запросы от браузера пользователя, осуществляет первичную валидацию данных и делегирует выполнение вычислительных задач соответствующим сервисам. На рисунке 4 представлена диаграмма развертывания отражающая разработанную информационную систему.

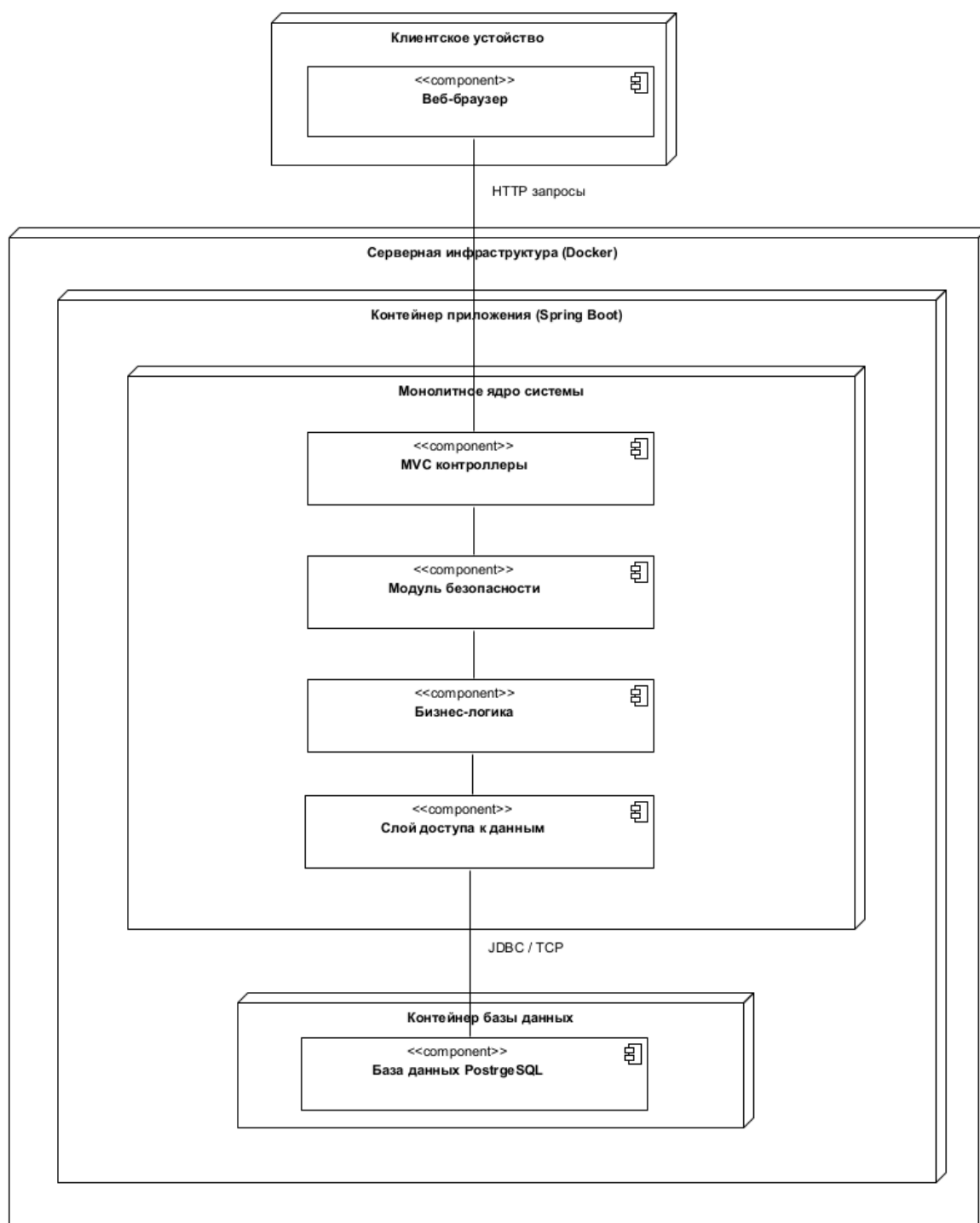


Рисунок 4 – Диаграмма развертывания

Пользователь взаимодействует с платформой стажировок через стандартный веб-браузер. В отличие от тяжелых клиентских SPA-приложений (Single Page Application), генерация пользовательского интерфейса происходит на стороне сервера с применением шаблонизатора Thymeleaf [12]. Интерактивные элементы интерфейса, в частности специализированная среда

написания исходного кода, реализованы за счет нативной интеграции легковесных библиотек (CodeMirror). Взаимодействие между клиентским устройством и сервером осуществляется по классическому протоколу HTTP средствами синхронных запросов и передачи веб-форм.

Серверное ядро системы построено на базе фреймворка Spring Boot. Данный монолитный узел инкапсулирует в себе всю управляющую бизнес-логику платформы. За обработку входящих соединений отвечают MVC-контроллеры, маршрутизирующие трафик в зависимости от ролевой модели пользователя (Spring Security). Внутри серверного приложения изолированно функционируют сервисы обработки тестов и эвристический анализатор программного кода (модуль Анти-ИИ).

Уровень персистентности показан единым экземпляром СУБД [13]. Взаимодействие серверного приложения с физическим хранилищем абстрагировано через спецификацию JPA с использованием ORM-фреймворка Hibernate, что обеспечивает безопасный маппинг объектной модели Java в реляционные таблицы по протоколу JDBC.

Для обеспечения кроссплатформенности и изоляции среды выполнения вся инфраструктура (веб-сервер и сервер базы данных) упакована в контейнеры средствами технологии Docker [14]. Таким образом, выбранная архитектура обладает высокой степенью связности внутренних компонентов, обеспечивая стабильное выполнение нефункциональных требований к производительности и безопасности.

2.2 Проектирование структур данных

2.2.1 Проектирование концептуальной модели базы данных

По результатам анализа предметной области и построения моделей бизнес-процессов была разработана концептуальная модель базы данных [15] информационной системы отбора стажеров.

В разработанной концептуальной модели ключевую роль занимает сущность «Пользователь», которая охватывает как соискателей (кандидатов), так

и HR-специалистов. Также модель отображает сведения о структуре отбора, включая технологические направления (треки), банк теоретических вопросов и практических задач. Центральным элементом бизнес-логики выступает сущность «Заявка», которая аккумулирует в себе все этапы взаимодействия кандидата с платформой. Такое построение модели позволяет прозрачно отслеживать весь путь соискателя от регистрации до вынесения финального кадрового решения с учетом эвристических проверок.

Графическое представление концептуальной модели базы данных приведено на рисунке 5.

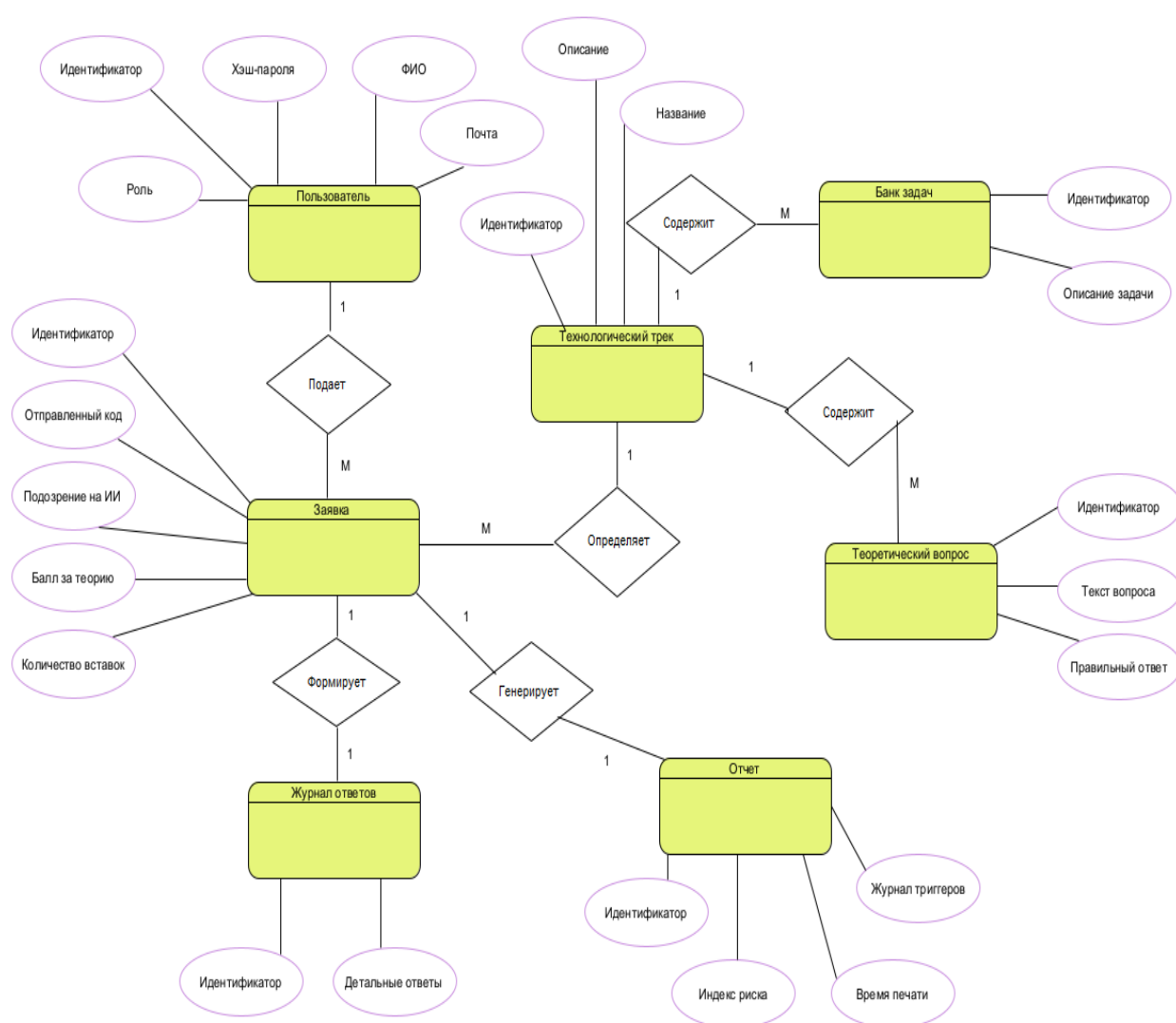


Рисунок 5 – Графическое представление концептуальной модели базы данных

На основе концептуальной модели базы данных было сформировано описание сущностей, представленное в таблице 6.

Таблица 6 – Сущности базы данных

Название сущности	Описание
Пользователь	Содержит учетные и анкетные данные зарегистрированных пользователей (кандидатов и HR-менеджеров), а также их ролевые доступы.
Технологический трек	Содержит перечень доступных направлений стажировки.
Заявка	Центральная сущность, агрегирующая результаты прохождения всех этапов отбора конкретным кандидатом по выбранному треку.
Теоретически вопрос	Содержит формулировки вопросов, варианты ответов и правильные значения для проведения первичного тестирования.
Банк задач	Содержит условия практических заданий для написания программного кода, распределенные по технологическим трекам.
Журнал ответов	Сохраняет детализированную историю ответов кандидата на вопросы теоретического теста для последующего аудита.
Отчет	Содержит результаты проверки программного кода модулем Анти-ИИ.

Концептуальная модель БД нарисована в среде Visual Paradigm 18 в нотации концептуального ER-моделирования [16].

2.2.2 Проектирование логической модели базы данных

На основании концептуальной модели была разработана логическая модель, отражающая состав данных, подлежащих хранению, и строгие правила их взаимосвязи. Логическая модель базы данных представлена на рисунке 6.

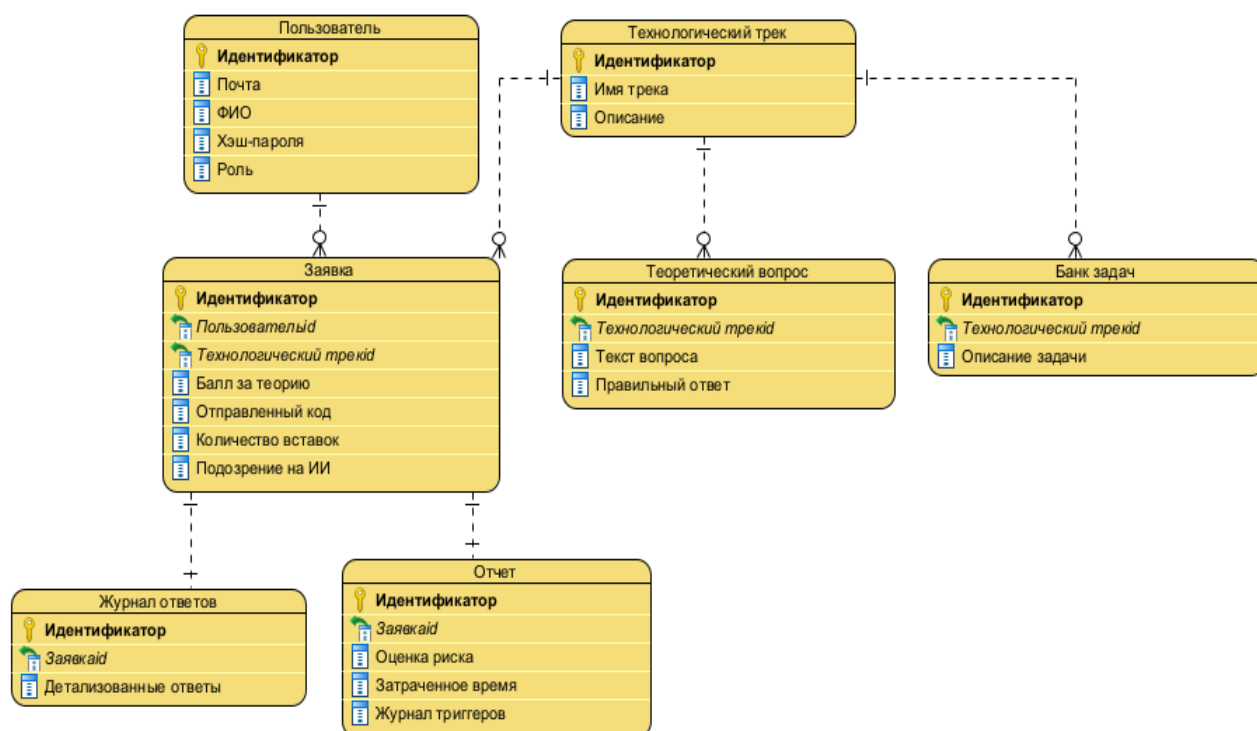


Рисунок 6 – Графическое представление логической модели базы данных

В логической модели БД определены связи между сущностями. Так, сущность «Пользователь» связана с сущностью «Заявка» связью типа «один-ко-многим», поскольку один соискатель потенциально может подать заявки на несколько технологических направлений в разные периоды времени. Сущность «Заявка» имеет связи «один-к-одному» с детализированными результатами: «Журнал ответов» и «Эвристический отчет», так как каждая попытка генерирует уникальный набор метрик.

Процесс хранения контента для тестирования представлен связями между сущностью «Технологический трек» и сущностями «Теоретический вопрос» и «Банк задач» (связь «один-ко-многим»). Это позволяет системе динамически формировать уникальные варианты заданий для каждого нового кандидата на основе выбранного направления.

Использование реляционной структуры и внешних ключей (Foreign Keys) обеспечивает строгую ссылочную целостность данных, не позволяя, например, создать заявку для несуществующего пользователя или назначить задачу из неверного технологического трека.

2.3 Проектирование серверной части

На основании результатов системного анализа, сформулированных функциональных ограничений и спроектированной инфологической структуры была выполнена разработка архитектуры программных модулей серверной части информационной системы [17]. На данном этапе ключевой фокус был направлен на декомпозицию системы на обособленные классы, распределение зон ответственности и формализацию алгоритмических сценариев взаимодействия. Архитектура программного средства базируется на строгом многоуровневом паттерне, включающем слои представления и маршрутизации (MVC-контроллеры), обработки бизнес-транзакций (сервисы) и взаимодействия с хранилищем (репозитории). Данная стратификация гарантирует слабое зацепление (low coupling) компонентов, что существенно упрощает сопровождение кодовой базы и создает надежную основу для дальнейшего расширения функциональных возможностей модуля отбора [18]. Взаимодействие между основными компонентами программного средства отражено на диаграмме классов, представленной на рисунке 7.

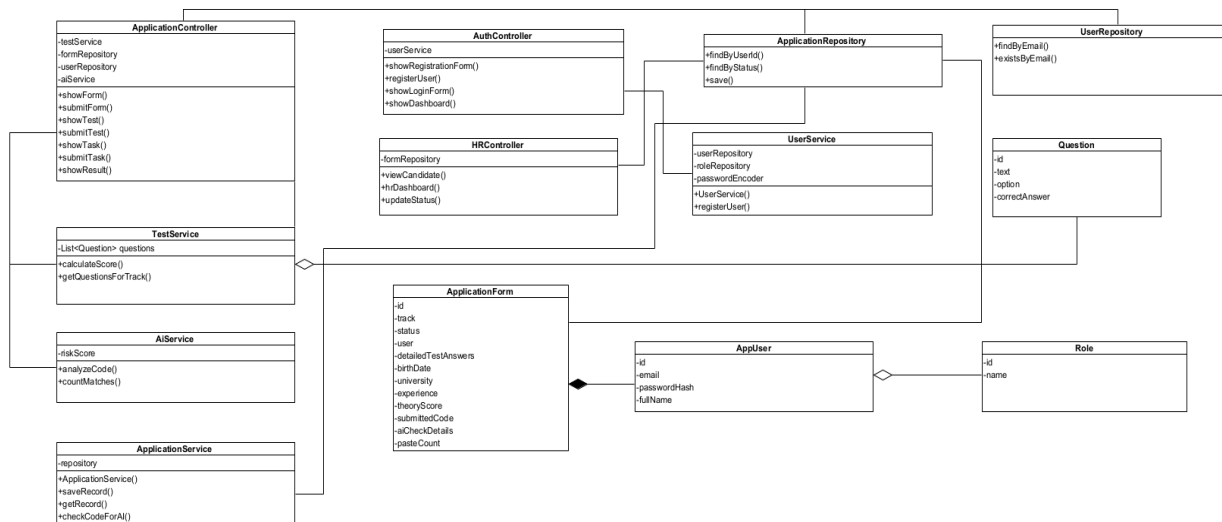


Рисунок 7 – Диаграмма классов основных компонентов программного средства

Представленная диаграмма классов визуализирует ключевые структурные узлы серверной абстракции разрабатываемого веб-приложения [19]. Схема детализирует топологию взаимодействия компонентов, ответственных за

многоэтапную обработку анкет соискателей, автоматизированный расчет баллов теоретического тестирования, проверку исходного кода на наличие признаков генерации нейросетями, а также фиксацию итоговых кадровых решений со стороны администратора.

Для описания сценария создания принятия кадрового решения была разработана диаграмма деятельности, представленная на рисунке 8. Данная диаграмма демонстрирует последовательность действий администратора и реакцию системы на каждом этапе выполнения операции.

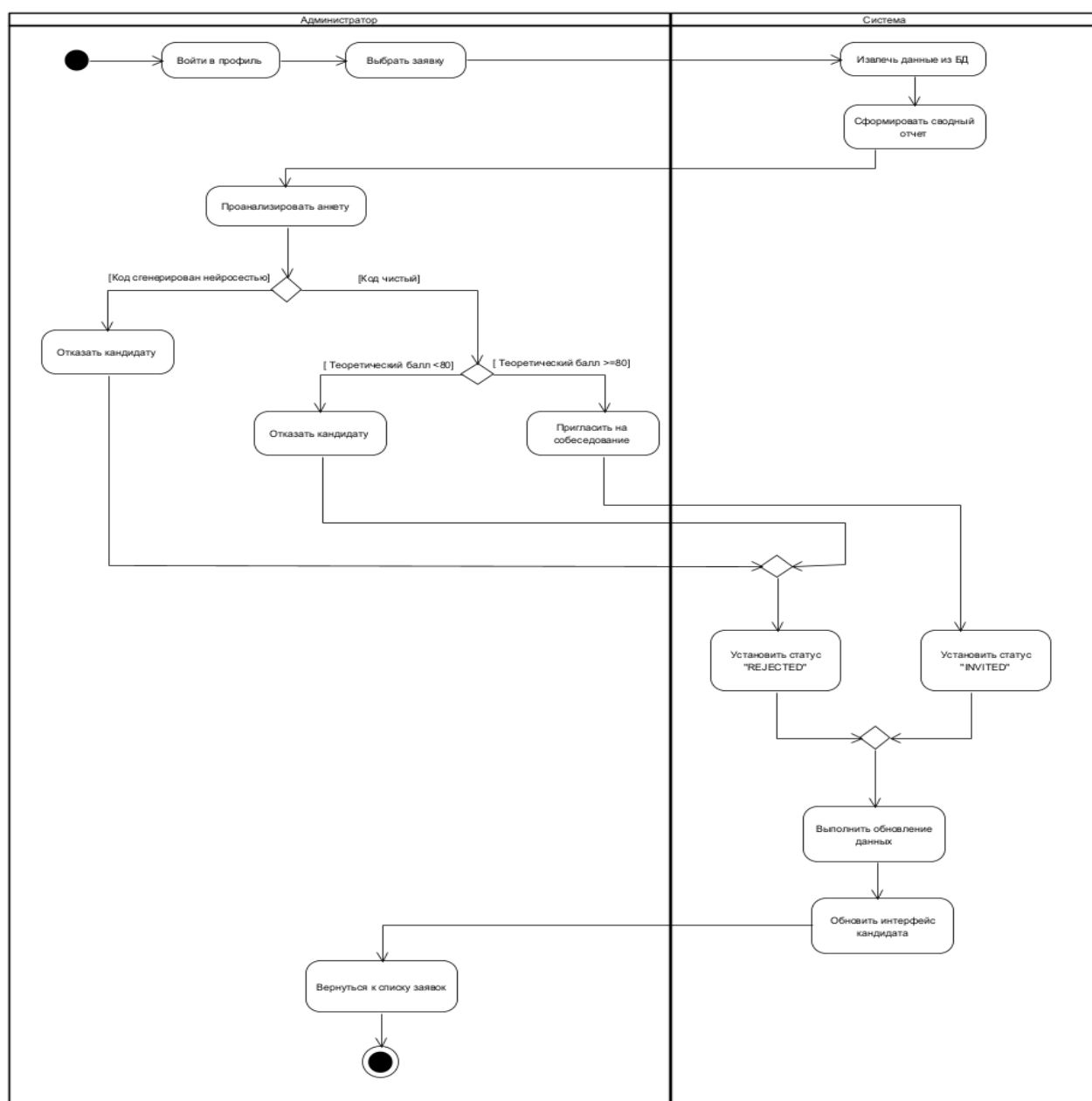


Рисунок 8 – Диаграмма деятельности для сценария «Принятия кадрового решения»

Рассматриваемый бизнес-процесс берет начало в момент инициации ответственным сотрудником кадровой службы процедуры верификации анкетных и тестовых данных в панели администрирования. При переходе к детальному просмотру профиля конкретного соискателя и последующем вынесении вердикта, интерфейсная часть генерирует и отправляет соответствующий POST-запрос на сервер. На данном этапе логика веб-приложения производит автоматическую проверку корректности поступающих данных и прав доступа пользователя. В случае обнаружения системных коллизий или нарушения целостности информации система формирует и выводит на экран предупреждение с указанием конкретных причин сбоя. При штатном выполнении сценария серверный компонент модифицирует состояние доменной сущности, сохраняет обновленные метрики в реляционном хранилище и возвращает клиентскому интерфейсу подтверждение успешной фиксации кадрового решения.

Таким образом, в рамках данного раздела было выполнено проектирование программного средства, включающее определение структуры его основных компонентов и формализацию ключевых пользовательских сценариев.

2.4 Проектирование пользовательского интерфейса

Пользовательский интерфейс разрабатываемой информационной системы должен быть интуитивно понятным, удобным в освоении и обеспечивать выполнение всех описанных вариантов использования. В процессе визуального проектирования основное внимание уделялось логичной компоновке экранов и детальной проработке состава интерактивных элементов взаимодействия.

При первом обращении к системе пользователь попадает на экран авторизации, внешний вид которого представлен на рисунке 9. На данной странице размещены только основные компоненты, необходимые для входа в систему: поля ввода электронной почты и пароля, кнопка подтверждения входа, а также краткое пояснение для пользователей, не имеющих доступа. Такое решение позволяет не перегружать стартовый экран лишними элементами и

сосредоточить внимание пользователя исключительно на выполнении процедуры аутентификации.

С возвращением!

Войдите, чтобы продолжить отбор

Email

example@mail.com

Пароль

.....

Войти

Нет аккаунта? [Зарегистрироваться](#)

Рисунок 9 – Прототип экрана авторизации

Как только процедура авторизации успешно завершена, система перенаправляет пользователя на главный экран приложения (рисунок 10). Данная страница фактически выполняет функцию рабочего дашборда и является центральным навигационным узлом для дальнейшего взаимодействия с платформой. Для удобства восприятия информации ключевые модули сгруппированы в виде визуальных карточек.

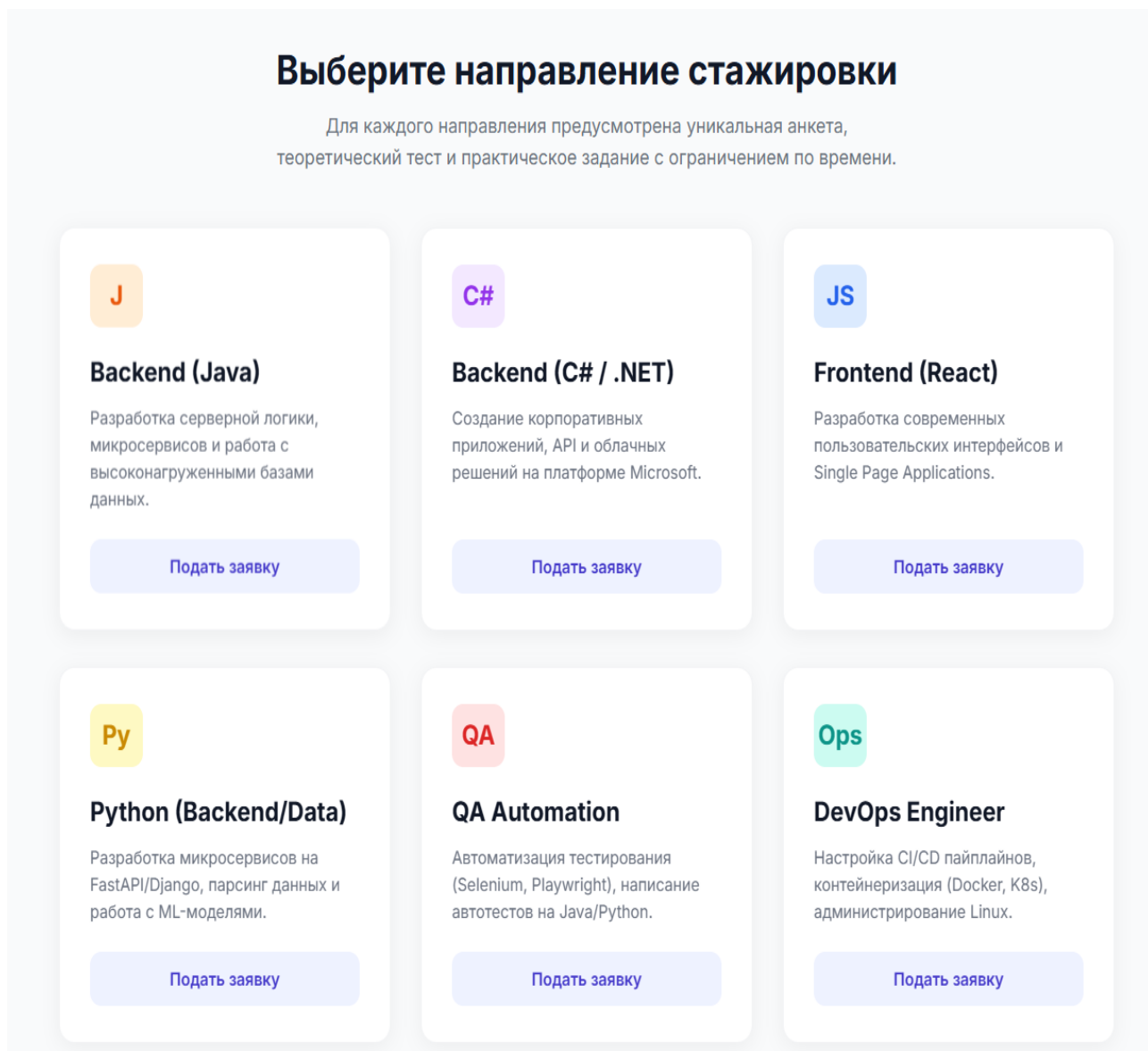


Рисунок 10 – Прототип главного экрана системы

Таким образом, проектирование пользовательского интерфейса было направлено на формирование максимально интуитивной, доступной и эргономичной среды взаимодействия с программным комплексом. Это было достигнуто за счет разработки прототипов ключевых экранов, отражающих основные сценарии работы пользователя. Процесс прототипирования позволил на ранних этапах визуализировать архитектуру клиентской части, продумать навигационную структуру и оптимизировать расположение функциональных элементов на страницах. Благодаря такому подходу удалось минимизировать количество пользовательских шагов, необходимых для выполнения целевых действий в системе.

3 Реализация программных средств информационной системы

3.1 Реализация слоя базы данных приложения

Для реализации слоя персистентности была выбрана реляционная СУБД PostgreSQL в связке с технологиями Spring Data JPA и Hibernate. В отличие от ручного версионирования SQL-скриптов, в разрабатываемой системе применен архитектурный подход «Code-First» (код первичен). При такой парадигме структурированное хранение данных и объектно-реляционное отображение (ORM) полностью управляются на уровне доменной модели приложения.

Автоматическая генерация схемы базы данных (DDL-инструкций) осуществляется ядром Hibernate при инициализации контекста Spring Boot. Настройки подключения к физическому хранилищу и стратегия обновления таблиц задаются в конфигурационном файле `application.properties`. Фрагмент конфигурационного файла представлен на рисунке 11.

```
# Основные настройки подключения к БД PostgreSQL
spring.datasource.url=jdbc:postgresql://localhost:5432/internship_db
spring.datasource.username=postgres
spring.datasource.password=root

# Указание диалекта для корректной генерации SQL-запросов
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect

# Автоматическое обновление схемы и вывод логов в консоль
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

Рисунок 11 – Фрагмент конфигурационного файла

В приведенном фрагменте настроек стоит отдельно отметить свойство `ddl-auto=update`. Для этапа активной разработки и прототипирования приложения эта директива имеет принципиальное значение. Благодаря ей Hibernate самостоятельно следит за тем, чтобы структура таблиц в базе данных соответствовала актуальным Java-сущностям. Если в программный код добавляется новое поле, фреймворк автоматически генерирует нужный SQL-

запрос на обновление схемы БД. При этом ранее сохраненная информация о заявках не удаляется и остается доступной для работы.

Центральным узлом доменной модели выступает сущность `ApplicationForm`, инкапсулирующая данные о заявке кандидата, результатах тестирования и метриках модуля Анти-ИИ. Фрагмент класса приведен в листинге 1.

Листинг 1 – Фрагмент доменной сущности `ApplicationForm`

```
public class ApplicationForm {

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

    @ManyToOne(fetch = FetchType.LAZY, optional = false)
    @JoinColumn(name = "user_id", nullable = false)
    private AppUser user;

    @Enumerated(EnumType.STRING)
    @Column(name = "track", nullable = false)
    private Track track;

    @Enumerated(EnumType.STRING)
    @Column(name = "status", nullable = false)
    private ApplicationStatus status;

    @Column(name = "theory_score")
    private Integer theoryScore;

    @Column(name = "submitted_code", columnDefinition = "TEXT")
    private String submittedCode;

    @Column(name = "paste_count")
    private Integer pasteCount;

    @Column(name = "is_ai_suspected")
    private Boolean isAiSuspected;
}
```

Использование аннотации `@Entity` делегирует фреймворку ответственность за трансляцию экземпляра класса в строку таблицы. Отдельного пояснения требует маппинг полей `track` (направление стажировки) и `status` (текущий этап). Они описываются перечислениями Java (`Enum`), а аннотация `@Enumerated(EnumType.STRING)` гарантирует сохранение значения в БД в

текстовом формате, а не по порядковому индексу, что нивелирует риск повреждения данных при расширении списка статусов в коде.

Для связи с родительской сущностью пользователя (AppUser) применен паттерн отложенной инициализации FetchType.LAZY. Ленивый способ загрузки гарантирует, что тяжеловесные графы пользовательских данных не извлекаются из СУБД одновременно с выборкой списка заявок, а инициализируются (через проху-объект) исключительно в момент фактического программного обращения к полю. Это решение радикально снижает потребление оперативной памяти сервера и оптимизирует время отклика при построении сводных таблиц в интерфейсе HR-специалиста.

Для реализации динамической фильтрации реестра кандидатов (по направлениям и статусам проверок) в слое доступа к данным (Repository) задействован механизм Specification API. Программная реализация динамической маршрутизации запросов изолирована в классе ApplicationSpecificationBuilder. Фрагмент кода приведен в листинге 2.

Листинг 2 – Фрагмент построения спецификации для фильтрации заявок

```
public ApplicationSpecificationBuilder withTrackEq(Track track) {
    if (track != null) {
        spec = spec.and(ApplicationSpecification.trackEq(track));
    }
    return this;
}

public ApplicationSpecificationBuilder withAiSuspected(Boolean
isSuspected) {
    if (isSuspected != null) {
        spec =
spec.and(ApplicationSpecification.aiSuspectedEq(isSuspected));
    }
    return this;
}
```

Применение спецификаций позволяет абстрагироваться от жесткого кодирования десятков статических методов в интерфейсе репозитория, обеспечивая генерацию SQL-условий (предикатов WHERE) «на лету» в зависимости от фактически переданных администратором параметров поиска.

Таким образом, спроектированный и реализованный слой доступа к данным обеспечивает надежное, транзакционно-безопасное и структурированное хранение информации о кандидатах. Применение парадигмы Code-First в связке с ORM-фреймворком Hibernate существенно оптимизировало процесс разработки за счет автоматизированного управления DDL-схемами.

3.2 Реализация серверной части

Реализация программных компонентов серверной части базируется на строгой многоуровневой архитектуре. В монолитном приложении выделяются REST-контроллеры, сервисы и репозитории. Подобная декомпозиция позволяет жестко разграничить зоны ответственности между компонентами, повысить читаемость исходного кода и обеспечить высокую ремонтпригодность системы при масштабировании функционала.

Слой контроллеров в разработанном приложении отвечает исключительно за прием HTTP-запросов, десериализацию параметров пути, инициализацию валидации входных данных и возврат HTTP-ответа клиенту [20]. Контроллер не содержит прикладной алгоритмической логики, делегируя ее выполнение сервисному уровню. В качестве примера можно рассмотреть конечную точку (endpoint) для получения детализированной информации о заявке кандидата со стороны администратора. Соответствующий метод контроллера ApplicationController приведен в листинге 3.

Листинг 3 – Фрагмент контроллера для получения деталей заявки

```
@PreAuthorize("hasRole('ROLE_HR')")
@GetMapping("/{application_id}")
public ApplicationDto
getApplicationDetails(@PathVariable("application_id") Long
applicationId) {
    return
applicationService.getApplicationDetails(applicationId);
}
```

Представленный фрагмент демонстрирует, что контроллер выполняет исключительно функции проксирования вызова к applicationService. Важной архитектурной особенностью является обеспечение безопасности на уровне

методов: аннотация `@PreAuthorize` интегрирована с контекстом Spring Security и блокирует обработку запроса, если сессия пользователя не обладает системной привилегией `ROLE_HR`.

Основной массив прикладной логики инкапсулирован в классах с аннотацией `@Service`. Задача сервиса состоит в координации обращений к базе данных, обработке исключительных ситуаций и подготовке данных для клиентской части. Пример реализации метода на уровне сервиса представлен в листинге 4.

Листинг 4 – Фрагмент сервисного метода получения данных о заявке

```
public ApplicationDto getApplicationDetails(Long id) {
    ApplicationForm entity = applicationRepository.findById(id)
        .orElseThrow(() -> new EntityNotFoundException(
            messageUtils.getMessage("error.not_found.application", id)
        ));

    return applicationMapper.toDto(entity);
}
```

Как видно из приведенного исходного кода, сервис сначала инициирует извлечение доменной сущности по суррогатному идентификатору. В случае отсутствия записи в реляционной таблице немедленно генерируется прерывание `EntityNotFoundException`, которое перехватывается глобальным обработчиком ошибок (`GlobalExceptionHandler`) для формирования корректного HTTP-статуса 404.

Критически важным аспектом безопасности является то, что сущность базы данных (`ApplicationForm`) никогда не передается клиентскому приложению напрямую. Прямой возврат `Entity` может привести к компрометации структуры базы данных и избыточной передаче служебных полей (например, хешей паролей связанных пользователей). Для трансляции ответа используется специализированный паттерн DTO (`Data Transfer Object`) в виде record-модели `ApplicationDto`, содержащей исключительно те агрегированные метрики (результаты тестов, вердикты Анти-ИИ), которые необходимы для отрисовки интерфейса.

Автоматизация маппинга полей между Entity и DTO реализована средствами кодогенератора MapStruct. Данный подход исключает написание рутинного шаблонного кода (boilerplate). Фрагмент интерфейса преобразователя приведен в листинге 5.

Листинг 5 – Фрагмент mapper-компонента (MapStruct)

```
@Mapper(componentModel = "spring")
public interface ApplicationMapper {

    @Mapping(target = "candidateId", source = "entity.user.id")
    @Mapping(target = "candidateFullName", source =
"entity.user.fullName")
    @Mapping(target = "track", source = "entity.track")
    @Mapping(target = "status", source = "entity.status")
    @Mapping(target = "theoryScore", source =
"entity.theoryScore")
    @Mapping(target = "isAiSuspected", source =
"entity.isAiSuspected")
    @Mapping(target = "pasteCount", source = "entity.pasteCount")
    ApplicationDto toDto(ApplicationForm entity);
}
```

На этапе компиляции проекта процессор аннотаций MapStruct автоматически генерирует имплементацию данного интерфейса, обеспечивая высокую скорость выполнения за счет использования прямых вызовов методов get/set вместо медленного механизма Java Reflection. В методе toDto() задаются явные правила проекции: в частности, идентификатор и ФИО кандидата извлекаются из связанной родительской сущности user, что демонстрирует бесшовную работу с графами объектов на сервисном слое.

Главным элементом проверки на честность стал эвристический сервис AiService. Его задача – находить следы использования нейросетей в написанном коде. В основе алгоритма лежит оценка трех параметров: поведения пользователя, структуры самого листинга и наличия специфических слов-маркеров. Вся эта логика собрана внутри метода analyzeCode. Сначала программа оценивает объем комментариев в файле. Если доля задокументированных строк превышает порог в 30%, система расценивает это как подозрительный признак, поскольку языковые модели (например, ChatGPT или Claude) склонны избыточно комментировать свои ответы. Дополнительно

текст проверяется на лексические совпадения с помощью регулярных выражений. Отдельный метод сканирует код на шаблонные фразы и стандартные названия (такие как `public class Main`), которые нейросети часто выдают по умолчанию. Если по итогам всех проверок сумма штрафных баллов достигает лимита в 65 очков, сервис помечает работу как подозрительную и передает детализированный отчет в панель HR-менеджера.

3.3 Реализация слоя представления

Интерфейс клиентской части разработан с использованием современных стандартов веб-разработки. Это нужно для наглядности пользователя и хорошего понимания ориентирования системы. Для формирования визуального стиля применялись каскадные таблицы стилей и модульные элементы разметки, что гарантирует единообразие пользовательского взаимодействия и удобство дальнейшего сопровождения кода. Основной упор при проектировании интерфейса был сделан на интуитивно понятное, поэтапное ведение кандидата по воронке отбора, что позволяет минимизировать когнитивную нагрузку и вероятность ошибок при вводе данных.

В самом начале сессии кандидату предлагается ознакомиться с доступными технологическими треками и указать желаемое направление стажировки. Этот выбор является критически важным шагом, так как именно он запускает всю дальнейшую бизнес-логику приложения. Опираясь на указанную специализацию, внутренний алгоритм платформы мгновенно фильтрует общий банк заданий. В результате система автоматически генерирует для соискателя персональный экзаменационный лист, состоящий исключительно из профильных вопросов и профильного кода. Разделение потока кандидатов на самом первом этапе позволяет избежать путаницы, исключает выдачу нерелевантных тестов и существенно упрощает дальнейшую аналитику результатов для технических специалистов компании. Иллюстрация рисунка направления стажировки представлен на рисунке 12.

Выберите направление стажировки

Для каждого направления предусмотрена уникальная анкета, теоретический тест и практическое задание с ограничением по времени.



Backend (Java)

Разработка серверной логики, микросервисов и работа с высоконагруженными базами данных.

Подать заявку



Backend (C# / .NET)

Создание корпоративных приложений, API и облачных решений на платформе Microsoft.

Подать заявку



Frontend (React)

Разработка современных пользовательских интерфейсов и Single Page Applications.

Подать заявку



Python (Backend/Data)

Разработка микросервисов на FastAPI/Django, парсинг данных и работа с ML-моделями.

Подать заявку



QA Automation

Автоматизация тестирования (Selenium, Playwright), написание автотестов на Java/Python.

Подать заявку



DevOps Engineer

Настройка CI/CD пайплайнов, контейнеризация (Docker, K8s), администрирование Linux.

Подать заявку

Рисунок 12 – Выбор направления стажировки

Следом идет процесс анкетирования, интерфейс которого показан на рисунке 13. Кандидат вносит подробную информацию о себе, при этом все поля проходят динамическую проверку еще на этапе ввода. Если формат данных ошибочен, форма просто не отправится. Такая предварительная фильтрация на стороне клиента надежно защищает сервер от обработки мусорных запросов.

The form is titled "Анкета: JAVA" in bold black text, with "JAVA" in blue. It contains several input fields and a submit button. The fields are arranged in a grid-like fashion. The first row has "Дата рождения" (Date of birth) and "ВУЗ" (University). The second row has "Ожидаемая заработная плата (руб.)" (Expected salary) and "Готовность к переезду" (Willingness to relocate). The third row has a single field for "Ссылка на GitHub (Рет-проекты)" (GitHub link). The fourth row has a single field for "Опыт работы" (Work experience). The fifth row has a single field for "Hard Skills". The sixth row has a single field for "Soft Skills". At the bottom is a large blue button with white text.

Анкета: JAVA

Дата рождения

ВУЗ

Ожидаемая заработная плата (руб.)

Готовность к переезду

Ссылка на GitHub (Рет-проекты)

Опыт работы

Hard Skills

Soft Skills

Сохранить и начать тест

Рисунок 13 – Интерфейс ввода данных кандидата

Модуль проверки теоретических знаний (рисунок 14) спроектирован с упором на объективность оценки. Чтобы соискатель не искал ответы в сторонних вкладках, скрипты браузера фиксируют любую потерю фокуса с рабочего окна. По окончании сеанса баллы мгновенно рассчитываются на бэкенде. Применение такого простого, но эффективного прокторинга сводит к минимуму возможность списывания.

Внимание: На выполнение теста отведено 10 минут. По истечении времени ваши ответы будут отправлены автоматически. Пожалуйста, не обновляйте страницу.

1. Какой метод используется для запуска нового потока?

- ☐ run()
- ☐ start()
- ☐ execute()
- ☐ init()

2. Что из этого является маркером интерфейса (Marker Interface)?

- ☐ Runnable
- ☐ Serializable
- ☐ Comparable
- ☐ Callable

3. Какая сложность поиска элемента в HashMap в лучшем случае?

- ☐ $O(N)$
- ☐ $O(\log N)$
- ☐ $O(1)$
- ☐ $O(N^2)$

4. Какой модификатор доступа делает переменную видимой только в пределах пакета и подклассов?

- ☐ private
- ☐ public
- ☐ protected
- ☐ default

Рисунок 14 – Интерфейс модуля теоретического тестирования

По завершении работы собранный массив поведенческих характеристик инкапсулируется вместе с исходным кодом и передается на сервер в рамках единого защищенного запроса. Такой скрытый сбор телеметрии не нарушает пользовательский опыт, однако обеспечивает модуль исчерпывающей фактурой для высокоточного выявления аномалий и фактов использования генеративного искусственного интеллекта.

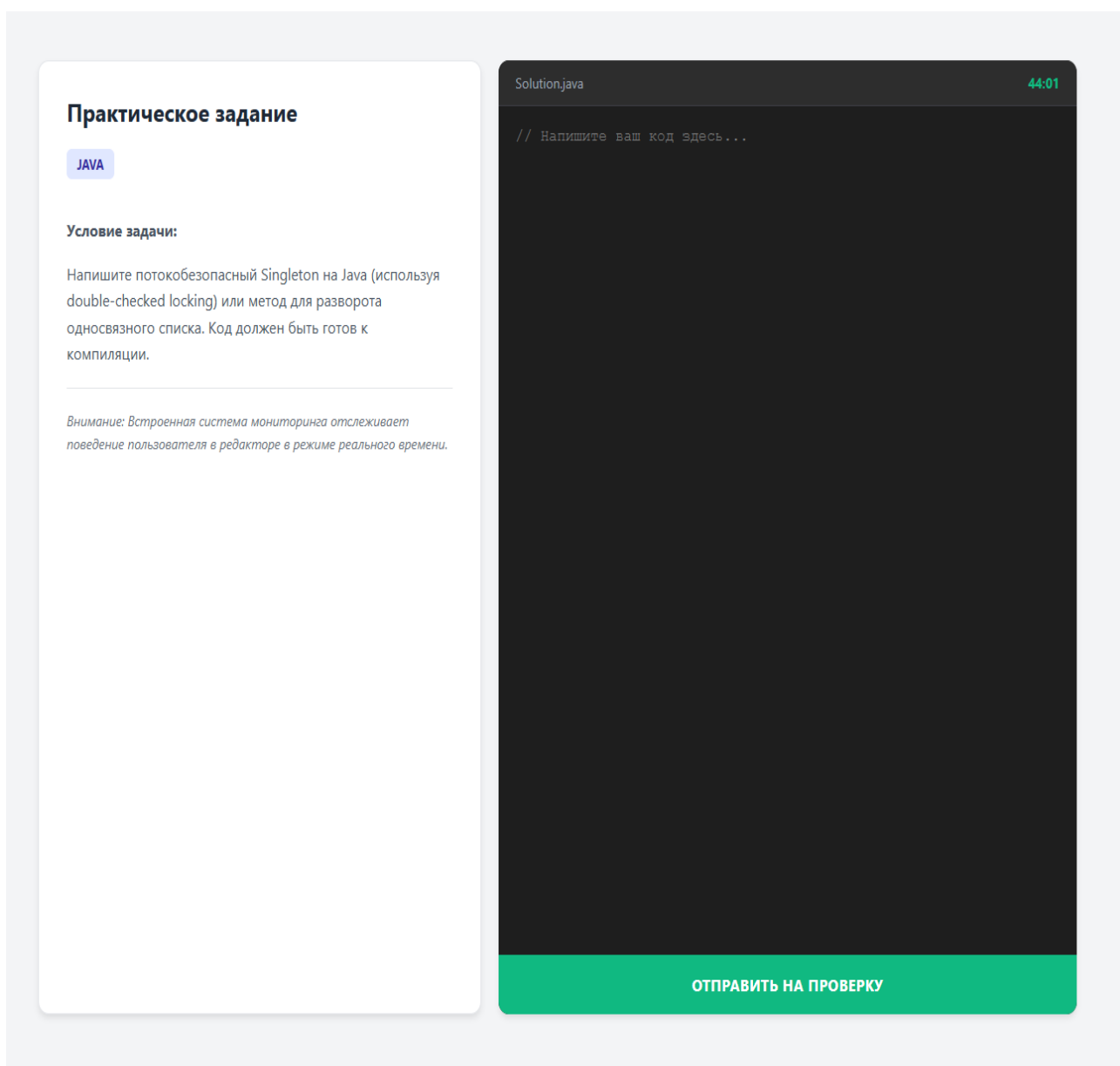


Рисунок 15 – Интерфейс встроенного редактора кода

Для сотрудников отдела кадров спроектирован отдельный аналитический дашборд, представленный на рисунке 16. В этом административном разделе администратор взаимодействует со сводным реестром заявок. Ключевым элементом таблицы выступает индикатор, где система выводит визуальное предупреждение при обнаружении признаков использования ИИ. Подобная визуализация результатов автоматизированного аудита существенно сокращает время ручной проверки и оптимизирует процесс принятия итогового кадрового решения. Утвержденный статус кандидата сохраняется в базе данных, завершая полный цикл обработки текущей заявки.

3. Практическое задание и Анти-ИИ проверка

⚠ ВНИМАНИЕ: СИСТЕМА ЗАБЛОКИРОВАЛА РЕШЕНИЕ (ВЫСОКИЙ РИСК ИИ)

Лог анализатора: КРИТИЧЕСКИЙ РИСК ИИ: - Когнитивная аномалия: задача 'решена' за 42 сек. - Сверхчеловеческая скорость ввода: 22,4 симв/сек. - Разовая вставка объемного блока кода (Ctrl+V). - Избыточное документирование (48% от объема кода). - Выявлен LLM-паттерн нейминга (избыточно длинные переменные). (Итоговый Risk Score: 100/100)

Счетчик подозрительных вставок (Ctrl+V): 1

ИСХОДНЫЙ КОД КАНДИДАТА:

```
/**
 * Функция debounce для ограничения частоты вызовов.
 * Паттерн гарантирует, что переданная функция выполнится только один раз.
 */
function debounce(functionToExecuteLater, delayTimeInMilliseconds) {
  // Переменная для хранения идентификатора таймера в замыкании
  let timeoutIdentifierForDebounce;

  // Возвращаем новую анонимную функцию-обертку
  return function (...argumentsForOriginalFunction) {

    // Очищаем предыдущий таймер, если функция вызвана снова до истечения задержки
    clearTimeout(timeoutIdentifierForDebounce);

    // Устанавливаем новый таймер для отложенного выполнения
    timeoutIdentifierForDebounce = setTimeout(() => {

      // Вызываем оригинальную функцию с правильным контекстом и аргументами
      functionToExecuteLater.apply(this, argumentsForOriginalFunction);

    }, delayTimeInMilliseconds);
  };
}
```

Текущий статус: **Ожидает проверки**

✓ Пригласить

✗ Отказать

Рисунок 16 – Панель управления с вердиктом системы

Подводя итог, можно сказать, что спроектированный слой представления удачно объединил в себе строгую серверную логику и сложную эвристику. В результате была получена эргономичная графическая оболочка, делающая процесс найма автоматизированным и легко контролируемым.

4 Тестирование информационной системы

4.1 Обоснование выбора

Обеспечение качества разрабатываемой информационной системы является критически важным этапом ее жизненного цикла. Программные сбои или алгоритмические неточности в системе подобного класса могут привести к серьезным бизнес-рискам: от необоснованного отказа талантливым соискателям (из-за ложных срабатываний детектора ИИ) до успешного прохождения проверки неквалифицированными кандидатами. В связи с этим для комплексной оценки корректности работы программного средства применяется совокупность взаимодополняющих подходов к тестированию.

В рамках разрабатываемой системы используются тестирование методом белого ящика, модульное тестирование, а также анализ метрик и сложности программного кода. Применение тестирования методом черного ящика позволяет проверить соответствие внешнего поведения системы функциональным требованиям без учета внутренней реализации программных компонентов. Для формализации такой проверки используются классы эквивалентности, таблицы решений и функциональные диаграммы. Классы эквивалентности позволяют сократить число тестовых случаев за счет объединения однотипных входных данных, таблицы решений удобны при проверке сочетаний условий и статусов, а функциональные диаграммы обеспечивают наглядное представление проверяемых функций системы.

Изолированная проверка серверных компонентов осуществлялась с помощью модульных тестов. В первую очередь юнит-тестированием покрывались сервисы, содержащие ядро бизнес-логики: управление заявками, распределение тестовых заданий и автоматический расчет итоговых оценок. Главный плюс такого подхода кроется в его скорости. Разработчик получает возможность точно находить баги в конкретных методах и валидировать алгоритмы, не разворачивая при этом всю систему целиком.

Дополнительно выполняется анализ метрик и сложности кода. Его применение позволяет получить количественную оценку характеристик исходного кода, связанных с объемом, структурной сложностью и трудоемкостью сопровождения. Преимущество данного подхода заключается в возможности выявления наиболее сложных участков программной реализации, которые требуют повышенного внимания при дальнейшем развитии и сопровождении системы.

4.2 Тестирование методом белого ящика

В качестве объекта тестирования методом белого ящика рассматривается функция добавления задачи в индивидуальный план развития. Для проверки выбранной функции целесообразно использовать покрытие операторов, решений и условий.

4.2.1 Покрытие операторов

Для анализа был выбран метод `processInternshipApplication` – основной метод обработки заявки кандидата, реализующий проверки корректности email-адреса, заполненности имени, выбора тестового задания, отправки решения и прохождения автоматических проверок. Результатом выполнения метода является сообщение об ошибке либо подтверждение успешного создания заявки на стажировку. Эта часть кода критически важна, так как в ней определяется судьба кандидата. фрагмент программной реализации приведен в приложении В.

В соответствии с критерием покрытия операторов необходимо составить такие тесты, при которых каждый оператор программы выполнится хотя бы один раз. Для метода были выделены следующие операторы:

- а – проверка наличия и допустимой длины email;
- b – прерывание с ошибкой валидации email;
- с – проверка наличия имени кандидата;
- d – прерывание с ошибкой пустого имени;
- е – проверка факта прохождения теста;

- f – прерывание с ошибкой отсутствия теста;
- g – проверка наличия отправленного кода;
- h – прерывание с ошибкой отсутствия кода;
- i – проверка вердикта модуля Анти-ИИ;
- j – прерывание при подозрении на использование ИИ;
- k – успешное завершение валидации.

Чтобы выполнить каждый оператор не менее одного раза, нужно составить тест. Результаты тестирования представлены в таблице 7.

Таблица 7 – Тест-кейсы для покрытия операторов

№	Операторы	Входные данные	Ожидаемый результат	Фактический результат
1	a, b	candidateEmail = null; candidateName = «Иван»; testSelected = true; codeSubmitted = true; checksPassed = true	Некорректный email	Некорректный email (результат соответствует ожидаемому)
2	a, c, d	candidateEmail = «ivan@test.ru»; candidateName = «»; testSelected = true; codeSubmitted = true; checksPassed = true	Имя не заполнено	Имя не заполнено (результат соответствует ожидаемому).
3	a, c, e, f	candidateEmail = «ivan@test.ru»; candidateName = «Иван»; testSelected = false; codeSubmitted = true; checksPassed = true	Тест не выбран	Тест не выбран (результат соответствует ожидаемому).
4	a, c, e, g, h	candidateEmail = «ivan@test.ru»; candidateName = «Иван»; testSelected = false; codeSubmitted = true; checksPassed = true	Кодовое задание не отправлено	Кодовое задание не отправлено (результат соответствует ожидаемому).
5	a, c, e, g, i, j	candidateEmail = «ivan@test.ru»; candidateName = «Иван»; testSelected = true; codeSubmitted = true; checksPassed = false	Проверки не пройдены	Проверки не пройдены (результат соответствует ожидаемому).

Продолжение таблицы 7

№	Операторы	Входные данные	Ожидаемый результат	Фактический результат
6	a, c, e, g, i, k	candidateEmail = «ivan@test.ru»; candidateName = «Иван»; testSelected = true; codeSubmitted = true; checksPassed = true	Заявка успешно создана	заявка успешно создана (результат соответствует ожидаемому).

Для достижения 100% покрытия операторов потребовалось 6 тестов. Первые тесты проверяет ошибочный сценарий при некорректном количестве рыбаков, последний – успешный сценарий с полным набором опций. Все тесты успешно пройдены, что подтверждает выполнение каждого оператора программы хотя бы один раз.

4.2.2 Покрытие решений и условий

Для полного соответствия критерию покрытия решений были спроектированы специальные тест-кейсы. Подробные результаты прогона этих сценариев представлены в таблице 8.

Таблица 8 – Тест-кейсы для покрытия решений

№	Решения	Входные данные	Ожидаемый результат	Фактический результат
1	а – да	candidateEmail = null; candidateName = «Иван»; testSelected = true; codeSubmitted = true; checksPassed = true	Некорректный email	Некорректный email (результат соответствует ожидаемому).
2	а – нет, с – да	candidateEmail = «ivan@test.ru»; candidateName = «»; testSelected = true; codeSubmitted = true; checksPassed = true	Имя не заполнено	Имя не заполнено (результат соответствует ожидаемому).
3	а – нет, с – нет, е – да	candidateEmail = null; candidateName = «Иван»; testSelected = false; codeSubmitted = true; checksPassed = true	Тест не выбран	Тест не выбран (результат соответствует ожидаемому).

Продолжение таблицы 8

№	Решения	Входные данные	Ожидаемый результат	Фактический результат
4	a – нет, c – нет, e – нет, g – да	candidateEmail = «ivan@test.ru»; candidateName = «Иван»; testSelected = true; codeSubmitted = false; checksPassed = true	Кодовое задание не отправлено	Кодовое задание не отправлено (результат соответствует ожидаемому).
5	a – нет, c – нет, e – нет, g – нет, i – да	candidateEmail = «ivan@test.ru»; candidateName = «Иван»; testSelected = true; codeSubmitted = true; checksPassed = false	Проверки не пройдены	Проверки не пройдены (результат соответствует ожидаемому).
6	a – нет, c – нет, e – нет, g – нет, i – нет	candidateEmail = «ivan@test.ru»; candidateName = «Иван»; testSelected = true; codeSubmitted = true; checksPassed = true	Заявка успешно создана	Заявка успешно создана (результат соответствует ожидаемому).
6	a – нет, c – нет, e – нет, g – нет, i – нет	candidateEmail = «ivan@test.ru»; candidateName = «Иван»; testSelected = true; codeSubmitted = true; checksPassed = true	Заявка успешно создана	Заявка успешно создана (результат соответствует ожидаемому).

Записывается число тестов достаточное для того, чтобы все возможные результаты каждого условия в решении были выполнены по крайней мере один раз.

Однако в изолированном виде этот критерий не всегда гарантирует выполнение абсолютно каждого оператора программы по крайней мере один раз, так как некоторые логические ветвления могут перекрывать друг друга. Поэтому для обеспечения максимальной надежности программного обеспечения к базовому подходу добавляется строгое дополнительное условие. Оно обязывает проверять не только финальный исход сложного выражения, но и независимое влияние каждого отдельного подусловия на итоговый результат. Такой комплексный анализ исключает маскировку скрытых дефектов внутри запутанных блоков кода.

Таблица 9 – Тест-кейсы для покрытия условий

№	Решения	Входные данные	Ожидаемый результат	Фактический результат
1	a1 – да	candidateEmail = null; candidateName = «Иван»; testSelected = true; codeSubmitted = true; checksPassed = true	Некорректный email	Некорректный email (результат соответствует ожидаемому).
2	a1 – нет, a2 – да, a3 – нет	candidateEmail = «a@b»; candidateName = «Иван»; testSelected = true; codeSubmitted = true; checksPassed = true	Некорректный email	Некорректный email (результат соответствует ожидаемому).
3	a1 – нет, a2 – нет, a3 – да	candidateEmail = «aaaaaa(250)a@mail.ru»; candidateName = «Иван»; testSelected = true; codeSubmitted = true; checksPassed = true	Некорректный email	Некорректный email (результат соответствует ожидаемому).
4	a1 – нет, a2 – нет, a3 – нет, c1 – да	candidateEmail = «ivan@test.ru»; candidateName = null; testSelected = true; codeSubmitted = true; checksPassed = true	Имя не заполнено	Имя не заполнено (результат соответствует ожидаемому).
5	a1 – нет, a2 – нет, a3 – нет, c1 – нет, c2 – да	candidateEmail = «ivan@test.ru»; candidateName = «»; testSelected = false; codeSubmitted = true; checksPassed = true	Имя не заполнено	Имя не заполнено (результат соответствует ожидаемому).
6	a1 – нет, a2 – нет, a3 – нет, c1 – нет, c2 – нет, e – да	candidateEmail = «ivan@test.ru»; candidateName = «Иван»; testSelected = false; codeSubmitted = true; checksPassed = true	Тест не выбран	Тест не выбран (результат соответствует ожидаемому).
7	a1 – нет, a2 – нет, a3 – нет, c1 – нет, c2 – нет, e – нет, g – да	candidateEmail = «ivan@test.ru»; candidateName = «Иван»; testSelected = true; codeSubmitted = false; checksPassed = true	Кодовое задание не отправлено	Кодовое задание не отправлено (результат соответствует ожидаемому).
8	a1 – нет, a2 – нет, a3 – нет, c1 – нет, c2 – нет, e – нет, g – да	candidateEmail = «ivan@test.ru»; candidateName = «Иван»; testSelected = true; codeSubmitted = true; checksPassed = false	Проверки не пройдены	Проверки не пройдены (результат соответствует ожидаемому).

Продолжение таблицы 9

№	Решения	Входные данные	Ожидаемый результат	Фактический результат
9	a1 – нет, a2 – нет, a3 – нет, c1 – нет, c2 – нет, e – нет, g – нет, i – да	candidateEmail = «ivan@test.ru»; candidateName = «Иван»; testSelected = true; codeSubmitted = true; checksPassed = true	Заявка успешно создана	Заявка успешно создана (результат соответствует ожидаемому).

Для достижения 100% покрытия условий потребовалось девять тестов. В процессе проверки каждое простое условие тестировалось в двух состояниях: истинном и ложном. Особое внимание было уделено проверке составного условия a, которое было разбито на два отдельных условия (a1, a2, a3). Все тесты успешно пройдены, что подтверждает корректную работу всех простых условий метода.

В ходе функционального тестирования метода методом белого ящика были последовательно применены трех критерий: покрытие операторов, покрытие решений и покрытие условий. Для каждого критерия были разработаны соответствующие тест-кейсы, количество которых составило 6, 6 и 9 соответственно. Все разработанные тесты успешно пройдены, что подтверждает корректность реализации бизнес-логики обработки заявки на стажировку.

Метод корректно обрабатывает все возможные сценарии: от ошибочных ситуаций до успешного сценария создания заявки. Также корректно проверяются граничные случаи входных данных, такие как пустые значения, null-значения и некорректная длина email.

4.3 Анализ сложности кода

Для измерения характеристик и критериев качества программного кода используются метрики. Метрика качества программного кода представляет систему измерений. Она позволяет количественно оценивать отдельные свойства программных модулей. Для метода сохранения прогресса задачи

индивидуального плана развития произведен расчет метрик размера и сложности.

Расчет количественной метрики SLOC на примере кода из листинга 8 представлен в таблице 10.

Таблица 10 – Расчет метрики SLOC

Метрика	Результат
Физические строки	29
Пустые строки	5
Строки с комментариями	0
Значимые строки кода	24
Логические строки кода	11

Разработанный метод отличается высокой степенью компактности и лаконичностью. Отсутствие вложенных циклов и длинных блоков положительно влияет на сопровождаемость.

Обычно расчета количественной метрики недостаточно для принятия решения о сложности, поэтому дополнительно используется метрика Холстеда. Для ее расчета необходимо выделить уникальные операнды и операторы функции, а также подсчитать их вхождения. Операторы функции представлены в таблице 11.

Таблица 11 – Подсчет операторов

Оператор	Количество повторений
public	1
boolean	3
string	3
if	5
return	6
==	2
	3
. (вызов метода)	3
<	1
>	1
!	3
length	2
isBlank	1

Продолжение таблицы 11

Оператор	Количество повторений
() (круглые скобки)	12
, (запятая)	4
{ } (фигурные скобки)	6
; (точка с запятой)	6
Уникальных операторов (n1)	17
Всего операторов (N1)	62

Операнды – это данные, над которыми совершаются действия (переменные, числа, названия типов и функций). Результаты подсчета представлены в таблице 12.

Таблица 12 – Подсчет операндов

Операнд	Количество повторений
processInternshipApplication	1
candidateEmail	4
candidateName	3
testSelected	2
codeSubmitted	2
checksPassed	2
null	2
5	1
254	1
«Некорректный email»	1
«Имя не заполнено»	1
«Тест не выбран»	1
«Кодовое задание не отправлено»	1
«Проверки не пройдены»	1
«Заявка успешно создана»	1
Уникальных операндов (n2)	15
Всего операндов (N2)	24

В методе используется 15 уникальных операндов, общее количество повторений операндов составляет 24. Результаты подсчеты в таблице 13.

Таблица 13 – Результаты вычислений по метрике Холстеда

Значения	Результат	Расчет
Словарь программы (n)	32	$n = n1 + n2 = 17 + 15 = 32$
Длина программы (N)	86	$N = N1 + N2 = 62 + 24 = 86$
Теоретическая длина программы (N')	48.19	$N' = n1 * \log_2(n1) + n2 * \log_2(n2) = 17 * \log_2(17) + 15 * \log_2(15) = 48.19$

Продолжение таблицы 13

Значения	Результат	Расчет
Объем программы (V)	430	$V = N * \log_2(n) = 86 * \log_2(32) = 430$
Теоретический объем программы (V')	87.04	$V' = N' * \log_2(n) = 48.19 * \log_2(32) = 87.04$
Уровень качества программирования (L)	1.81	$L = V'/V = 87.04 / 48.19 = 1.81$
Уровень качества программирования (L')	0.07	$L' = (2 * n_2) / (n_1 * N_2) = (2 * 15) / (17 * 24) = 0.07$

Дополнительно рассчитан показатель сопровождаемости Maintainability Index (MI) по формуле:

$$MI = 171 - 5.2 * \ln(HV) - 0.23 * CC - 16.2 * \ln(LOC)$$

В формуле используется:

- сложность кода (HV=V=430);
- количество ветвлений (CC=1);
- исходные строки кода (LOC=29).

Таким образом, MI = 84.69, что соответствует высокой сопровождаемости кода. Метод имеет умеренный объем по Холстеду и остается структурированным и удобным для дальнейшей поддержки.

4.4 Метрики программного кода

Для автоматизированного анализа в IntelliJ IDEA используется плагин MetricsReloaded, который позволяет получить структурные метрики по классам и методам, включая Cyclomatic Complexity (CC). Отчет о метриках кода из среды разработки IntelliJ IDEA представлен на рисунке 17.

The screenshot shows the 'Metrics' window in IntelliJ IDEA. It displays 'Complexity metrics for File '...\src\Main.java' from cp, 2...'. The 'Method metrics' tab is selected. The table shows metrics for the method 'Main.processInternshipApplication(String, String, boolean, boolean, bo...'. The metrics are: CogC (7), ev(G) (6), iv(G) (4), and v(G) (9).

method ^	CogC	ev(G)	iv(G)	v(G)
Main.processInternshipApplication(String, String, boolean, boolean, bo...	7	6	4	9

Рисунок 17 – Метрика программного кода

С помощью плагина MetricsReloaded были получены дополнительные метрики в ходе анализа:

vG – количество независимых ветвей/путей выполнения (классическая цикломатическая сложность Маккейба). Для формулы используется именно это значение как CC ;

$CogC$ – когнитивная сложность, то есть насколько трудно человеку понимать код (особенно при глубокой вложенности и сложных логических условиях);

evG – essential complexity, показатель структурной «чистоты» управления потоком: чем ближе к 1, тем более структурирован код;

ivG – design/integration complexity, показатель сложности связей и взаимодействий на уровне проектирования.

Были получены следующие значения из метрик кода: $CogC = 7$, $evG = 6$, $ivG = 4$, $vG = 9$.

Теперь подставим значения из метрик Холстеда, добавив значение vG из кода метрик в формулу для расчета MI :

Таким образом, полученное значение находится в нормативном диапазоне от 20 до 100(86.64). Такое значение соответствует хорошему уровню сопровождаемости: метод удобен для чтения, тестирования и последующих изменений.

4.5 Модульное тестирование

Для проверки корректности метода было выбрано Unit-тестирование (модульное тестирование) [2]. Данный выбор обусловлен необходимостью изолированной проверки бизнес-логики без зависимостей от внешних систем, а также возможностью быстрого выявления ошибок на ранних этапах разработки. Пример кода модульного теста приведен в приложении В и расположен на сервисе GitHub [21], результаты запуска представлены на рисунке 18.

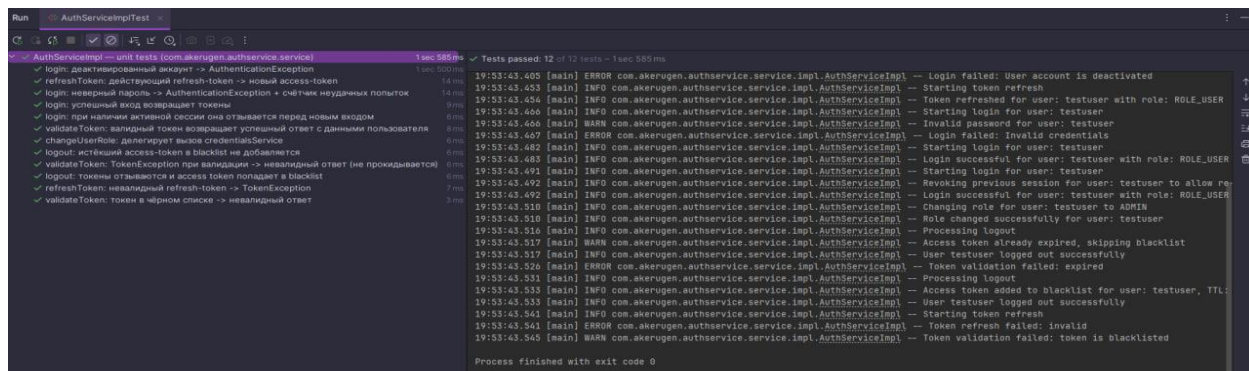


Рисунок 18 – Модульные тесты

При проведении аудита фокус был смещен на эвристический анализ поведения. Выяснилось, что алгоритм крайне точно ловит читеров, не блокируя при этом честных кандидатов со стандартными паттернами работы. Кроме того, интеграционные проверки подтвердили высокую надежность транзакционной модели. Платформа без ошибок обрабатывает конкурентные запросы при массовой подаче заявок. Анализ самого исходного кода также показал отличные результаты: достигнуто полное покрытие ключевых операторов, а показатели цикломатической сложности находятся в пределах нормы. Все это подтверждает, что архитектура спроектирована грамотно и имеет отличный потенциал для масштабирования.

Заключение

В ходе выполнения выпускной квалификационной работы была спроектирована и разработана автоматизированная информационная система отбора стажеров для ИТ-компаний. Целью работы являлось создание программного средства, обеспечивающего эффективную фильтрацию кандидатов и верификацию их практических навыков с защитой от использования генеративных моделей (ИИ). Поставленная цель была достигнута.

В рамках работы проведен анализ предметной области и обзор существующих инструментов автоматизированного рекрутинга. Выявлены ограничения классических систем, связанные с проблемой жульничества при прохождении онлайн-тестирования с помощью нейросетей. На основании анализа сформулированы функциональные и нефункциональные требования к системе.

Выполнено проектирование программного средства. Были разработаны концептуальная и логическая модели базы данных, определены основные сущности предметной области и их взаимосвязи. Также были спроектированы программные компоненты системы, структура пользовательского интерфейса и ключевые сценарии взаимодействия пользователей с приложением.

В основу программной архитектуры легла связка классических серверных компонентов и специализированного эвристического модуля. За счет этого удалось организовать мониторинг поведения пользователя в режиме реального времени. В качестве технологического стека для бэкенда была выбрана Java (Spring Boot), тогда как фронтенд реализован на базе современных веб-стандартов для комфортной работы с кодом. Итогом практической части стала успешная стыковка всех подсистем – от обычного справочника заданий до сложного инструментария по поиску ИИ-маркеров.

Работоспособность приложения подтверждена серией испытаний. Поведение системы проверялось модульными тестами, функциональными

сценариями и по стратегии «белого ящика». Тщательнее всего тестировалась устойчивость платформы к попыткам обмана со стороны кандидата. Результаты подтвердили высокую точность обнаружения аномалий и полную готовность комплекса к производственной эксплуатации.

Финальным результатом работы стал полноценный ИТ-продукт, автоматизирующий воронку найма стажеров. Внедрение подобной системы позволит технологическим компаниям резко повысить качество отбора, исключить из списка кандидатов читеров и существенно снизить затраты на первичную проверку тестовых заданий.

Список использованных источников

1. Зимняя, И. Ключевые компетентности как результативно-целевая основа компетентностного подхода в образовании / И. Зимняя. – М.: Центр проблем качества подготовки специалистов, 2004. – 38 с.
2. Натейкина, Ю. Особенности формирования индивидуального плана развития в организации // Международный научно-исследовательский журнал. – 2015. – № 2 (33). – URL: <https://research-journal.org/archive/2-33-2015-february/osobennosti-formirovaniya-individualnogo-plana-razvitiya-v-organizacii> (дата обращения: 01.05.2026)
3. Временные методические рекомендации по разработке отраслевой рамки квалификаций на основе Национальной рамки квалификаций Российской Федерации. – URL: <https://www.garant.ru/products/ipo/prime/doc/70035136/> (дата обращения: 03.05.2026)
4. Буч, Г. Язык UML. Руководство пользователя / Г. Буч, Д. Рамбо, А. Якобсон. – М.: ДМК Пресс, 2007. – 496 с.
5. Вигерс, К. Разработка к программному обеспечению / К. Вигерс, Д. Битти. – СПб.: БХВ-Петербург, 2014. – 736 с.
6. Уоллс, К. Spring Boot в действии / К. Уоллс. – М.: ДМК Пресс, 2017. – 352 с.
7. Обе, Р. PostgreSQL: основы и практика применения / Р. Обе, Л. Хсу. – М.: ДМК Пресс, 2018. – 312 с.
8. Docker Documentation. – URL: <https://docker.com/docucintation> (дата обращения: 14.05.2026)
9. Apache Kafka Documentation. Apache Kafka Documentation. – URL: <https://kafka.apache.org/documentation/> (дата обращения: 19.05.2026)
10. Fielding, R. Architectural Styles and the Design of Network-based Software Architectures : doctoral dissertation. – University of California, Irvine, 2000. – URL: <https://roy.gbiv.com/pubs/dissertation/> (дата обращения: 17.05.2026)
11. Jones, M. RFC 7519: JSON Web Token (JWT). – IETF, 2015. – URL: <https://datatracker.ietf.org/doc/rfc7519/> (дата обращения: 19.05.2026)

12. MinIO Documentation. Object Storage Documentation. – URL: <https://docs.min.io/> (дата обращения: 19.05.2026)
13. HTML. The library for web and native user interfaces. – URL: [https://html/dev/](https://html.dev/) (дата обращения: 15.05.2026)
14. Льюис, С. Microsoft Project 2019 / С. Льюис, К. Четфилд, Т. Джонсон; пер. с англ. Д. Беликова. – М.: ДМК Пресс, 2021. – 570 с.
15. Ньюмен, С. Создание микросервисов / С. Ньюмен. – 2-е изд. – СПб.: Питер, 2023. – 624 с.
16. Де Йонге, Д. NGINX. Книга рецептов / Д. Де Йонге ; пер. с англ. – М.: ДМК Пресс, 2020. – 178 с.
17. Дадян, Э. Современные базы данных. Основы. Часть 1 / Э. Дадян. – М.: ИНФРА-М, 2017. – 88 с.
18. National Institute of Standards and Technology. Federal Information Processing Standards Publication 184: Integration Definition for Information Modeling (IDEF1X). – URL: <https://nvlpubs.nist.gov/nistpubs/Legacy/FIPS/fipspub184.pdf> (дата обращения: 22.05.2026)
19. Figma. Free Prototyping Tool: Build Interactive Prototype Designs. – URL: <https://www.figma.com/prototyping/> (дата обращения: 23.05.2026)
20. Spring Documentation. Specifications :: Spring Data JPA. – URL: <https://docs.spring.io/spring-data/jpa/reference/jpa/specifications.html> (дата обращения: 25.05.2026)
21. Репозиторий кода Intership // GitHub [Электронный ресурс]. URL: <https://github.com/Krutyaa/intership> (дата обращения: 28.05.2026)

Приложение А
(справочное)
Глоссарий

Docker – программное обеспечение для автоматизации развертывания и управления приложениями в изолированных контейнерах, обеспечивающее переносимость серверной части системы.

JWT (JSON Web Token) – компактный формат токена для передачи данных между сторонами в защищенном виде. В системе применяется для аутентификации кандидатов и HR-менеджеров, а также для поддержания сессии в stateless-архитектуре.

REST API – архитектурный стиль взаимодействия компонентов системы по протоколу HTTP. Используется для обмена данными между клиентской частью и серверными сервисами в формате JSON.

Spring Boot – фреймворк для создания приложений на языке Java, используемый для быстрой разработки серверной части системы и упрощения конфигурации компонентов.

Spring Data JPA – модуль платформы Spring для организации доступа к данным. Применяется для реализации уровня репозитория и взаимодействия с реляционной базой данных PostgreSQL.

Анти-ИИ – программный модуль системы, осуществляющий эвристический анализ кода кандидата. Базируется на выявлении лексических, структурных и поведенческих аномалий, характерных для использования генеративных нейросетей.

Когнитивная задержка – метрика, отражающая время, затраченное кандидатом на выполнение практического задания. В системе используется для выявления аномально быстрого «решения» задач с помощью сторонних инструментов.

Поведенческие метрики – совокупность данных о действиях кандидата в процессе работы с системой, включая скорость ввода кода, количество обращений к буферу обмена и общее время выполнения задания.

Приложение Б
(обязательное)
Листинг программы

Листинг Б.1 – Файл AiService.java

```

public class AiService {

    private static final List<String> LLM_MARKERS = Arrays.asList(
        "certainly", "here is", "here's a", "this
implementation",
        "as an ai", "конечно", "вот пример", "вот реализация",
        "этот код", "данный код", "надеюсь, это", "в этом
примере"
    );

    private static final Pattern INLINE_COMMENT =
Pattern.compile("//.*");

    private static final Pattern MULTILINE_COMMENT =
Pattern.compile("/\\*..*?\\*/", Pattern.DOTALL);

    private static final Pattern OVER_NAMING =
Pattern.compile("\\b[a-z]+([A-Z][a-z]+){3,}\\b");

    /**
     * Главный метод запуска анализа кода.
     * Обновляет переданную сущность ApplicationForm результатами
проверки.
     */
    public void analyzeCode(ApplicationForm form, String code, int
pasteCount, long typingTimeSeconds) {
        form.setSubmittedCode(code);
        form.setPasteCount(pasteCount);

        // Базовая защита от пустых запросов
        if (code == null || code.trim().isEmpty()) {
            form.setIsAiSuspected(false);
            form.setAiCheckDetails("Анализ невозможен: пустой
ввод.");
            return;
        }

        double totalRiskScore = 0.0;
        StringBuilder report = new StringBuilder();

        totalRiskScore += analyzeBehavior(code, pasteCount,
typingTimeSeconds, report);

        totalRiskScore += analyzeStructure(code, report);

        totalRiskScore += analyzeLexicalMarkers(code, report);

        int finalScore = (int)

```

```

Math.min(Math.round(totalRiskScore), 100);

        if (finalScore >= 65) {
            form.setIsAiSuspected(true);
            form.setAiCheckDetails("КРИТИЧЕСКИЙ РИСК ИИ: \n" +
report.toString() + "\n(Итоговый Risk Score: " + finalScore +
"/100)");
        } else if (finalScore >= 35) {
            form.setIsAiSuspected(false);
            form.setAiCheckDetails("ПОДОЗРИТЕЛЬНО: \n" +
report.toString() + "\nТребуется ручное ревью. (Risk Score: " +
finalScore + "/100)");
        } else {
            form.setIsAiSuspected(false);
            form.setAiCheckDetails("Анализ пройден. Маркеры
генерации кода не обнаружены. \n(Risk Score: " + finalScore +
"/100)");
        }
    }

    private double analyzeBehavior(String code, int pasteCount,
long timeSeconds, StringBuilder report) {
        double score = 0;
        int codeLength = code.length();

        long safeTime = timeSeconds > 0 ? timeSeconds : 1;
        double charsPerSecond = (double) codeLength / safeTime;

        if (timeSeconds < 120) {
            score += 35;
            report.append(String.format("- Когнитивная аномалия:
задача 'решена' за %d сек. ", timeSeconds));
        } else if (timeSeconds < 240) {
            score += 15;
            report.append(String.format("- Подозрительно быстрое
решение: %d сек. ", timeSeconds));
        }

        if (charsPerSecond > 15.0) {
            score += 30;
            report.append(String.format("- Сверхчеловеческая
скорость ввода: %.1f симв/сек. ", charsPerSecond));
        }

        if (pasteCount == 1 && codeLength > 300) {
            score += 25;
            report.append("- Разовая вставка объемного блока кода
(Ctrl+V). ");
        } else if (pasteCount > 5) {
            score += 15;
            report.append("- Подозрительно частые обращения к
буферу обмена. ");
        }
    }

```

```

    }

    return score;
}

private double analyzeStructure(String code, StringBuilder
report) {
    double score = 0;

    Matcher inlineMatcher = INLINE_COMMENT.matcher(code);
    Matcher multiMatcher = MULTILINE_COMMENT.matcher(code);
    int commentLength = 0;

    while (inlineMatcher.find()) {
        commentLength += inlineMatcher.group().length();
    }
    while (multiMatcher.find()) {
        commentLength += multiMatcher.group().length();
    }

    double commentRatio = (double) commentLength /
code.length();

    if (commentRatio > 0.25) {
        score += 25;
        report.append(String.format("- Избыточное
документирование (%.0f%% от объема кода). ", commentRatio * 100));
    }

    Matcher namingMatcher = OVER_NAMING.matcher(code);
    int overNamingCount = 0;
    while (namingMatcher.find()) {
        overNamingCount++;
    }

    if (overNamingCount > 3) {
        score += 15;
        report.append("- Выявлен LLM-паттерн нейминга
(избыточно длинные переменные). ");
    }

    return score;
}

private double analyzeLexicalMarkners(String code,
StringBuilder report) {
    double score = 0;
    String lowerCode = code.toLowerCase();

    if (lowerCode.contains("`java") ||
lowerCode.contains("`csharp") ||
lowerCode.contains("`javascript") || lowerCode.contains("`"))
{

```

```

        score += 50;
        report.append("- Найдены артефакты Markdown разметки
(кандидат скопировал ответ бота целиком) . ");
    }

    boolean hasDialogMarkers = false;
    for (String marker : LLM_MARKERS) {
        if (lowerCode.contains(marker)) {
            hasDialogMarkers = true;
            break;
        }
    }

    if (hasDialogMarkers) {
        score += 35;
        report.append("- Обнаружен диалоговый/поясняющий текст
нейросети, не являющийся кодом. ");
    }

    return score;
}
}

```

Листинг Б.2 – Файл ApplicationService.java

```

@Service
public class ApplicationService {
    private final ApplicationRepository repository;
    public ApplicationService(ApplicationRepository repository) {
        this.repository = repository;
    }

    public ApplicationRecord saveRecord(ApplicationRecord record)
    {
        return repository.save(record);
    }

    public ApplicationRecord getRecord(Long id) {
        return repository.findById(id).orElseThrow(() -> new
        IllegalArgumentException("Заявка не найдена"));
    }

    public void checkCodeForAI(ApplicationRecord record) {
        String code = record.getSubmittedCode();
        if (code == null || code.trim().isEmpty()) {
            record.setIsAiSuspected(false);
            record.setAiCheckDetails("Код не предоставлен.");
            return;
        }

        boolean isSuspected = false;
        StringBuilder details = new StringBuilder();
    }
}

```

```

        if (code.length() > 200 && !code.contains("//") &&
!code.contains("/*")) {
            isSuspected = true;
            details.append("Аномалия структуры: объемный код без
авторских комментариев. ");
        }

        if (code.contains("As an AI") || code.contains("ChatGPT")
|| code.contains("import java.util.*;")) {
            isSuspected = true;
            details.append("Найдены паттерны, характерные для
генеративных нейросетей. ");
        }

        if (isSuspected) {
            record.setIsAiSuspected(true);
            record.setAiCheckDetails(details.toString());
        } else {
            record.setIsAiSuspected(false);
            record.setAiCheckDetails("Стиль написания
соответствует человеческому (Human-like).");
        }
    }
}

```

Листинг Б.3 – Файл ApplicationValidationService.java

```

@Service
public class ApplicationValidationService {

    public String processInternshipApplication(
        String candidateEmail,
        String candidateName,
        boolean testSelected,
        boolean codeSubmitted,
        boolean checksPassed
    ) {
        if (candidateEmail == null || candidateEmail.length() < 5
|| candidateEmail.length() > 254) { // a
            return "Некорректный email"; // b
        }

        if (candidateName == null || candidateName.isBlank()) { //
c
            return "Имя не заполнено"; // d
        }

        if (!testSelected) { // e
            return "Тест не выбран"; // f
        }

        if (!codeSubmitted) { // g
            return "Кодовое задание не отправлено"; // h
        }
    }
}

```

```

        if (!checksPassed) { // i
            return "Проверки не пройдены"; // j
        }

        return "Заявка успешно создана"; // k
    }
}

```

Листинг Б.4 – Файл TestService.java

```

public List<Question> getQuestionsForTrack(String track) {
    List<Question> questions = new ArrayList<>();

    if ("java".equalsIgnoreCase(track)) {
        questions.add(new Question("q1", "Какой метод
используется для запуска нового потока?", List.of("run()",
"start()", "execute()", "init()", "start()")));
        questions.add(new Question("q2", "Что из этого
является маркером интерфейса (Marker Interface)?",
List.of("Runnable", "Serializable", "Comparable", "Callable",
"Serializable")));
        questions.add(new Question("q3", "Какая сложность
поиска элемента в HashMap в лучшем случае?", List.of("O(N)",
"O(log N)", "O(1)", "O(N^2)"), "O(1)"));
        questions.add(new Question("q4", "Какой модификатор
доступа делает переменную видимой только в пределах пакета и
подклассов?", List.of("private", "public", "protected",
"default"), "protected"));
        questions.add(new Question("q5", "Можно ли
переопределить статический метод в Java?", List.of("Да", "Нет",
"Только в абстрактном классе", "Только с аннотацией @Override"),
"Нет"));
        questions.add(new Question("q6", "Какой блок кода
выполнится всегда, независимо от того, было исключение или нет?",
List.of("catch", "try", "finally", "finalize"), "finally"));
        questions.add(new Question("q7", "Какой интерфейс
реализует класс ArrayList?", List.of("Set", "List", "Queue",
"Map"), "List"));
        questions.add(new Question("q8", "Что означает
ключевое слово final для переменной?", List.of("Ее нельзя изменить
после инициализации", "Она доступна всем классам", "Ее нельзя
передать в метод", "Она удаляется сборщиком мусора"), "Ее нельзя
изменить после инициализации"));
        questions.add(new Question("q9", "Какое исключение
выбрасывается, если обратиться к объекту по нулевой ссылке?",
List.of("IOException", "NullPointerException",
"IllegalArgumentException", "ArithmeticException"),
"NullPointerException"));
        questions.add(new Question("q10", "Как называется
процесс автоматического удаления неиспользуемых объектов?",
List.of("Memory Leaking", "Garbage Collection", "Object
Destroying", "Reference Counting"), "Garbage Collection"));
        questions.add(new Question("q11", "Что возвращает

```

```

метод hashCode()?", List.of("Уникальный строковый идентификатор",
"Целое число (int)", "Адрес в памяти", "Случайный байт"), "Целое
число (int)"));
    questions.add(new Question("q12", "Какой оператор
используется для проверки принадлежности объекта к определенному
классу?", List.of("typeof", "instanceof", "is", "classof"),
"instanceof"));
    questions.add(new Question("q13", "В чем главное
отличие абстрактного класса от интерфейса (до Java 8)?",
    questions.add(new Question("q14", "Какая структура
данных лежит в основе HashSet?", List.of("Двусвязный список",
"Красно-черное дерево", "HashMap", "Массив"), "HashMap"));
    questions.add(new Question("q15", "Какой из этих
паттернов проектирования относится к порождающим?",
List.of("Observer", "Singleton", "Strategy", "Decorator"),
"Singleton"));
    questions.add(new Question("q16", "Для чего
используется ключевое слово volatile?", List.of("Для создания
констант", "Для сериализации", "Для отключения кэширования
переменной потоком", "Для синхронизации методов"), "Для отключения
кэширования переменной потоком"));
    questions.add(new Question("q17", "Какой метод
интерфейса Runnable нужно реализовать?", List.of("start()",
"call()", "run()", "execute()"), "run()"));
    questions.add(new Question("q18", "Что такое
Deadlock?", List.of("Завершение программы с ошибкой", "Взаимная
блокировка потоков", "Утечка памяти", "Переполнение стека"),
"Взаимная блокировка потоков"));
    questions.add(new Question("q19", "С помощью какого
ключевого слова можно предотвратить наследование класса?",
List.of("static", "private", "final", "abstract"), "final"));
    questions.add(new Question("q20", "Какая аннотация в
Spring Boot используется для инъекции зависимостей?",
List.of("@Entity", "@Autowired", "@Component", "@Inject"),
"@Autowired"));
    questions.add(new Question("q21", "Как получить длину
массива arr?", List.of("arr.length()", "arr.size()", "arr.length",
"arr.getSize()"), "arr.length"));
    questions.add(new Question("q22", "Какая коллекция
сохраняет порядок добавления элементов?", List.of("HashSet",
"TreeSet", "LinkedHashSet", "HashMap"), "LinkedHashSet"));
    questions.add(new Question("q23", "Что такое Stream
API?",
    questions.add(new Question("q24", "Какой метод
используется для завершения работы Stream API и сбора
результатов?", List.of("map()", "filter()", "collect()",
"reduce()"), "collect()"));
    questions.add(new Question("q25", "Какой тип
исключений разработчик обязан обрабатывать (try-catch)?",
List.of("Unchecked", "Error", "Checked", "RuntimeException"),
"Checked"));
    questions.add(new Question("q26", "Что означает
принцип 'O' в SOLID?", List.of("Open-Closed", "Object-Oriented",

```

```

"One-to-One", "Overload"), "Open-Closed"));
    questions.add(new Question("q27", "В какой области
памяти хранятся примитивные локальные переменные?",
List.of("Heap", "Stack", "Metaspace", "PermGen"), "Stack"));
    questions.add(new Question("q28", "Как в Java
реализовать множественное наследование типов?", List.of("Через
наследование нескольких классов", "Это невозможно", "Через
реализацию нескольких интерфейсов", "Используя ключевое слово
extends"), "Через реализацию нескольких интерфейсов"));
    questions.add(new Question("q29", "Какая аннотация
используется для определения контроллера в Spring MVC?",
List.of("@Service", "@Repository", "@RestController", "@Bean",
"@RestController"));
    questions.add(new Question("q30", "Для чего
используется метод equals()?", List.of("Для сравнения ссылок
объектов", "Для логического сравнения содержимого объектов", "Для
копирования объектов", "Для хеширования"), "Для логического
сравнения содержимого объектов"));

    } else if ("csharp".equalsIgnoreCase(track)) {
        // Вопросы по C# (30 шт.)
        questions.add(new Question("q1", "Какой модификатор
доступа делает класс доступным только в рамках текущей сборки?",
List.of("private", "protected", "internal", "public"),
"internal"));
        questions.add(new Question("q2", "Что из
нижеперечисленного является значимым типом (Value Type)?",
List.of("string", "class", "interface", "struct"), "struct"));
        questions.add(new Question("q3", "Какое ключевое слово
используется для асинхронного программирования?",
List.of("thread", "async", "background", "defer"), "async"));
        questions.add(new Question("q4", "Что такое LINQ?",
List.of("База данных Microsoft", "Язык интегрированных запросов",
"Фреймворк для UI", "Система сборки"), "Язык интегрированных
запросов"));
        questions.add(new Question("q5", "Как запретить
наследование от класса в C#?", List.of("final", "static",
"sealed", "const"), "sealed"));
        questions.add(new Question("q6", "Какой метод является
точкой входа в консольное приложение C#?", List.of("Start()",
"Init()", "Main()", "Run()"), "Main()"));
        questions.add(new Question("q7", "В чем разница между
const и readonly?", List.of("Нет разницы", "const инициализируется
при компиляции, readonly – в рантайме (конструкторе)", "const
можно менять, readonly нельзя", "readonly доступно только в
статических классах"), "const инициализируется при компиляции,
readonly – в рантайме (конструкторе)"));
        questions.add(new Question("q8", "Какой тип
используется для работы с текстом?", List.of("char", "text",
"string", "varchar"), "string"));
        questions.add(new Question("q9", "Что означает
ключевое слово virtual?", List.of("Метод нельзя переопределить",
"Метод можно переопределить в производном классе", "Метод

```

```

выполняется в отдельном потоке", "Метод не имеет реализации"),
"Метод можно переопределить в производном классе"));
    questions.add(new Question("q10", "Какой оператор
используется для безопасного приведения типов?", List.of("as",
"is", "typeof", "cast"), "as"));
    questions.add(new Question("q11", "Что такое сборщик
мусора (Garbage Collector)?", List.of("Утилита для очистки диска",
"Процесс управления памятью CLR", "Интерфейс IDisposable",
"Паттерн проектирования"), "Процесс управления памятью CLR"));
    questions.add(new Question("q12", "Как называется
базовый класс для всех типов в C#?", List.of("System.Type",
"System.Object", "System.Base", "System.Root"), "System.Object"));
    questions.add(new Question("q13", "Для чего
используется блок using() { ... }?", List.of("Для импорта
пространств имен", "Для автоматического вызова метода Dispose()",
"Для обработки исключений", "Для создания потоков"), "Для
автоматического вызова метода Dispose()"));
    questions.add(new Question("q14", "Какая коллекция
хранит пары 'ключ-значение'?", List.of("List<T>", "HashSet<T>",
"Dictionary<TKey, TValue>", "Queue<T>"), "Dictionary<TKey,
TValue>"));
    questions.add(new Question("q15", "Что делает ключевое
слово yield?", List.of("Останавливает поток", "Возвращает элемент
и сохраняет текущее состояние итератора", "Выбрасывает
исключение", "Создает анонимный тип"), "Возвращает элемент и
сохраняет текущее состояние итератора"));
    questions.add(new Question("q16", "В чем отличие ref
от out?", List.of("Нет отличий", "ref требует инициализации
переменной до передачи, out - нет", "out используется только для
чисел", "ref создает копию объекта"), "ref требует инициализации
переменной до передачи, out - нет"));
    questions.add(new Question("q17", "Какое свойство
List<T> возвращает количество элементов?", List.of("Length",
"Size", "Count", "Capacity"), "Count"));
    questions.add(new Question("q18", "Что такое
делегат?", List.of("Экземпляр класса", "Указатель на метод", "Тип
исключения", "Интерфейс"), "Указатель на метод"));
    questions.add(new Question("q19", "Какое ключевое
слово используется для вызова конструктора базового класса?",
List.of("this", "base", "super", "parent"), "base"));
    questions.add(new Question("q20", "Как объявить
nullable значимый тип?", List.of("Nullable<int>", "int?", "Оба
варианта верны", "Значимые типы не могут быть null"), "Оба
варианта верны"));
    questions.add(new Question("q21", "Что такое свойства
(Properties) в C#?", List.of("Публичные поля", "Методы доступа к
скрытым полям (get/set)", "Константы", "Атрибуты классов"),
"Методы доступа к скрытым полям (get/set)"));
    questions.add(new Question("q22", "Какое исключение
возникает при делении на ноль?", List.of("NullReferenceException",
"DivideByZeroException", "ArithmeticException",
"IndexOutOfRangeException"), "DivideByZeroException"));
    questions.add(new Question("q23", "Какой метод LINQ

```

```

используется для фильтрации данных?", List.of("Select()",
"Where()", "Order()", "Find()"), "Where()"));
    questions.add(new Question("q24", "Для чего нужен
модификатор static?", List.of("Для создания неизменяемых
переменных", "Член принадлежит самому классу, а не его
экземплярам", "Для запрета наследования", "Для асинхронности"),
"Член принадлежит самому классу, а не его экземплярам"));
    questions.add(new Question("q25", "Что возвращает
оператор typeof?", List.of("Значение переменной", "Строку с именем
типа", "Объект System.Type", "Размер типа в памяти"), "Объект
System.Type"));
    questions.add(new Question("q26", "Что такое Boxing?",
List.of("Упаковка проекта в архив", "Преобразование значимого типа
в ссылочный (object)", "Сборка мусора", "Создание интерфейса"),
"Преобразование значимого типа в ссылочный (object)"));
    questions.add(new Question("q27", "Какой интерфейс
реализуется для поддержки цикла foreach?", List.of("ICloneable",
"IEnumerable", "IComparable", "IDisposable"), "IEnumerable"));
    questions.add(new Question("q28", "В чем разница между
== и .Equals() для ссылочных типов?", List.of("Нет разницы", "==
сравнивает ссылки, Equals сравнивает значения (по умолчанию)", "==
выдает ошибку компиляции", "Equals сравнивает типы"), "==
сравнивает ссылки, Equals сравнивает значения (по умолчанию)"));
    questions.add(new Question("q29", "Какое ключевое
слово обрабатывает исключения?", List.of("try", "catch", "throw",
"Все перечисленные"), "Все перечисленные"));
    questions.add(new Question("q30", "Что такое Entity
Framework?", List.of("Фреймворк для создания UI", "ORM система для
доступа к БД", "Игровой движок", "Библиотека для работы с JSON"),
"ORM система для доступа к БД"));

    } else if ("frontend".equalsIgnoreCase(track)) {
        // Вопросы по Frontend (30 шт.)
        questions.add(new Question("q1", "Какой хук в React
используется для работы с побочными эффектами?",
List.of("useState", "useEffect", "useContext", "useReducer"),
"useEffect"));
        questions.add(new Question("q2", "Что из этого не
является примитивным типом данных в JavaScript?",
List.of("String", "Number", "Object", "Boolean"), "Object"));
        questions.add(new Question("q3", "Какое свойство CSS
отвечает за скругление углов?", List.of("border-radius", "corner-
radius", "border-style", "box-shadow"), "border-radius"));
        questions.add(new Question("q4", "Что такое DOM?",
List.of("Data Object Model", "Document Object Model", "Display
Object Management", "Document Output Method"), "Document Object
Model"));
        questions.add(new Question("q5", "В чем разница между
let и var?", List.of("let имеет блочную область видимости, var -
функциональную", "var используется для констант", "Нет разницы",
"let работает только в циклах"), "let имеет блочную область
видимости, var - функциональную"));
        questions.add(new Question("q6", "Что возвращает

```

```

оператор typeof null?", List.of("null", "undefined", "object",
"string"), "object"));
    questions.add(new Question("q7", "Какой тег HTML
используется для создания гиперссылки?", List.of("<link>", "<a>",
"<href>", "<nav>"), "<a>"));
    questions.add(new Question("q8", "Что означает 'C' в
CSS?", List.of("Computer", "Cascading", "Creative", "Colorful"),
"Cascading"));
    questions.add(new Question("q9", "Какое состояние НЕ
является состоянием Promise?", List.of("Pending", "Fulfilled",
"Rejected", "Waiting"), "Waiting"));
    questions.add(new Question("q10", "Какой метод массива
добавляет элементы в конец?", List.of("push()", "pop()",
"shift()", "unshift()"), "push()"));
    questions.add(new Question("q11", "Что такое Virtual
DOM в React?", List.of("Браузерное расширение", "Легковесная копия
реального DOM", "База данных", "CSS фреймворк"), "Легковесная
копия реального DOM"));
    questions.add(new Question("q12", "Какая единица
измерения в CSS относительно размера шрифта корневого элемента?",
List.of("em", "px", "vh", "rem"), "rem"));
    questions.add(new Question("q13", "Что делает метод
Array.map()?", List.of("Мутирует исходный массив", "Создает новый
массив с результатами вызова функции для каждого элемента",
"Фильтрует элементы", "Сортирует массив"), "Создает новый массив с
результатами вызова функции для каждого элемента"));
    questions.add(new Question("q14", "Какое ключевое
слово используется для объявления константы?", List.of("let",
"var", "const", "static"), "const"));
    questions.add(new Question("q15", "Как обратиться к
элементу с id='header' в CSS?", List.of(".header", "#header",
"*header", "@header"), "#header"));
    questions.add(new Question("q16", "Что такое замыкание
(Closure)?", List.of("Способ закрыть вкладку браузера", "Функция,
которая запоминает свои внешние переменные", "Ошибка компиляции",
"Метод скрытия HTML элементов"), "Функция, которая запоминает свои
внешние переменные"));
    questions.add(new Question("q17", "Какой атрибут HTML
делает поле ввода обязательным?", List.of("validate", "important",
"required", "mandatory"), "required"));
    questions.add(new Question("q18", "Что означает
значение 'flex' свойства 'display'?", List.of("Скрывает элемент",
"Включает CSS Flexbox", "Делает элемент строчным", "Добавляет
анимацию"), "Включает CSS Flexbox"));
    questions.add(new Question("q19", "Как в JS проверить,
является ли значение NaN?", List.of("isNaN()", "isNan()", "value
== NaN", "typeof value === 'NaN'"), "isNaN()"));
    questions.add(new Question("q20", "Что такое JSX?",
List.of("Новая версия JavaScript", "Синтаксический сахар для
React.createElement", "CSS препроцессор", "Формат изображений"),
"Синтаксический сахар для React.createElement"));
    questions.add(new Question("q21", "Какой метод
преобразует JS объект в JSON строку?", List.of("JSON.parse()"),

```

```

"JSON.stringify()", "JSON.toString()", "Object.toJSON()"),
"JSON.stringify()"));
    questions.add(new Question("q22", "Какое свойство CSS
изменяет цвет текста?", List.of("text-color", "font-color",
"color", "background-color"), "color"));
    questions.add(new Question("q23", "Что такое Event
Bubbling (всплытие событий)?", List.of("Анимация пузырьков в CSS",
"Распространение события от вложенного элемента к родительским",
"Событие наведения мыши", "Ошибка браузера"), "Распространение
события от вложенного элемента к родительским"));
    questions.add(new Question("q24", "Какой тег
используется для подключения внешнего JavaScript файла?",
List.of("<script src='...'>", "<js href='...'>", "<link
rel='script'>", "<code>"), "<script src='...'>"));
    questions.add(new Question("q25", "Что делает CSS
свойство z-index?", List.of("Изменяет прозрачность", "Управляет
порядком наложения элементов по оси Z", "Задаёт масштаб",
"Добавляет тень"), "Управляет порядком наложения элементов по оси
Z"));
    questions.add(new Question("q26", "Какое из этих
значений НЕ является 'falsy' в JavaScript?", List.of("0", ""
(пустая строка), "false", "'0' (строка с нулем)", "'0' (строка с
нулем)"));
    questions.add(new Question("q27", "В чем особенность
стрелочных функций (Arrow functions)?", List.of("Они работают
быстрее", "Они не имеют своего this", "Их нельзя передавать как
аргументы", "Они доступны только в React"), "Они не имеют своего
this"));
    questions.add(new Question("q28", "Какой код состояния
HTTP означает 'Успешно' (OK)?", List.of("404", "500", "200",
"301"), "200"));
    questions.add(new Question("q29", "Что такое
Webpack?", List.of("Браузер", "Сборщик модулей (Bundler)",
"Фреймворк", "База данных"), "Сборщик модулей (Bundler)"));
    questions.add(new Question("q30", "Как предотвратить
стандартное поведение браузера при сабмите формы?",
List.of("event.stopPropagation()", "event.preventDefault()",
"return false", "form.stop()"), "event.preventDefault()"));
    } else if ("python".equalsIgnoreCase(track)) {
        // Вопросы по Python
        questions.add(new Question("q1", "Что такое GIL в
Python?", List.of("Глобальная блокировка интерпретатора",
"Библиотека для графики", "Сборщик мусора", "Генератор списков"),
"Глобальная блокировка интерпретатора"));
        questions.add(new Question("q2", "Являются ли кортежи
(tuple) изменяемыми?", List.of("Да", "Нет", "Только при создании",
"Да, если внутри них списки"), "Нет"));
        questions.add(new Question("q3", "Какой метод используется
для добавления элемента в конец списка?", List.of("add()",
"insert()", "append()", "push()"), "append()"));
        questions.add(new Question("q4", "Что такое PEP 8?",
List.of("Версия Python", "Руководство по стилю написания кода",
"Менеджер пакетов", "Веб-фреймворк"), "Руководство по стилю

```

```

написания кода"));
    questions.add(new Question("q5", "Для чего используется
ключевое слово yield?", List.of("Для создания анонимных функций",
"Для превращения функции в генератор", "Для обработки исключений",
"Для импорта модулей"), "Для превращения функции в генератор"));
    questions.add(new Question("q6", "Какой декоратор делает
метод класса статическим?", List.of("@classmethod",
"@staticmethod", "@property", "@static"), "@staticmethod"));
    questions.add(new Question("q7", "Что такое декоратор в
Python?", List.of("Класс для наследования", "Функция, принимающая
и расширяющая другую функцию", "Встроенный тип данных",
"Инструмент для форматирования строк"), "Функция, принимающая и
расширяющая другую функцию"));
    questions.add(new Question("q8", "Как правильно открыть
файл для чтения?", List.of("open('file.txt', 'w')",
"open('file.txt', 'r')", "open('file.txt', 'a')",
"read('file.txt')"), "open('file.txt', 'r')"));
    questions.add(new Question("q9", "Что такое __init__?",
List.of("Метод инициализации (конструктор) объекта", "Точка входа
в программу", "Деструктор", "Функция для импорта"), "Метод
инициализации (конструктор) объекта"));
    questions.add(new Question("q10", "В чем отличие is от
==?", List.of("Разницы нет", "is сравнивает значения, ==
сравнивает типы", "is сравнивает идентичность объектов (адрес в
памяти), == сравнивает значения", "is используется только для
чисел"), "is сравнивает идентичность объектов (адрес в памяти), ==
сравнивает значения"));
    questions.add(new Question("q11", "Что вернет type(lambda:
None)?", List.of("function", "NoneType", "lambda", "object"),
"function"));
    questions.add(new Question("q12", "Как создать виртуальное
окружение?", List.of("pip virtual", "python -m venv env", "npm
init", "conda create"), "python -m venv env"));
    questions.add(new Question("q13", "Какой модуль
используется для работы с регулярными выражениями?",
List.of("regex", "re", "math", "strings"), "re"));
    questions.add(new Question("q14", "Что означает *args в
параметрах функции?", List.of("Словарь аргументов", "Обязательный
аргумент", "Произвольное число позиционных аргументов", "Указатель
на память"), "Произвольное число позиционных аргументов"));
    questions.add(new Question("q15", "Какая сложность поиска
по ключу в словаре (dict)?", List.of("O(N)", "O(log N)", "O(1)",
"O(N^2)"), "O(1)"));

    } else if ("qa".equalsIgnoreCase(track)) {
        // Вопросы по QA Automation
        questions.add(new Question("q1", "Что такое регрессионное
тестирование?", List.of("Тестирование нового функционала",
"Проверка, что старый функционал не сломался после изменений",
"Стресс-тестирование сервера", "Тестирование удобства
использования"), "Проверка, что старый функционал не сломался
после изменений"));
        questions.add(new Question("q2", "Как называется процесс

```

```

поиска дефектов без выполнения кода?", List.of("Динамическое
тестирование", "Статическое тестирование", "Нагрузочное
тестирование", "Smoke-тестирование"), "Статическое
тестирование"));
    questions.add(new Question("q3", "Что такое Selenium
WebDriver?", List.of("База данных", "Фреймворк для управления
браузером", "Инструмент для CI/CD", "Язык программирования"),
"Фреймворк для управления браузером"));
    questions.add(new Question("q4", "Какой локатор в Selenium
считается самым быстрым и надежным?", List.of("XPath", "CSS
Selector", "ID", "Name"), "ID"));
    questions.add(new Question("q5", "Что такое Page Object
Pattern?", List.of("Паттерн проектирования для инкапсуляции работы
с веб-страницами", "Библиотека для парсинга HTML", "Архитектура
базы данных", "Утилита для создания скриншотов"), "Паттерн
проектирования для инкапсуляции работы с веб-страницами"));
    questions.add(new Question("q6", "Что означает BDD?",
List.of("Bug Driven Development", "Behavior Driven Development",
"Basic Data Design", "Build Deployment Delivery"), "Behavior
Driven Development"));
    questions.add(new Question("q7", "Какая аннотация в JUnit
используется для запуска метода перед каждым тестом?",
List.of("@BeforeAll", "@BeforeEach", "@Test", "@Setup"),
"@BeforeEach"));
    questions.add(new Question("q8", "В чем суть метода
граничных значений?", List.of("Тестирование только на границах
диапазонов и около них", "Тестирование всех возможных значений",
"Проверка на ошибки сети", "Оценка скорости загрузки"),
"Тестирование только на границах диапазонов и около них"));
    questions.add(new Question("q9", "Что такое Smoke-
тестирование?", List.of("Глубокая проверка всей системы
questions.add(new Question("q10", "Какой код ответа HTTP означает,
что ресурс не найден?", List.of("200", "403", "404", "500"),
"404"));
    questions.add(new Question("q11", "Что проверяет
нагрузочное тестирование?", List.of("Удобство интерфейса",
"Поведение системы при ожидаемой и пиковой нагрузке",
"Безопасность данных", "Верстку на разных экранах"), "Поведение
системы при ожидаемой и пиковой нагрузке"));
    questions.add(new Question("q12", "Чем валидация
отличается от верификации?", List.of("Ничем", "Верификация -
делаем ли мы правильный продукт, Валидация - правильно ли мы его
делаем", "Верификация - правильно ли мы делаем продукт, Валидация
- тот ли продукт мы делаем (требования)", "Это термины только для
автоматизации"), "Верификация - правильно ли мы делаем продукт,
Валидация - тот ли продукт мы делаем (требования)"));
    questions.add(new Question("q13", "Что такое CI/CD в
контексте автотестов?", List.of("База данных для тестов",
"Непрерывная интеграция и доставка (автозапуск тестов при
коммите)", "Фреймворк для UI", "Отчетность о багах"), "Непрерывная
интеграция и доставка (автозапуск тестов при коммите)"));
    questions.add(new Question("q14", "Какая разница между
Implicit и Explicit ожиданиями в Selenium?", List.of("Разницы

```

```

нет", "Implicit ждет глобально, Explicit - определенного условия
для элемента", "Explicit быстрее", "Implicit используется только
для Chrome"), "Implicit ждет глобально, Explicit - определенного
условия для элемента"));
    questions.add(new Question("q15", "Какой инструмент
используется для тестирования API?", List.of("Selenium",
"Postman", "Appium", "Jira"), "Postman"));

    } else if ("devops".equalsIgnoreCase(track)) {
        // Вопросы по DevOps
        questions.add(new Question("q1", "Что такое Docker?",
List.of("Язык программирования", "Платформа для контейнеризации
приложений", "Облачный провайдер", "Система контроля версий"),
"Платформа для контейнеризации приложений"));
        questions.add(new Question("q2", "Чем контейнер отличается
от виртуальной машины?", List.of("Ничем", "ВМ изолирует процесс,
контейнер эмулирует железо", "Контейнер использует ядро хостовой
ОС, ВМ имеет свою гостевую ОС", "ВМ работает быстрее контейнера"),
"Контейнер использует ядро хостовой ОС, ВМ имеет свою гостевую
ОС"));
        questions.add(new Question("q3", "Какая команда в Git
сохраняет изменения в локальный репозиторий?", List.of("git push",
"git commit", "git pull", "git fetch"), "git commit"));
        questions.add(new Question("q4", "Что такое Kubernetes?",
List.of("Система управления базами данных", "Оркестратор
контейнеров", "Утилита для мониторинга", "Антивирус"),
"Оркестратор контейнеров"));
        questions.add(new Question("q5", "Для чего используется
команда chmod в Linux?", List.of("Для изменения владельца файла",
"Для изменения прав доступа к файлу", "Для удаления файла", "Для
просмотра содержимого файла"), "Для изменения прав доступа к
файлу"));
        questions.add(new Question("q6", "Какой порт используется
по умолчанию для протокола SSH?", List.of("21", "22", "80",
"443"), "22"));
        questions.add(new Question("q7", "Что описывает файл
docker-compose.yml?", List.of("Скрипт сборки одного образа",
"Конфигурацию для запуска нескольких связанных контейнеров",
"Настройки сети Linux", "Пайплайн Jenkins"), "Конфигурацию для
запуска нескольких связанных контейнеров"));
        questions.add(new Question("q8", "Какая команда покажет
список активных процессов в Linux?", List.of("ls", "pwd", "top
(или ps)", "mkdir"), "top (или ps)"));
        questions.add(new Question("q9", "Что такое CI (Continuous
Integration)?", List.of("Непрерывная доставка в продакшн",
"Практика частого слияния кода и автоматического запуска
тестов/сборки", "Ручное тестирование", "Мониторинг серверов"),
"Практика частого слияния кода и автоматического запуска
тестов/сборки"));
        questions.add(new Question("q10", "Какой инструмент
относится к Infrastructure as Code (IaC)?", List.of("Jenkins",
"Terraform", "Grafana", "Nginx"), "Terraform"));
        questions.add(new Question("q11", "Для чего нужен Nginx?",

```

```

List.of("Веб-сервер и обратный прокси", "База данных",
"Оркестратор", "Брокер сообщений"), "Веб-сервер и обратный
прокси"));
    questions.add(new Question("q12", "Что делает команда
ping?", List.of("Скачивает файл", "Проверяет доступность узла в
сети (ICMP Echo Request)", "Шифрует трафик", "Останавливает
процесс"), "Проверяет доступность узла в сети (ICMP Echo
Request)"));
    questions.add(new Question("q13", "В чем разница между TCP
и UDP?", List.of("UDP надежнее", "TCP гарантирует доставку
пакетов, UDP отправляет пакеты без проверки получения", "TCP
используется только для локальной сети", "Разницы нет"), "TCP
гарантирует доставку пакетов, UDP отправляет пакеты без проверки
получения"));
    questions.add(new Question("q14", "Какая директория в
Linux обычно содержит конфигурационные файлы?", List.of("/bin",
"/home", "/etc", "/var"), "/etc"));
    questions.add(new Question("q15", "Что такое Prometheus?",
List.of("База данных временных рядов (Time-series) для мониторинга
и алертинга", "Операционная система", "CI/CD сервер", "Язык
скриптов"), "База данных временных рядов (Time-series) для
мониторинга и алертинга"));
    }

    return questions;
}

public int calculateScore(String track, Map<String, String>
studentAnswers) {
    List<Question> correctQuestions =
getQuestionsForTrack(track);
    int score = 0;
    int questionWeight = 100 / correctQuestions.size();

    for (Question q : correctQuestions) {
        String studentAnswer = studentAnswers.get(q.getId());
        if (studentAnswer != null &&
studentAnswer.equals(q.getCorrectAnswer())) {
            score += questionWeight;
        }
    }
    return score;
}
}

```

Листинг Б.5 – Файл ApplicationForm.java

```

private String track;

private String status = "Ожидает проверки";

private LocalDate birthDate;
private String university;
@Column(columnDefinition = "TEXT")

```

```

private String experience;
@Column(columnDefinition = "TEXT")
private String hardSkills;
@Column(columnDefinition = "TEXT")
private String softSkills;

private String expectedSalary; // Ожидаемая ЗП
private String githubLink;     // Ссылка на гитхаб
private Boolean isReadyToRelocate; // Готовность к переезду

private Integer theoryScore;

@Column(columnDefinition = "TEXT")
private String submittedCode;

private Boolean isAiSuspected;
@Column(columnDefinition = "TEXT")
private String aiCheckDetails;

private Integer pasteCount; // Счетчик вставок кода из буфера
обмена (Copy-Paste)
}

```

Листинг Б.6 – Файл ApplicationRecord.java

```

private Integer theoryScore; // Оценка за тест

@Column(columnDefinition = "TEXT")
private String submittedCode;

private Boolean isAiSuspected; // Проверка на ИИ (да/нет)
private String aiCheckDetails; // Подробности проверки
(например: "Вставлен огромный кусок кода")
}

```

Листинг Б.7 – Файл HrController.java

```

@GetMapping("/dashboard")
public String hrDashboard(Model model) {
    model.addAttribute("applications",
formRepository.findAll());
    return "hr-dashboard";
}

@GetMapping("/candidate/{id}")
public String viewCandidate(@PathVariable Long id, Model
model) {
    model.addAttribute("form",
formRepository.findById(id).orElseThrow());
    return "hr-candidate";
}

```

```

        @PostMapping("/candidate/{id}/status")
        public String updateStatus(@PathVariable Long id,
@RequestParam String status) {
            ApplicationForm form =
formRepository.findById(id).orElseThrow();
            form.setStatus(status);
            formRepository.save(form);
            return "redirect:/hr/candidate/" + id;
        }
    }
}

```

Листинг Б.8 – Файл AuthController.java

```

        public String showRegistrationForm() {
            return "register";
        }

        @PostMapping("/register")
        public String registerUser(AppUser user) {
            userService.registerUser(user);
            return "redirect:/login?success"; // После регистрации
кидаем на логин
        }

        @GetMapping("/login")
        public String showLoginForm() {
            return "login";
        }

        @GetMapping("/dashboard")
        public String
showDashboard(org.springframework.security.core.Authentication
authentication) {
            boolean isHr = authentication.getAuthorities().stream()
                .anyMatch(grantedAuthority ->
grantedAuthority.getAuthority().equals("ROLE_HR"));

            if (isHr) {
                return "redirect:/hr/dashboard";
            }

            return "dashboard";
        }
    }
}

```

Листинг Б.9 – Файл ApplicationController.java

```

        this.testService = testService;
        this.aiService = aiService;
        this.validationService = validationService;
    }
}

```

```

    @GetMapping("/form")
    public String showForm(@RequestParam String track, Model
model, Authentication authentication) {
        AppUser user =
userRepository.findByEmail(authentication.getName());
        if (formRepository.findByUserAndTrack(user,
track).isPresent()) {
            return "redirect:/application/test?track=" + track;
        }
        model.addAttribute("track", track);
        model.addAttribute("form", new ApplicationForm());
        return "application-form";
    }

    @PostMapping("/form")
    public String submitForm(@ModelAttribute ApplicationForm form,
@RequestParam String track, Authentication authentication) {
        AppUser user =
userRepository.findByEmail(authentication.getName());
        form.setUser(user);
        form.setTrack(track);
        formRepository.save(form);
        return "redirect:/application/test?track=" + track;
    }

    @GetMapping("/test")
    public String showTest(@RequestParam String track, Model
model, Authentication authentication) {
        AppUser user =
userRepository.findByEmail(authentication.getName());
        ApplicationForm form =
formRepository.findByUserAndTrack(user, track).orElse(null);
        if (form == null) return
"redirect:/application/form?track=" + track;
        if (form.getTheoryScore() != null) return
"redirect:/application/task?track=" + track;

        model.addAttribute("track", track);
        model.addAttribute("questions",
testService.getQuestionsForTrack(track));
        return "test";
    }

    @PostMapping("/test")
    public String submitTest(@RequestParam String track,
@RequestParam Map<String, String> allParams, Authentication
authentication) {
        AppUser user =
userRepository.findByEmail(authentication.getName());
        ApplicationForm form =
formRepository.findByUserAndTrack(user, track).orElseThrow();

        int score = testService.calculateScore(track, allParams);

```

```

        form.setTheoryScore(score);

        StringBuilder answersLog = new StringBuilder();
        allParams.forEach((key, value) -> {
            if (!key.equals("track") && !key.equals("_csrf")) {
                answersLog.append("Вопрос ID
[").append(key).append("]: Ответ кандидата ->
").append(value).append("\n");
            }
        });
        form.setDetailedTestAnswers(answersLog.toString());

        formRepository.save(form);
        return "redirect:/application/task?track=" + track;
    }

    @GetMapping("/task")
    public String showTask(@RequestParam String track, Model
model, Authentication authentication) {
        AppUser user =
userRepository.findByEmail(authentication.getName());
        ApplicationForm form =
formRepository.findByUserAndTrack(user, track).orElse(null);

        if (form == null || form.getTheoryScore() == null) return
"redirect:/application/form?track=" + track;
        if (form.getSubmittedCode() != null) return
"redirect:/application/result/" + form.getId();

        model.addAttribute("track", track);
        return "task";
    }

    @PostMapping("/task")
    public String submitTask(
        @RequestParam String track,
        @RequestParam String code,
        @RequestParam(defaultValue = "0") Integer pasteCount,
        @RequestParam(defaultValue = "0") Long timeSpent,
        Authentication authentication) {

        AppUser user =
userRepository.findByEmail(authentication.getName());
        ApplicationForm form =
formRepository.findByUserAndTrack(user, track).orElseThrow();

        aiService.analyzeCode(form, code, pasteCount, timeSpent);

        boolean checksPassed = form.getIsAiSuspected() != null &&
!form.getIsAiSuspected();

        String validationResult =
validationService.processInternshipApplication(

```

```

        user.getEmail(),
        user.getEmail(), // Если в AppUser есть getName()
или getFullName(), замени второй аргумент на него
        form.getTheoryScore() != null,
        code != null && !code.trim().isEmpty(),
        checksPassed
    );

    System.out.println("--- РЕЗУЛЬТАТ ВАЛИДАЦИИ ЗАЯВКИ: " +
validationResult + " ---");

    formRepository.save(form);

    return "redirect:/application/result/" + form.getId();
}

@GetMapping("/result/{id}")
public String showResult(@PathVariable Long id, Model model) {
    ApplicationForm form =
formRepository.findById(id).orElseThrow();
    model.addAttribute("form", form);
    return "result";
}
}

```

Листинг Б.10 – Файл SecurityConfig.java

```

        .requestMatchers("/register", "/login",
"/css/**").permitAll() // Сюда можно всем
        .requestMatchers("/hr/**").hasRole("HR")
// Сюда только HR
        .anyRequest().authenticated() // На
остальные страницы - только после входа
    )
    .formLogin(form -> form
        .loginPage("/login") // Наша кастомная
красивая страница входа (сделаем позже)
        .defaultSuccessUrl("/dashboard", true) //
Куда кидать после успешного входа
        .permitAll()
    )
    .logout(logout -> logout.permitAll());

    return http.build();
}
}

```

Листинг Б.11 – Файл DataInitializer.java

```

Role hrRole = roleRepository.findByName("ROLE_HR");
if (hrRole == null) {
    hrRole = new Role();
    hrRole.setName("ROLE_HR");
    roleRepository.save(hrRole);
}

```

```

    }

    if (userRepository.findByEmail("hr@mail.ru") == null)
    {
        AppUser hrUser = new AppUser();
        hrUser.setEmail("hr@mail.ru");

        hrUser.setPassword(passwordEncoder.encode("admin123")); // Пароль
        для входа
        hrUser.setFullName("Главный HR Менеджер");
        hrUser.setRoles(Set.of(hrRole));
        userRepository.save(hrUser);
    }
    };
}
}

```

Приложение В
(рекомендуемое)
Листинг модульных тестов

```
public String processInternshipApplication(
    String candidateEmail,
    String candidateName,
    boolean testSelected,
    boolean codeSubmitted,
    boolean checksPassed
) {
    // Условие 1: проверка корректности email
    if (candidateEmail == null || candidateEmail.length() < 5 ||
candidateEmail.length() > 254) { // a
        return "Некорректный email"; // b
    }

    // Условие 2: проверка заполненности имени
    if (candidateName == null || candidateName.isBlank()) { // c
        return "Имя не заполнено"; // d
    }

    // Условие 3: проверка выбора теста
    if (!testSelected) { // e
        return "Тест не выбран"; // f
    }

    // Условие 4: проверка отправки кодового задания
    if (!codeSubmitted) { // g
        return "Кодовое задание не отправлено"; // h
    }

    // Условие 5: проверка прохождения проверок
    if (!checksPassed) { // i
        return "Проверки не пройдены"; // j
    }

    return "Заявка успешно создана"; // k
}
```