МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

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

Факультет	*Кафедра*
вычислительной техники	МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

Направленность Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

Проектирование и разработка автоматизированной системы поиска вакансий

Студент _____ Мальцева Валерия Олеговна
(подпись, дата) (ФИО полностью)

Руководитель _____ Макарычев П. П.
(подпись, дата) (фамилия, инициалы)

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

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

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

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

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

Пенза, 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 Мальцева Валерия Олеговна гр. 22ВП2 факультета ВТ
<Фамилия, имя, отчество полностью>

2.Тема работы Проектирование и разработка автоматизированной системы поиска вакансий

Тема утверждена приказом ПГУ №_____от "_____"_____20_____

3. Руководитель работы к.т.н., доцент кафедры «МО и ПЭВМ» Макарычев П. П.

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

1) Назначение:

2) Среда выполнения: Windows 10, Linux, Mac OS.

Языковые средства разработки: Python, SQL, UML.

Инструментальные средства: FastAPI, Docker, Nginx, SQLite.

Технологии разработки: асинхронное программирование, монолитная модульная архитектура

3) Основные функции приложения:

а) управление несколькими учетными записями hh.ru;

б) автоматический сбор вакансий по заданным поисковым URL с фильтрацией;

в) автоматическая отправка откликов с сопроводительными письмами;

г) интеграция с языковыми моделями для генерации текстов и авто-ответов;

д) мониторинг статистики просмотров резюме, приглашений и переговоров;

е) автоматическое поднятие резюме в поиске по расписанию;

ж) ведение журнала операций и экспорт данных.

5. Перечень подлежащих разработке вопросов:

1) *Анализ предметной области и постановка задачи;*
2) *Проектирование;*
3) *Реализация;*
4) *Контроль качества программного обеспечения;*
5) *Оформление пояснительной записки.*

5. 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 – 14.04.2026	
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 2026г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

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

Реферат

Пояснительная записка содержит 105 листов, 27 рисунков, 13 таблиц, 14 использованных источников и 9 приложений.

АВТОМАТИЗАЦИЯ ПОИСКА ВАКАНСИЙ, HEADHUNTER, FASTAPI, PYTHON, WEBSOCKET, ЯЗЫКОВЫЕ МОДЕЛИ, LLM, DOCKER, SPA, МОНОЛИТНАЯ АРХИТЕКТУРА, АВТОМАТИЗАЦИЯ ТРУДОУСТРОЙСТВА.

Объект исследования – процесс поиска вакансий и взаимодействия соискателя с рекрутинговыми платформами.

Цель выпускной квалификационной работы – спроектировать и разработать автоматизированную систему поиска вакансий «CareerFlow» для автоматизации операций мониторинга, анализа и обработки вакансий на платформе HeadHunter с поддержкой управления несколькими аккаунтами и интеграцией с языковыми моделями.

Технологии разработки – серверная часть реализована на Python с использованием фреймворка FastAPI, клиентская часть – в виде одностраничного веб-приложения (SPA) на JavaScript с WebSocket для обновлений в реальном времени. Хранение данных организовано в реляционной СУБД SQLite. Развертывание производится посредством контейнеризации в Docker с использованием Nginx в качестве обратного прокси.

Среда исполнения – персональный компьютер под управлением Windows 10 или Linux.

Результаты работы – спроектирована и разработана автоматизированная система поиска вакансий «CareerFlow», автоматизирующая основные операции соискателя на платформе HeadHunter: сбор вакансий, отправку откликов, взаимодействие с работодателями через языковые модели и централизованное управление несколькими аккаунтами через единый веб-интерфейс.

Содержание

Введение 7
1 Анализ предметной области и выявление требований на разработку автоматизированной системы поиска вакансий. 10
1.1 Исследование предметной области 10
1.2 Сравнительный анализ приложений для автоматизированного поиска вакансий. 11
1.3 Анализ функциональных требований 17
1.4 Анализ нефункциональных требований 22
2 Проектирование автоматизированной системы поиска вакансий 25
2.1 Архитектура приложения 25
2.2 Проектирование структур данных 27
2.2.1 Концептуальная модель данных 27
2.2.2 Логическая модель баз данных 29
2.3 Проектирование пользовательского интерфейса 32
2.4 Проектирование программных средств 37
3 Разработка программного обеспечения автоматизированной системы 40
3.1 Разработка архитектуры программного обеспечения 40
3.2 Организация хранения данных 41
3.3 Разработка архитектурных слоев системы 42
3.3.1 Слой представления данных 42
3.3.2 Разработка слоя бизнес-логики 43
3.3.3 Разработка слоя доступа к данным 43
3.4 Реализация ключевых бизнес-процессов 44
3.4.1 Процесс сбора и фильтрации вакансий 44
4 Контроль качества программного обеспечения 47
4.1 Обоснование выбора модели тестирования и метрик кода 47
4.2 Функциональное тестирование отправки отклика на вакансию 48
4.2.1 Тестирование методом черного ящика 49
4.2.2 Оценка сложности программных средств по метрикам Холстеда 52

4.2.3 Автоматизированное тестирование 54
4.3 Планирование разработки и оценка бюджета 56
Заключение 59
Список использованных источников 61
Приложение А (справочное) Глоссарий 63
Приложение Б (рекомендуемое) Спецификация атрибутов таблиц 66
Приложение В (обязательное) Объекты обращения к базам данных 71
Приложение Г (обязательное) SQL-скрипты 75
Приложение Д (обязательное) Структуры слоя представления данных 82
Приложение Е (обязательное) Структуры слоя бизнес-логики 88
Приложение Ж (обязательное) Структуры слоя обращения к базе данных 92
Приложение И (рекомендуемое) Листинг модульных тестов 98
Приложение К (рекомендуемое) Иллюстрационные материалы
к тестированию API 102

Введение

В последние годы поиск аботы практически полностью переместился в цифровую среду. Основной площадкой для трудоустройства в России является платформа HeadHunter (hh.ru), на которой ежедневно публикуются десятки тысяч новых вакансий [1]. Для успешного поиска работы соискателю необходимо регулярно отслеживать появление новых предложений, анализировать требования работодателей, подготавливать сопроводительные письма, своевременно отправлять отклики и поддерживать коммуникацию с рекрутерами.

На практике значительная часть времени тратится не на оценку вакансий и принятие решений, а на выполнение повторяющихся действий. Пользователь вынужден вручную просматривать большое количество предложений, применять одинаковые фильтры, отправлять отклики, отслеживать их статус и отвечать на сообщения работодателей. Особенно заметна данная проблема при активном поиске работы, когда количество вакансий исчисляется сотнями, а скорость реакции напрямую влияет на вероятность получения приглашения на собеседование.

Платформа HeadHunter предоставляет встроенные средства автоматизации, однако их возможности ограничены [2]. Пользователь не может гибко управлять логикой поиска, использовать несколько аккаунтов одновременно, автоматически обрабатывать сообщения работодателей или заполнять опросники вакансий. Существующие сторонние решения также ориентированы преимущественно на отдельные этапы процесса трудоустройства и не обеспечивают комплексного управления всеми действиями соискателя в рамках единой системы.

Актуальность работы обусловлена необходимостью повышения эффективности поиска работы за счёт автоматизации рутинных операций, связанных с мониторингом вакансий, отправкой откликов и взаимодействием с работодателями. Использование современных технологий позволяет сократить

временные затраты пользователя и обеспечить более оперативное реагирование на новые предложения работодателей.

Объектом исследования является процесс поиска вакансий и взаимодействия соискателя с рекрутинговыми платформами.

Предметом исследования являются методы, модели и программные средства автоматизации поиска вакансий, отправки откликов и обработки результатов взаимодействия с работодателями.

Целью выпускной квалификационной работы является проектирование и разработка автоматизированной системы поиска вакансий «CareerFlow», предназначенной для автоматизации поиска вакансий, массовой отправки откликов, мониторинга активности работодателей и управления несколькими аккаунтами платформы HeadHunter.

Для достижения поставленной цели необходимо решить следующие задачи:

– исследовать предметную область поиска вакансий и определить основные этапы процесса трудоустройства с использованием электронных платформ;

– провести сравнительный анализ существующих программных решений для автоматизации поиска работы и выявить их ограничения;

– сформулировать функциональные и нефункциональные требования к разрабатываемой системе;

– спроектировать архитектуру программного обеспечения и определить структуру его основных компонентов;

– разработать концептуальную и логическую модели данных для хранения информации об аккаунтах, вакансиях, откликах и статистике;

– реализовать серверную часть системы с модулями автоматического сбора вакансий, отправки откликов и интеграции с языковыми моделями;

– разработать пользовательский интерфейс в виде одностраничного веб-приложения с поддержкой обновлений в реальном времени;

– выполнить тестирование реализованного программного продукта и оценить качество кода с использованием метрик Холстеда;

– произвести планирование разработки и оценку бюджета проекта.

При выполнении работы использовались методы анализа предметной области, сравнительного анализа программных решений с применением метода взвешенной экспертной оценки, методы структурного и объектно-ориентированного проектирования, а также средства UML-моделирования. Для контроля качества разработанного программного обеспечения применялись методы функционального, модульного и интеграционного тестирования.

Практическая значимость работы заключается в создании программного продукта, обеспечивающего комплексную автоматизацию поиска работы на платформе HeadHunter. Разработанная система позволяет автоматически выполнять поиск вакансий по заданным критериям, отправлять отклики, вести учёт результатов взаимодействия с работодателями, отслеживать статистику резюме и использовать языковые модели для автоматизации коммуникации. Применение системы позволяет существенно сократить время, затрачиваемое на поиск работы, повысить оперативность обработки новых вакансий и обеспечить более эффективное управление процессом трудоустройства.

1 Анализ предметной области и выявление требований на разработку автоматизированной системы поиска вакансий.

1.1 Исследование предметной области

В основе предметной области выпускной квалификационной работы лежит процесс трудоустройства и взаимодействие соискателей с рекрутинговыми платформами. Рассмотрим основные термины и понятия выбранной области для детального погружения в специфику задачи.

Поиск работы – процесс взаимодействия соискателя с рынком труда, направленный на нахождение подходящей вакансии и последующее трудоустройство.

Рекрутинговая платформа (агрегатор вакансий) – специализированный интернет-сервис, предоставляющий инструменты для публикации вакансий, размещения резюме и осуществления коммуникации между работодателями и кандидатами. В российском сегменте рынка труда ключевую роль в данном сегменте занимает платформа HeadHunter (hh.ru).

Резюме – структурированный документ, содержащий информацию о профессиональном опыте, навыках и образовательном уровне соискателя.

Вакансия – описание открытой позиции у работодателя, включающее перечень требований к кандидату, условия труда и размер предлагаемого вознаграждения.

Отклик – действие соискателя, выражающее заинтересованность в вакансии и инициирующее процесс рассмотрения его кандидатуры работодателем.

Сопроводительное письмо – текстовое сообщение, прилагаемое к отклику. Оно раскрывает мотивацию кандидата, а также соответствие его навыков требованиям конкретной вакансии.

Мониторинг вакансий – регулярный анализ новых предложений на рекрутинговых платформах с целью своевременного отклика на актуальные позиции.

Автоматизация поиска работы – применение программных средств для выполнения рутинных операций процесса трудоустройства. К таким операциям относятся массовая отправка откликов, отслеживание статусов и обработка входящих сообщений.

Программные средства для автоматизации поиска вакансий разрабатываются с целью снижения временных затрат соискателей на рутинные операции. Они позволяют централизованно управлять процессом трудоустройства, вести аналитику откликов и делегировать первичную коммуникацию алгоритмам, сохраняя при этом полный контроль над взаимодействием с работодателями.

1.2 Сравнительный анализ приложений для автоматизированного поиска вакансий.

Проанализировав рынок программных средств, схожих по функциональности с разрабатываемым в рамках выпускной квалификационной работы приложением, были выделены следующие сервисы для проведения сравнения: Quick Offer, НейроКарьера, JobTurbo и встроенный механизм автоматических откликов платформы HeadHunter. Целью данного анализа являлось выявление наиболее подходящего программного решения, которое может быть использовано в качестве прототипа при проектировании собственной системы. В таблице 1 представлено краткое описание общих сведений о выделенных аналогах.

Таблица 1 – Краткое описание общих сведений об аналогах

Наименование	**Производитель**	**Описание**
Quick Offer	Облачный сервис	Платформа для автоматизации поиска вакансий и массовой отправки откликов с использованием элементов искусственного интеллекта для генерации текста.
НейроКарьера	SaaS-платформа	Сервис, ориентированный на применение нейросетей для анализа вакансий, адаптации резюме и формирования сопроводительных писем.

Продолжение таблицы 1

Наименование	**Производитель**	**Описание**
JobTurbo	Независимые разработчики	Приложение для ускорения массовой отправки откликов, делающее основной акцент на автоматизации действий пользователя и гибких фильтрах.
Автоотклик hh.ru	HH.ru (встроенная функция)	Штатный механизм платформы, позволяющий настроить параметры поиска и включить автоматическую отправку откликов без стороннего ПО.

Каждое из представленных приложений включает в себя различный функционал и возможности для взаимодействия с сервисом. Анализ и сравнение с аналогами включает в себя выделение плюсов и минусов каждого из выбранных программных средств.

Quick Offer – облачная платформа для автоматизации поиска вакансий и массовой отправки откликов. Пользователь задаёт параметры поиска, после чего система самостоятельно выполняет подбор и отправку [3]. Графический интерфейс сервиса представлен на рисунке 1.

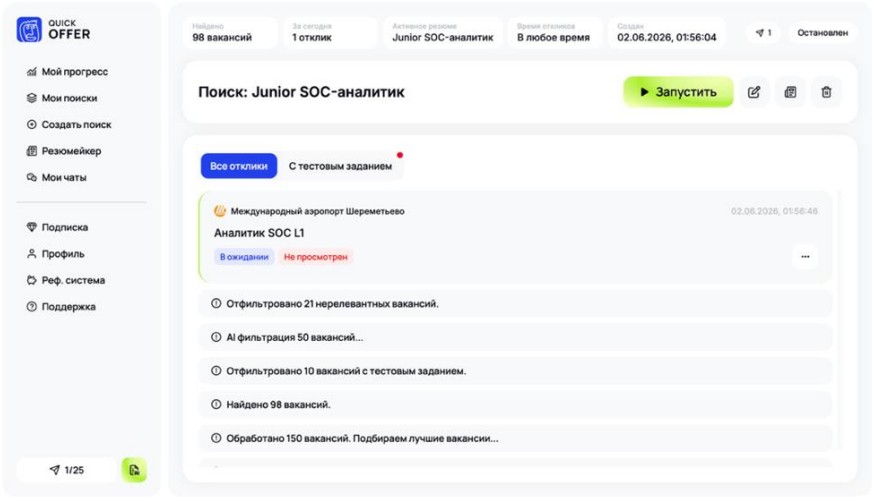


Рисунок 1 – Интерфейс сервиса Quick Offer

К основным плюсам Quick Offer можно отнести:

– высокий уровень автоматизации процесса трудоустройства;

– наличие функции генерации сопроводительных писем с помощью ИИ.

К минусам можно отнести:

– закрытость алгоритмов работы системы;

– отсутствие возможности детально управлять настройками обработки вакансий.

НейроКарьера – сервис для помощи в трудоустройстве с упором на технологии искусственного интеллекта. Система выполняет анализ вакансий и помогает адаптировать резюме [4]. Графический интерфейс сервиса представлен на рисунке 2.

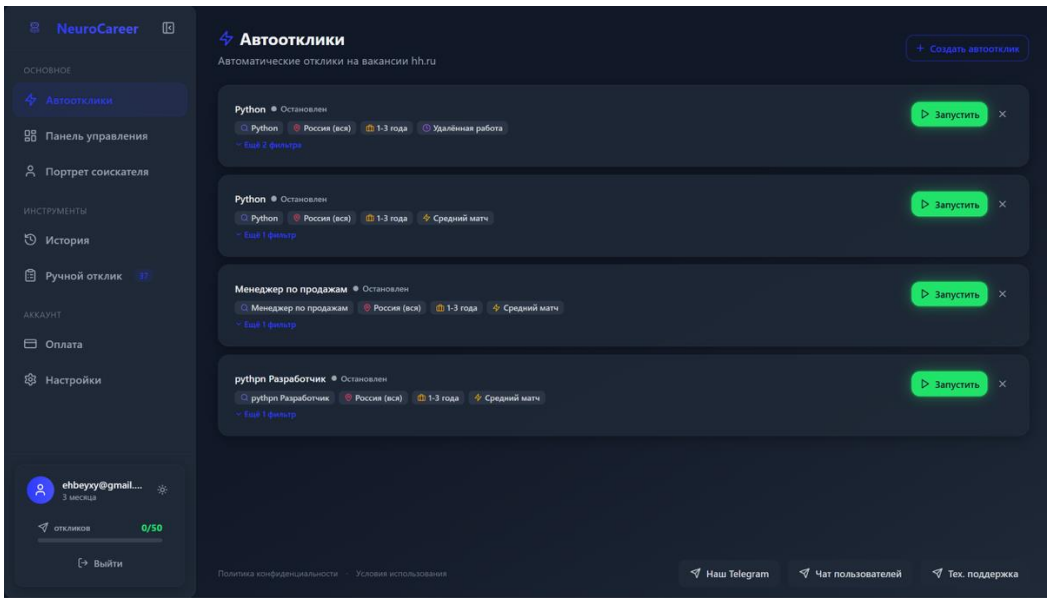


Рисунок 2 – Интерфейс сервиса НейроКарьера

К основным преимуществам НейроКарьеры можно отнести:

– помощь в адаптации резюме под конкретные вакансии;

– упрощенный процесс использования за счет выноса вычислений на удаленные серверы.

К основным недостаткам можно отнести:

– ограниченные возможности настройки системы под индивидуальные требования;

– зависимость от внешних серверов разработчика.

JobTurbo – приложение для ускорения массовой отправки откликов на вакансии. Основной акцент сделан на автоматизации действий пользователя и сокращении времени обработки большого количества вакансий [5]. Сервис

предоставляет расширенные фильтры для точной настройки параметров поиска, включая критерии по опыту работы, типу занятости и уровню заработной платы. Особенностью JobTurbo является возможность пакетной обработки вакансий с автоматическим заполнением стандартных полей отклика на основе заранее подготовленных шаблонов. Графический интерфейс сервиса представлен на рисунке 3.

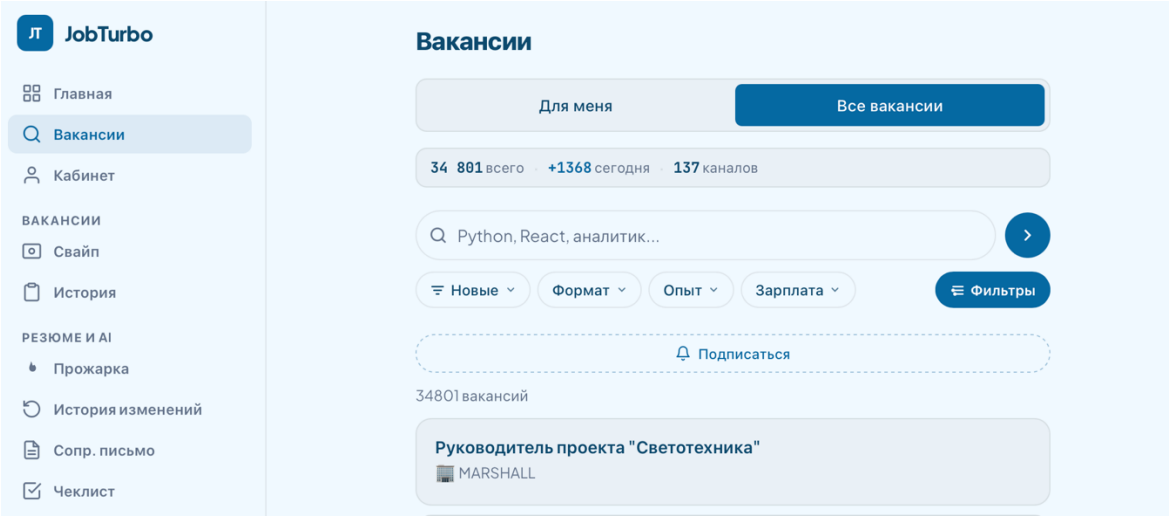


Рисунок 3 – Интерфейс сервиса JobTurbo

К основным преимуществам JobTurbo можно отнести:

– широкий набор параметров настройки по сравнению с другими аналогами;

– сокращение времени обработки большого количества вакансий. Основными недостатками JobTurbo являются:

– ограниченные средства аналитики и статистики;

– отсутствие поддержки интеллектуальной обработки взаимодействия с работодателями.

Встроенный механизм автооткликов hh.ru интегрирован непосредственно в сервис поиска работы и не требует установки дополнительного программного обеспечения [6]. Графический интерфейс настройки представлен на рисунке 4.

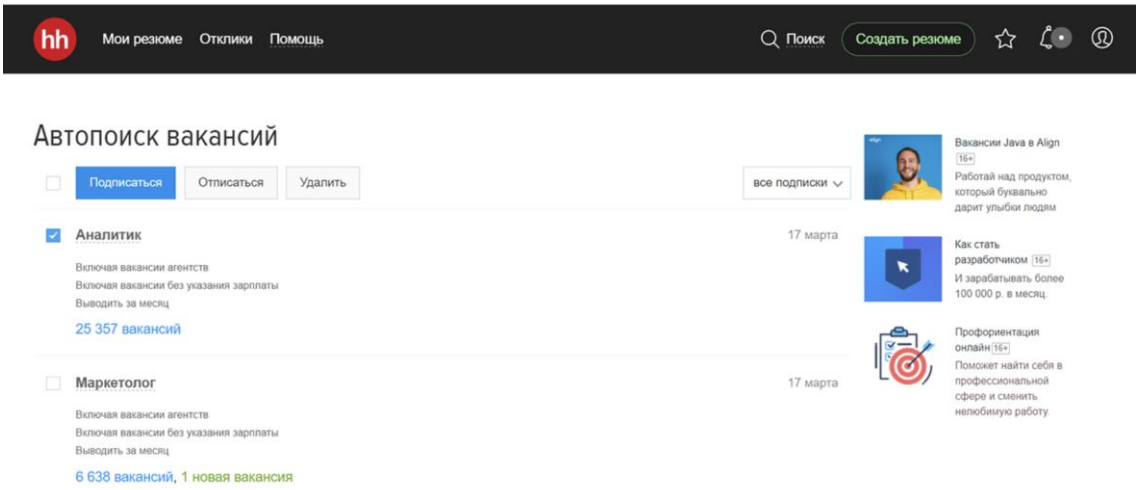


Рисунок 4 – Интерфейс настройки автооткликов hh.ru

К основным преимуществам данного решения можно отнести:

– отсутствие необходимости установки дополнительного программного обеспечения;

– полная совместимость с правилами платформы.

К основным недостаткам можно отнести:

– сильно ограниченный набор настроек поиска;

– наличие специальной отметки для работодателя, указывающей на автоматический характер отклика.

Для проведения сравнительного анализа и обоснования выбора прототипа был использован метод анализа иерархий (МАИ), разработанный Т. Саати [7]. Данный метод представляет собой систематическую процедуру принятия решений, основанную на построении матриц парных сравнений и расчёте приоритетов альтернатив.

В рамках данного исследования было выделено 4 ключевых критерия сравнения:

– К1 – автоматический поиск вакансий;

– К2 – автоматическая отправка откликов;

– К3 – гибкость настройки параметров поиска;

– К4 – использование технологий искусственного интеллекта.

Для определения относительной важности каждого критерия была построена матрица парных сравнений размером 4×4, в которой элементы отражают степень предпочтения одного критерия перед другим по шкале Саати (от 1 – равная важность, до 9 – абсолютное превосходство). После нормализации матрицы и усреднения по строкам были получены веса критериев, представленные в таблице 2.

Таблица 2 – Расчёт весов критериев сравнения

Критерий	К1	К2	К3	К4	Вес
К1. Автоматический поиск вакансий	1	1/3	2	4	0,2326
К2. Автоматическая отправка откликов	3	1	5	7	0,5791
К3. Гибкость настройки параметров	1/2	1/5	1	2	0,1213
К4. Использование технологий ИИ	1/4	1/7	1/2	1	0,0670
Сумма					1,0000

Наибольшую важность получил критерий К2 (автоматическая отправка откликов) с весом 0,5791, что отражает ключевую роль данной функции в процессе автоматизации поиска работы. Критерий К1 (автоматический поиск) получил вес 0,2326, критерий К3 (гибкость настроек) – 0,1213, а критерий К4 (технологии ИИ) – 0,0670.

Для каждого из 4 критериев была построена матрица парных сравнений аналогов размером 4×4, отражающая относительную эффективность каждого программного решения по данному критерию. В таблице 3 представлен пример расчёта локальных приоритетов для критерия К1 (автоматический поиск вакансий).

Таблица 3 – Расчёт приоритетов для критерия К1

Решение	Quick Offer	НейроКарьера	JobTurbo	Автоотклик hh.ru	Приоритет
Quick Offer	1	2	1	2	0,3333
НейроКарьера	1/2	1	1/2	1	0,1667
JobTurbo	1	2	1	2	0,3333
Автоотклик hh.ru	1/2	1	1/2	1	0,1667

Аналогичным образом были рассчитаны локальные приоритеты аналогов по остальным трём критериям (К2, К3, К4).

Итоговая оценка каждого решения рассчитывалась как взвешенная сумма локальных приоритетов по всем критериям:

$$R_j = \sum_{i=1}^{4} W_i \times P_{ij},$$

где W_i – вес i-го критерия; P_{ij} – локальный приоритет j-го аналога по i-му критерию.

После выполнения расчётов были получены результаты, представленные в таблице 4.

Таблица 4 – Итоговые результаты сравнения

Решение	Итоговая оценка
Quick Offer	0,2987
НейроКарьера	0,1844
JobTurbo	**0,3144**
Автоотклик hh.ru	0,2025

Таким образом, по итогам сравнительного анализа сервис **JobTurbo** получил наибольшую итоговую оценку (0,3144) и был выбран в качестве прототипа для разрабатываемой системы. Он обеспечивает базовую автоматизацию откликов и обладает гибкими настройками поиска, что соответствует наиболее важным критериям (К2 и К3).

Однако анализ выявил ряд ограничений этого решения: отсутствие полноценной поддержки нескольких аккаунтов, невозможность автоматической обработки входящих сообщений и опросников с помощью ИИ, а также слабые средства аналитики. Именно эти недостатки прототипа определяют направления для улучшения, которые будут сформулированы как требования к разрабатываемой системе в следующем подразделе.

1.3 Анализ функциональных требований

На основе исследования предметной области и сравнительного анализа аналогов (где в качестве прототипа был выбран сервис JobTurbo) были сформированы требования к разрабатываемой системе. Главная цель разработки

– создание инструмента, который не только автоматизирует отправку откликов, но и обеспечивает централизованный контроль над всем циклом поиска работы, устраняя недостатки существующих решений. В частности, разрабатываемая система должна компенсировать отсутствие в JobTurbo поддержки нескольких аккаунтов, автоматической обработки сообщений работодателей и интеграции с языковыми моделями для генерации текстов. Реализация указанных функций позволит свести к минимуму ручное вмешательство пользователя в рутинные операции и повысить конверсию откликов за счет персонализации коммуникации. В результате формируется не просто инструмент для массовой рассылки, а единая среда управления трудоустройством, обеспечивающая прозрачность и аналитику всех этапов взаимодействия с работодателями.

Для структурирования требований были выделены основные классы пользователей и внешних систем, взаимодействующих с приложением:

– соискатель (Пользователь системы) – физическое лицо, которое настраивает параметры поиска, управляет аккаунтами hh.ru, контролирует процесс отправки откликов и анализирует статистику;

– внешняя платформа (hh.ru) – источник данных о вакансиях и среда, в которой эмулируются действия пользователя (авторизация, отклики, переписка);

– сторонние сервисы (API языковых моделей) – используются для генерации текста сопроводительных писем и ответов на сообщения работодателей.

С учетом выявленных ограничений прототипа, функциональные требования были сгруппированы по ключевым подсистемам:

Управление аккаунтами и сессиями:

– добавление, редактирование и удаление учетных записей hh.ru;

– безопасное хранение и поддержка активных пользовательских сессий для каждого аккаунта;

– обеспечение одновременной независимой работы нескольких аккаунтов в рамках единого интерфейса.

Сбор и обработка вакансий:

– настройка профилей поиска (ключевые слова, URL-адреса, фильтры по зарплате, опыту, формату занятости);

– автоматический парсинг вакансий по заданным критериям с исключением ранее обработанных позиций;

– просмотр, фильтрация, сортировка и постраничный вывод списка вакансий через веб-интерфейс.

Взаимодействие с работодателями:

– автоматическая и ручная отправка откликов на выбранные вакансии;

– формирование сопроводительных писем на основе пользовательских шаблонов или с генерацией текста через языковую модель (LLM);

– автоматическое заполнение стандартных опросников, прикрепленных к вакансиям;

– обработка входящих сообщений от работодателей и генерация контекстных ответов с использованием LLM.

Мониторинг и аналитика:

– автоматическое поднятие резюме в поиске по заданному расписанию;

– сбор и агрегация статистики: просмотры резюме, приглашения, отказы, статусы откликов;

– ведение детального журнала операций и событий системы в реальном времени;

– экспорт собранных данных во внешние файлы.

Для визуализации взаимодействия акторов с системой была разработана диаграмма вариантов использования. Она помогает верхнеуровнево отобразить связи между пользователями и необходимыми сценариями, не углубляясь во внутреннее устройство системы [8]. Диаграмма вариантов использования представлена на рисунке 5.

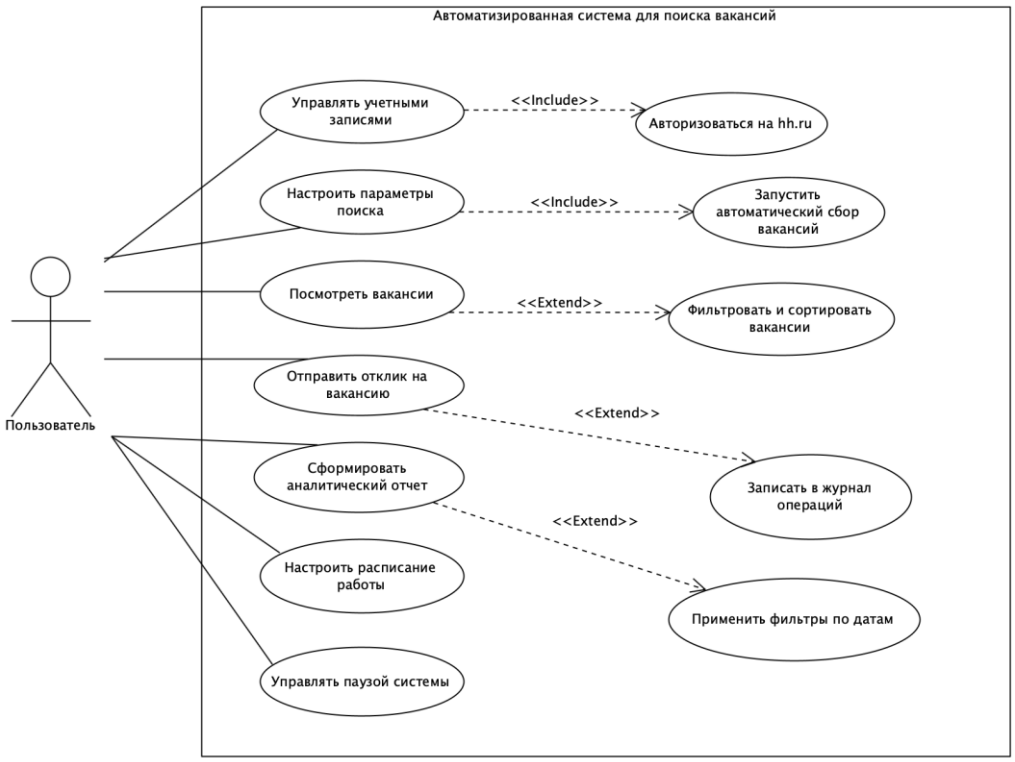


Рисунок 5 – Диаграмма вариантов использования системы

Для детализации требований были разработаны спецификации наиболее критичных сценариев работы системы.

Сценарий варианта использования «Управлять учётными записями» представлен в таблице 5.

Таблица 5 – Сценарий варианта использования «Управлять учётными записями»

Прецедент: Управлять учётными записями
ID: 1
Краткое описание: Пользователь добавляет, редактирует и удаляет аккаунты hh.ru, а также инициирует поддержание активных сессий.
Главные роли: Соискатель
Второстепенные актеры: Платформа hh.ru
Предусловие: 1. Система запущена. 2. Пользователь авторизован в веб-интерфейсе приложения.

Продолжение таблицы 5

Основной поток: 1. Пользователь открывает раздел «Учётные записи». 2. Система отображает список добавленных аккаунтов с их текущими статусами. 3. Пользователь выбирает действие: добавить, отредактировать или удалить аккаунт. 4. При добавлении пользователь вводит учётные данные hh.ru. 5. Система выполняет фоновую авторизацию на платформе hh.ru и сохраняет токены сессии. 6. Система обновляет список, отображая статус «Активен».
Постусловие: Список учётных записей актуализирован. Сессия на hh.ru активна и готова к использованию в сценариях поиска.

Следующим по приоритету сценарием является запуск фонового сбора вакансий, который инициируется системой на основе настроенных профилей поиска и использует активную сессию, полученную на предыдущем этапе.

Таблица 6 – Сценарий варианта использования «Запустить автоматический сбор вакансий»

Прецедент: Запустить автоматический сбор вакансий
ID: 2
Краткое описание: Система выполняет фоновый поиск и сохранение вакансий с платформы hh.ru по заданным критериям.
Главные роли: Система
Второстепенные актеры: Платформа hh.ru
Предусловие: 1. Существует хотя бы один активный профиль поиска. 2. Сессия связанного аккаунта hh.ru активна.
Основной поток: 1. Система считывает параметры из профиля поиска. 2. Выполняется запрос к API/страницам hh.ru с учётом заданных фильтров. 3. Система получает список вакансий и сверяет его с локальной базой данных, исключая дубликаты. 4. Новые вакансии сохраняются в базе с отметкой времени и статусом «Не обработана». 5. Система обновляет счётчик новых вакансий в пользовательском интерфейсе.
Постусловие: Локальная база вакансий актуализирована. Данные доступны для просмотра и фильтрации.

После того как вакансии собраны в локальную базу и доступны для просмотра, пользователь может перейти к следующему ключевому сценарию – отправке откликов на найденные позиции с использованием шаблонов или генерацией текста через языковую модель.

Таблица 7 – Сценарий варианта использования «Отправить отклик на вакансию»

Прецедент: Отправить отклик на вакансию
ID: 3
Краткое описание: Пользователь формирует и отправляет отклик на выбранную вакансию с сопроводительным письмом, система записывает результат в журнал операций
Главные роли: Соискатель
Второстепенные актеры: Платформа hh.ru
Предусловие: 1. В базе данных есть вакансии 2. Учетная запись hh.ru 3. Настроен хотя бы один шаблон сопроводительного письма или подключена языковая модель
Основной поток: 1. Пользователь выбирает вакансию из списка 2. Пользователь инициирует отправку отклика 3. Система предлагает выбрать шаблон сопроводительного письма или сгенерировать письмо с помощью языковой модели 4. Пользователь выбирает способ формирования письма 5. Система формирует текст письма на основе шаблона или запроса к языковой модели 6. Система отображает предпросмотр сформированного отклика 7. Пользователь подтверждает отправку 8. Система отправляет отклик через интеграцию с hh.ru 9. Система записывает результат отклика в журнал операций
Постусловие: 1. Отклик отправлен на платформу hh.ru 2. Запись об отклике добавлена в журнал операций 3. Статус отклика доступен для последующего контроля

Проведенный анализ позволил определить минимальный набор функций, необходимый для корректной работы всех бизнес-сценариев. Описанные прецеденты отражают реальные процессы поиска работы и служат основой для проектирования архитектуры и структур данных.

1.4 Анализ нефункциональных требований

Нефункциональные требования определяют характеристики качества разрабатываемой системы и устанавливают технические ограничения, связанные с её эксплуатацией. При их формулировании учитывалась специфика взаимодействия с внешними платформами, имеющими строгие лимиты и защиту от автоматизации.

Были сформированы следующие ключевые нефункциональные требования:

– безопасность: изоляция данных разных учетных записей, шифрованное хранение параметров авторизации (cookies/tokens), контроль активных сессий и строгое ограничение частоты запросов к внешним API для предотвращения блокировки аккаунтов со стороны hh.ru;

– производительность: выполнение операций поиска и отправки откликов в фоновом режиме без блокировки пользовательского интерфейса, время отклика интерфейса при фильтрации и сортировке не должно превышать 2 секунд;

– надёжность: корректная обработка ошибок соединения с внешними сервисами, механизм автоматического восстановления упавших сессий hh.ru, сохранение накопленных данных и состояния задач при перезапуске системы;

– масштабируемость: возможность одновременной работы с несколькими учетными записями и увеличения объёма обрабатываемых вакансий без необходимости изменения базовой архитектуры приложения;

– сопровождаемость: наличие модульной структуры программного обеспечения, позволяющей развивать и модифицировать отдельные компоненты независимо друг от друга;

– развёртываемость и переносимость: возможность контейнеризации приложения и его запуска на различных программно-аппаратных платформах без существенных изменений конфигурации;

– удобство использования: минимизация количества действий для настройки поиска, наглядное цветовое кодирование статусов операций и адаптивный веб-интерфейс, корректно отображающийся в современных браузерах.

Опираясь на сформулированные требования, был обоснован выбор технологического стека. Требование модульности и фоновой обработки задач определило выбор серверной части на языке Python с использованием

асинхронного фреймворка FastAPI, который обеспечивает высокую производительность при работе с сетевыми запросами [9].

Необходимость переносимости и простоты развертывания привела к решению использовать контейнеризацию через Docker Compose [10] (включая основной контейнер приложения, фоновый worker и прокси-сервер nginx). Для хранения структурированных данных о вакансиях, откликах и логах выбрана встраиваемая реляционная база данных SQLite, что снижает порог входа для развертывания системы на персональном компьютере пользователя. Клиентская часть реализована как одностраничное веб-приложение (SPA) на HTML, CSS и JavaScript, что исключает необходимость установки специализированного клиентского ПО.

Сформулированные нефункциональные требования задают критерии качества и будут использоваться в дальнейшем при контроле качества программного обеспечения и тестировании системы.

2 Проектирование автоматизированной системы поиска вакансий

2.1 Архитектура приложения

На основе анализа функциональных и нефункциональных требований для разрабатываемой системы была выбрана монолитная модульная архитектура с трехуровневым разделением. Такой подход предполагает построение приложения как единого программного комплекса, внутри которого функциональность разделена на независимые логические модули, взаимодействующие через внутренние интерфейсы [11].

Выбор монолитной архитектуры обусловлен спецификой проекта: система предназначена для локального развертывания на персональном компьютере или единичном сервере, где отсутствует необходимость в независимом масштабировании отдельных компонентов. При этом внутри приложения выделены три уровня.

Слой представления реализован в виде одностраничного веб-приложения (SPA). Пользователь получает доступ ко всем функциям системы через браузер без установки дополнительного программного обеспечения.

Слой бизнес-логики включает набор специализированных модулей, отвечающих за выполнение основных бизнес-процессов. Центральным компонентом является менеджер управления задачами, который координирует работу отдельных аккаунтов HeadHunter и распределяет фоновые операции между рабочими потоками.

Слой хранения данных обеспечивает сохранение конфигурации системы, истории вакансий, откликов, статистики и служебной информации.

Взаимодействие компонентов организовано следующим образом. Пользователь через веб-браузер обращается к интерфейсу Dashboard, который загружает статические файлы через веб-сервер и устанавливает WebSocket-соединение с сервером приложений для получения обновлений в реальном времени. Сервер обрабатывает HTTP-запросы, обращается к хранилищу данных и инициирует фоновые задачи в слое автоматизации. Слой автоматизации через

модуль интеграции с hh.ru выполняет сбор вакансий или отправку откликов, а через модуль интеграции с языковыми моделями генерирует ответы работодателям. Результаты операций сохраняются в хранилище и передаются пользователю через WebSocket.

Дополнительно система поддерживает отправку уведомлений через мессенджер, что позволяет контролировать процесс поиска работы вне веб-интерфейса.

Развертывание приложения выполняется с использованием технологии контейнеризации. Для управления контейнерами применяется оркестратор, обеспечивающий автоматический запуск веб-сервера, прикладного сервера и вспомогательных компонентов. Веб-сервер является единственной точкой входа, обеспечивает TLS-терминацию и обратное проксирование запросов к прикладному серверу. На рисунке 6 представлена диаграмма развертывания системы.

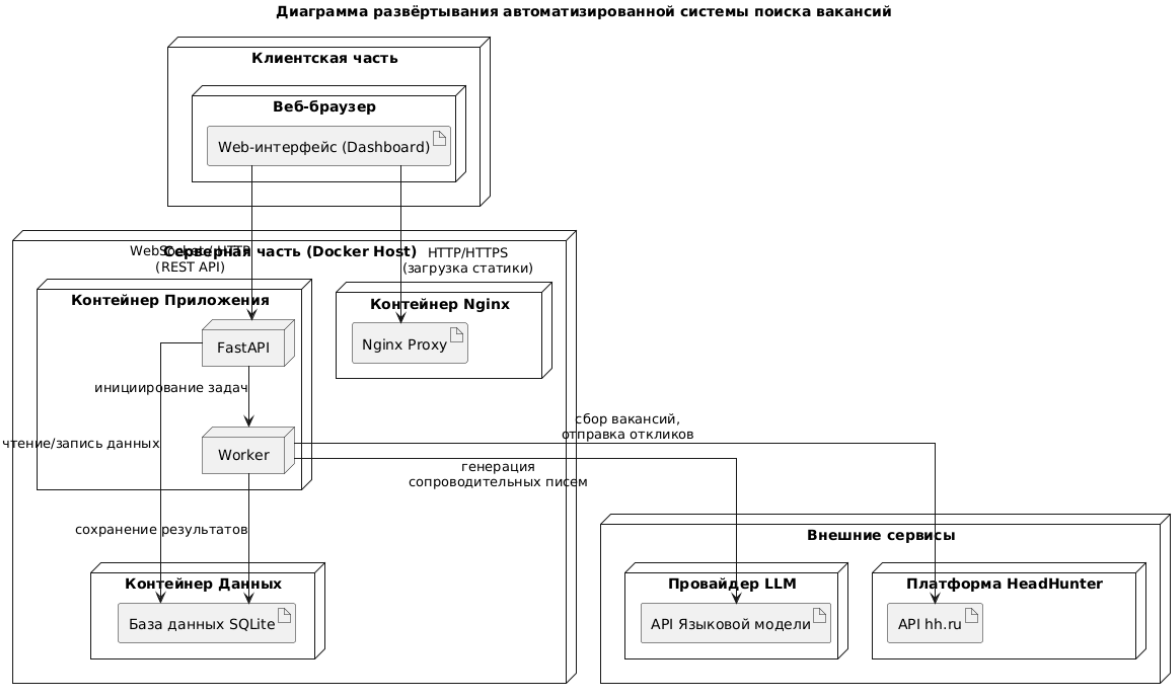


Рисунок 6 – Диаграмма развертывания

Данная диаграмма наглядно отражает физическое распределение программных компонентов по узлам и контейнерам, а также каналы передачи данных между ними.

2.2 Проектирование структур данных

2.2.1 Концептуальная модель данных

Концептуальная модель данных используется для формализации структуры предметной области на раннем этапе проектирования. Такая модель отражает ключевые сущности и связи между ними независимо от конкретной реализации в хранилище [12].

Для автоматизированной системы поиска вакансий была разработана концептуальная модель, включающая следующие основные сущности:

– учетная запись hh.ru – содержит данные авторизации на платформе (cookies сессии, идентификатор резюме, индивидуальные настройки поиска);

– вакансия – хранит информацию о найденных вакансиях (название, компания, требования, ссылка, дата обнаружения);

– отклик – содержит данные об отправленных откликах (дата, статус, текст сопроводительного письма, метаданные вакансии);

– вакансия с тестом – вакансии, требующие прохождения опросника или тестового задания от работодателя;

– переговоры – статус взаимодействия с работодателями (интервью, просмотры, отказы, возможные предложения);

– шаблон письма – тексты сопроводительных писем и опросников для различных категорий вакансий;

– настройки поиска – пул поисковых URL, фильтры по зарплате, лимиты откликов, параметры интеграции с языковыми моделями;

– журнал операций – события системы (поиск, отправка откликов, ошибки, ответы языковых моделей).

Концептуальная модель отражает следующие связи между сущностями:

– учетная запись связана с откликами (1:M);

– вакансия связана с откликами (1:M);

– шаблон письма может использоваться для множества откликов (1:M);

– настройки поиска относятся к системе в целом (1:1);

– журнал операций связан с учетной записью, над которой выполнялась операция (M:1);

– переговоры связаны с учетной записью (M:1).

На рисунке 7 представлена диаграмма "сущность-связь" концептуальной модели.

Рисунок 7 – Концептуальная модель базы данных

Диаграмма, разработанная в процессе проектирования концептуальной модели базы данных, наглядно отображает все сущности и связи между ними. В таблице 8 представлено описание сущностей, которые были выделены для организации работы автоматизированной системы поиска вакансий.

Таблица 8 – Описание сущностей

Название сущности	Идентификатор	Назначение
Учетная запись hh.ru	accounts	Хранит данные авторизации и настройки аккаунта
Вакансия	vacancies	Хранит найденные вакансии
Отклик	applications	Хранит отправленные отклики
Вакансия с тестом	test_vacancies	Хранит вакансии, требующие прохождения опросника
Переговоры	interviews	Хранит статус переговоров с работодателями
Шаблон письма	letter_templates	Хранит тексты сопроводительных писем
Настройки поиска	search_configs	Хранит параметры поиска и лимиты
Журнал операций	logs	Хранит события системы

Построенная концептуальная модель служит основой для перехода к логическому проектированию, на котором будут определены конкретные атрибуты сущностей, типы данных, первичные и внешние ключи, а также нормализована структура базы данных для обеспечения целостности и минимизации избыточности информации.

2.2.2 Логическая модель баз данных

Логическая модель данных представляет собой развитие концептуальной модели и предназначена для формализации структуры хранения информации на уровне атрибутов и связей [13].

В проекте применена реляционная СУБД SQLite. Выбор обусловлен простотой развёртывания без необходимости настройки отдельного серверного процесса, достаточной производительностью при объёме данных до сотен тысяч записей и удобством работы со структурированной информацией. SQLite хранит всю базу данных в едином файле, что обеспечивает переносимость между средами выполнения и упрощает резервное копирование. Такой подход соответствует специфике приложения, ориентированного на автономное развёртывание.

База данных системы CareerFlow состоит из девяти таблиц, разделённых на две группы по характеру зависимостей. Центральной сущностью является таблица учётных записей, с которой через механизм внешних ключей связаны три зависимые таблицы: история отправленных откликов, вакансии с

обязательным тестированием и переговоры с работодателями. Остальные пять таблиц – глобальная конфигурация системы, поисковые адреса, браузерные сессии, шаблоны сопроводительных писем и шаблоны ответов на опросники – являются независимыми и не имеют внешних ключей на таблицу аккаунтов.

Логическая схема базы данных представлена на рисунке 8.

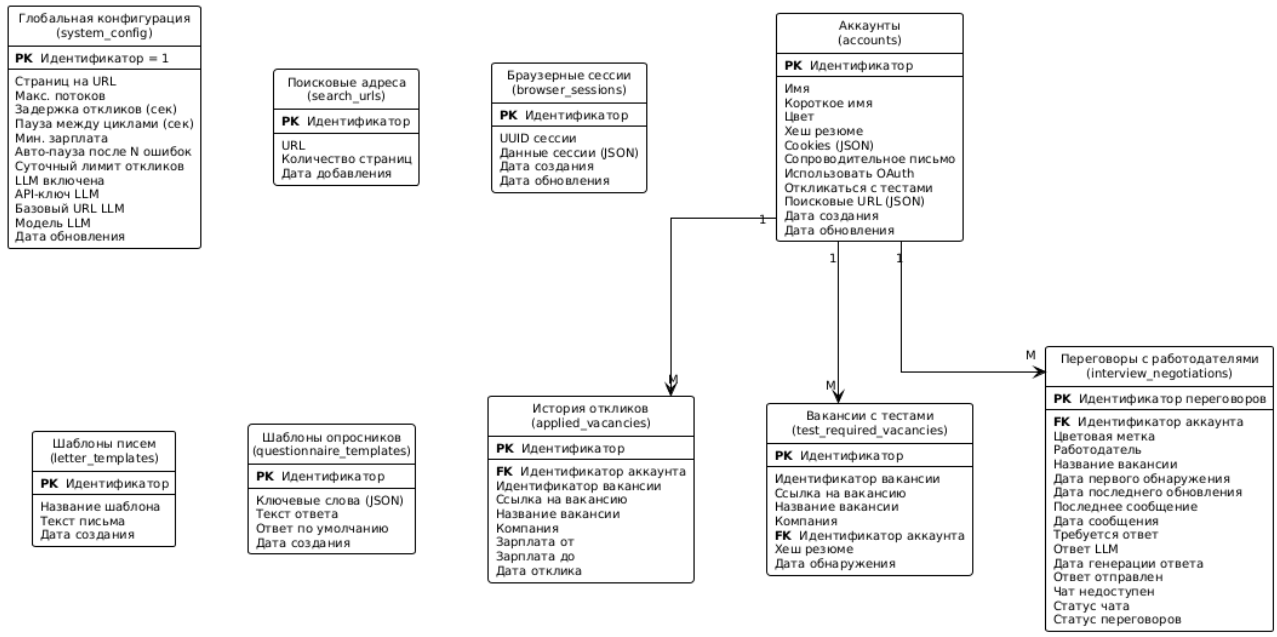


Рисунок 8 – Логическая схема базы данных системы CareerFlow

Таблица учётных записей хранит полный набор параметров каждого аккаунта hh.ru: отображаемое имя, цветовой индикатор для интерфейса, хеш резюме, сериализованные cookies сессии, сопроводительное письмо, индивидуальный пул поисковых адресов, а также флаги использования OAuth и режима работы с вакансиями, требующими прохождения теста. Уникальные ограничения на полях имени и хеша резюме предотвращают дублирование записей.

Таблица истории откликов фиксирует каждый отправленный отклик с привязкой к аккаунту, идентификатором и ссылкой на вакансию, названием, компанией, диапазоном зарплаты и временной меткой. Таблица вакансий с тестами регистрирует вакансии, пропущенные системой из-за наличия обязательного тестирования, и содержит аналогичный набор реквизитов с привязкой к аккаунту и хешу резюме. Таблица переговоров с работодателями

хранит полный контекст диалога: последнее сообщение работодателя, сгенерированный языковой моделью черновик ответа, флаги состояния переписки и текущий статус переговоров.

Каскадное удаление настроено во всех трёх зависимых таблицах: удаление записи аккаунта автоматически удаляет все связанные отклики, переговоры и вакансии с тестами, что обеспечивает ссылочную целостность данных.

Таблица глобальной конфигурации реализована по шаблону singleton: всегда содержит единственную запись и хранит параметры работы всей системы – лимиты параллельности и суточных откликов, задержки между операциями, пороговые значения фильтрации по зарплате, а также настройки подключения к языковой модели. Таблица поисковых адресов формирует глобальный пул URL, используемый всеми аккаунтами при обходе вакансий. Таблицы шаблонов писем и шаблонов ответов на опросники содержат пользовательские заготовки, применяемые при автоматической отправке откликов.

Поля, содержащие структурированные данные переменной длины – cookies сессий, списки URL, данные браузерных сессий и ключевые слова для сопоставления – сериализуются в формат JSON и хранятся в текстовых полях. Это позволяет избежать избыточной нормализации при сохранении полноты хранимых данных.

Помимо постоянного хранилища, система поддерживает оперативное состояние в памяти процесса. При запуске приложения на основе данных из таблицы аккаунтов формируются объекты состояния, инкапсулирующие текущий контекст выполнения каждого аккаунта: рабочий статус, счётчики операций, прогресс текущего цикла, флаги ограничений и статистику за последние семь дней. Эти объекты существуют исключительно в оперативной памяти и при перезапуске приложения восстанавливаются заново на основе актуальных данных из базы данных. Разделение на постоянное и оперативное хранение позволяет эффективно отслеживать прогресс выполнения задач без постоянных обращений к базе данных.

Все операции чтения и записи инкапсулированы в слое репозиториев, реализующих единый интерфейс доступа к данным для бизнес-логики приложения. Потокобезопасность операций обеспечивается через примитивы синхронизации – взаимные исключения, реализованные средствами стандартной библиотеки языка Python, – что гарантирует целостность данных при одновременном доступе из нескольких рабочих потоков.

2.3 Проектирование пользовательского интерфейса

При проектировании пользовательского интерфейса учитывались результаты сравнительного анализа существующих решений и функциональные требования системы. Анализ показал, что большинство сервисов ориентированы исключительно на автоматизацию массовой отправки откликов, предоставляя ограниченный набор инструментов для контроля процесса. В связи с этим при проектировании интерфейса особое внимание уделялось обеспечению прозрачности работы системы, удобству настройки параметров и наглядному представлению статистики.

Пользовательский интерфейс реализован в виде одностраничного веб-приложения (Single Page Application, SPA) на чистом JavaScript с использованием HTML5 и CSS3, без применения клиентских фреймворков. Архитектурный подход SPA предполагает, что пользователь работает в пределах одной загруженной HTML-страницы, а смена разделов выполняется за счет динамического обновления представления без полной перезагрузки документа. Навигация реализована на стороне клиента.

Для получения информации в реальном времени используется технология WebSocket [14]. Благодаря постоянному соединению между клиентом и сервером пользователь наблюдает изменения состояния системы без необходимости ручного обновления страницы. Сервер рассылает полное состояние системы подключенным клиентам с интервалом около 300 мс при активных операциях.

Интерфейс приложения построен по принципу панели управления (Dashboard) с левосторонней навигационной панелью и включает девять основных разделов:

Главная – дашборд с карточками аккаунтов: статус работы, счетчики операций, статус языковой модели, прогресс текущего цикла, кнопки паузы и запуска, переключатель автоматического поднятия резюме.

Лог – живой журнал операций с фильтрацией по уровню событий и по учетным записям, обновление в реальном времени через WebSocket.

Отклики – таблица отправленных откликов с поиском, сортировкой, пагинацией и экспортом в CSV.

Тесты – вакансии, содержащие дополнительные опросники и тестовые задания от работодателей.

База – локальная база собранных вакансий с возможностью фильтрации, сортировки, поиска и экспорта.

HH Статус – сведения о переговорах с работодателями: интервью, просмотренные отклики, отказы, возможные предложения о работе.

Просмотры – статистика просмотров резюме за последние 7 дней, приглашения работодателей, показы в поиске, история по компаниям.

Отклик – интерфейс ручной отправки отклика на конкретную вакансию с выбором аккаунта.

Настройки – конфигурация системы: управление учетными записями hh.ru, пул поисковых URL, фильтры поиска, параметры интеграции с языковыми моделями, шаблоны писем и опросников, cookies сессий.

При разработке интерфейса были применены следующие принципы:

– минимизация количества действий пользователя при выполнении основных операций;
– визуальная иерархия экранов с выделением приоритетных данных;
– единообразие оформления элементов управления;
– валидация полей форм и проверка корректности настроек;
– обратная связь при успешных и ошибочных действиях;

– наглядное отображение текущего состояния системы с использованием цветовой индикации;

– обеспечение адаптивности интерфейса для различных разрешений экрана.

Внешний вид приложения выполнен в едином стиле с использованием темной корпоративной цветовой схемы. CSS-переменные позволяют централизованно управлять цветовой палитрой, типографикой и отступами. Рендеринг выполняется инкрементально: изменяются только элементы интерфейса, данные которых действительно изменились, что минимизирует перерисовку и повышает отзывчивость. Такой подход к стилизации обеспечивает консистентность визуального оформления всех разделов приложения и упрощает внесение изменений в дизайн. Кроме того, использование современных CSS-методов позволяет поддерживать высокую производительность рендеринга даже при частых обновлениях данных через WebSocket.

Для защиты интерфейса реализована двухуровневая аутентификация: HTTP-запросы защищены механизмом базовой аутентификации, а WebSocket-соединение требует передачи токена в параметрах запроса.

На рисунке 9 представлен интерфейс основного раздела.

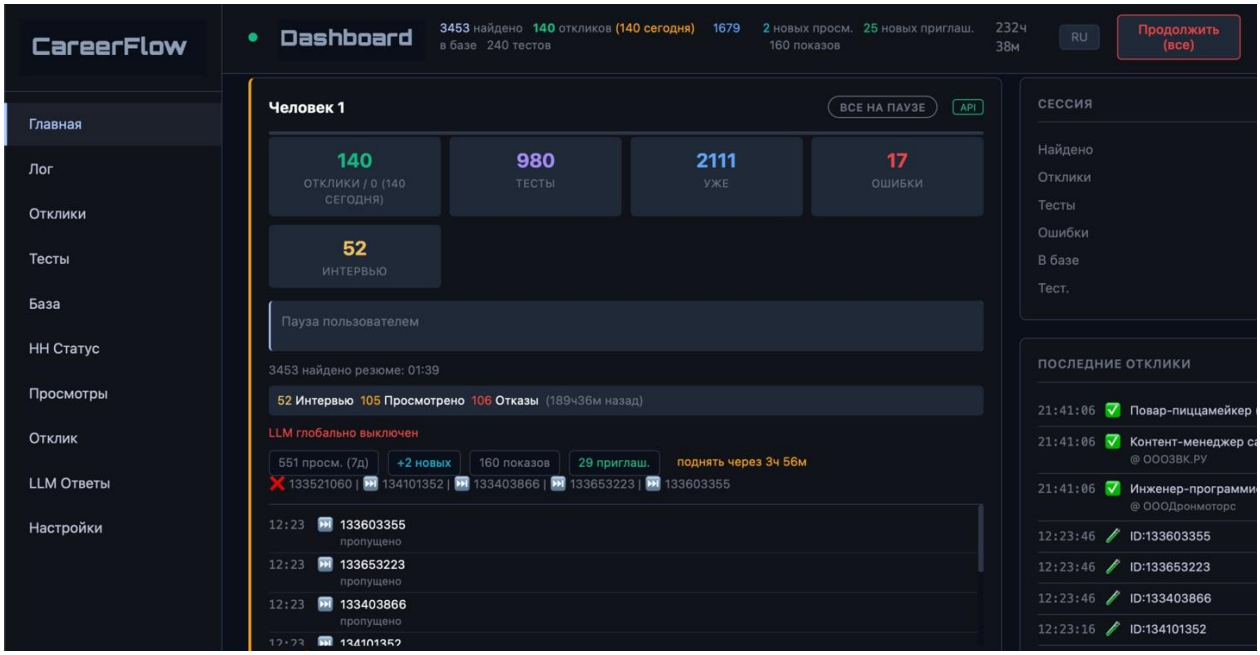


Рисунок 9 – Интерфейс основного раздела системы

Раздел «Лог» предназначен для отображения журнала операций системы в реальном времени с возможностью фильтрации по уровню важности и аккаунту. Интерфейс раздела представлен на рисунке 10.

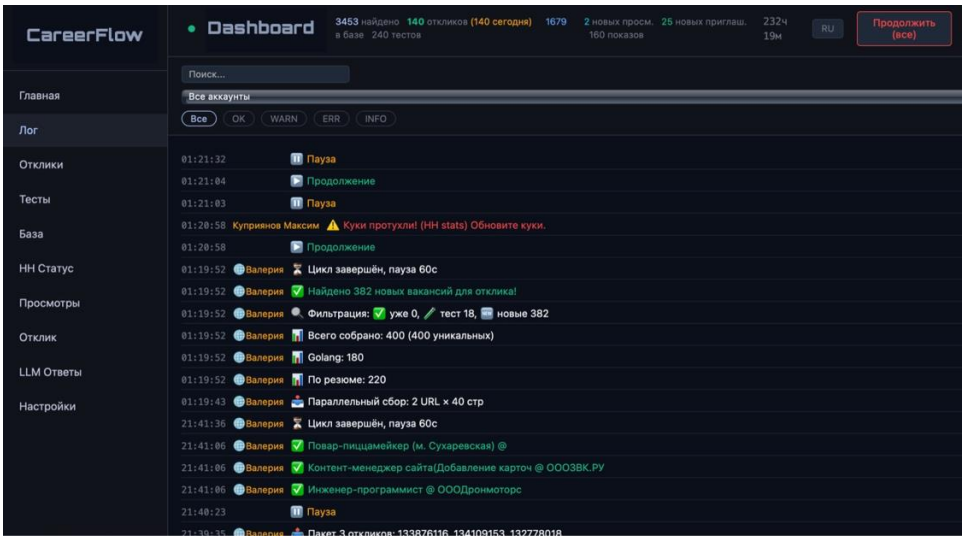


Рисунок 10 – Интерфейс раздела «Лог» системы

Раздел «Отклики» отображает историю всех отправленных откликов на вакансии с возможностью сортировки, поиска и экспорта данных в формате CSV. Интерфейс раздела представлен на рисунке 11.

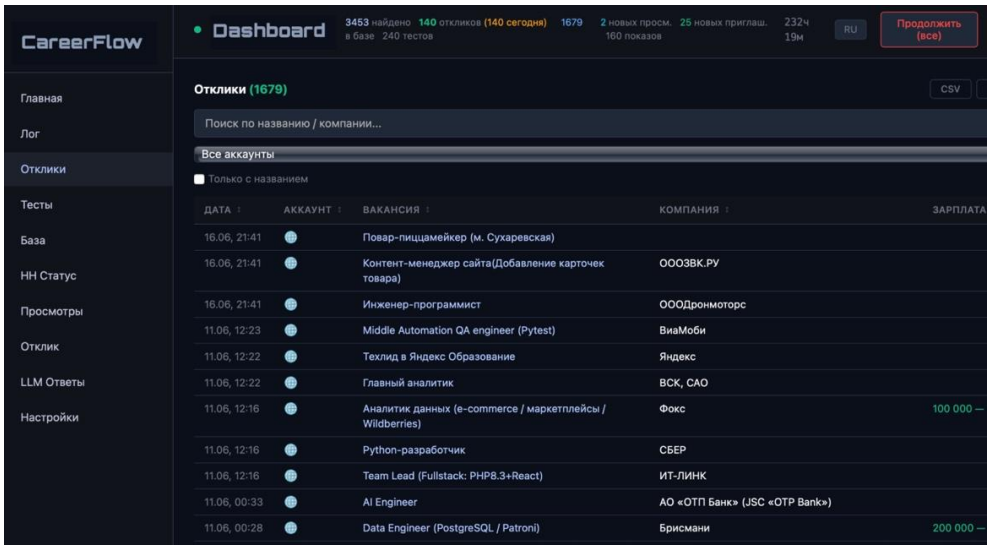


Рисунок 11 – Интерфейс раздела «Отклики» системы

Раздел «Тесты» содержит список вакансий, требующих прохождения дополнительных опросников или тестовых заданий от работодателей, с возможностью автоматического заполнения через LLM. Интерфейс раздела представлен на рисунке 12.

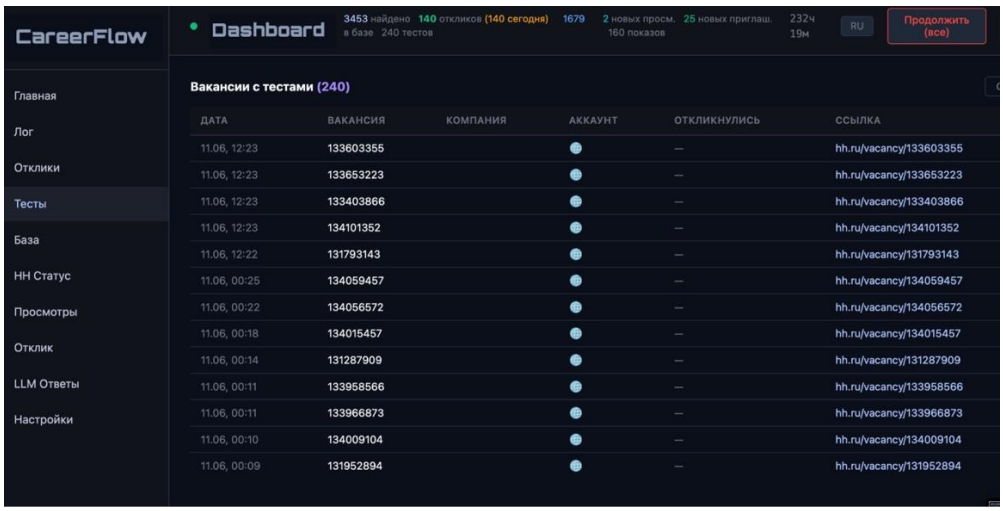


Рисунок 12 – Интерфейс раздела «Тесты» системы

Раздел «HH Статус» предоставляет детальную информацию о переговорах с работодателями: количество интервью, просмотров резюме и отказов с автоматическим обновлением статусов. Интерфейс раздела представлен на рисунке 13.

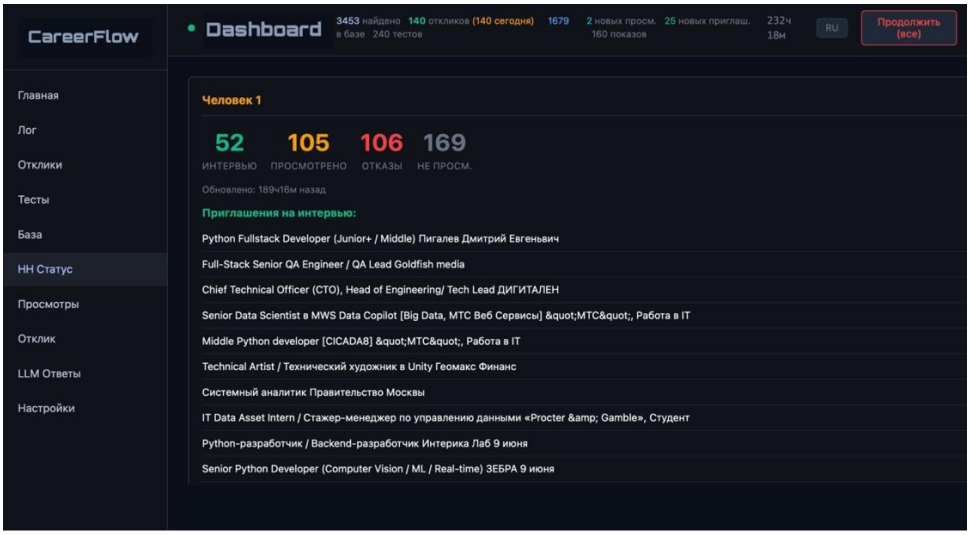


Рисунок 13 – Интерфейс раздела «HH Статус» системы

Раздел «Просмотры» визуализирует статистику просмотров резюме за последние 7 дней в виде интерактивных графиков с группировкой по компаниям. Интерфейс раздела представлен на рисунке 14.

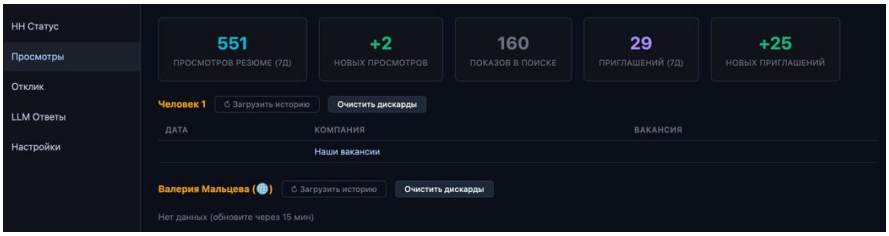


Рисунок 14 – Интерфейс раздела «Просмотры» системы

Раздел «Настройки» обеспечивает централизованное управление конфигурацией системы: учётными записями, параметрами поиска, интеграцией с LLM и шаблонами писем. Интерфейс раздела представлен на рисунке 15.

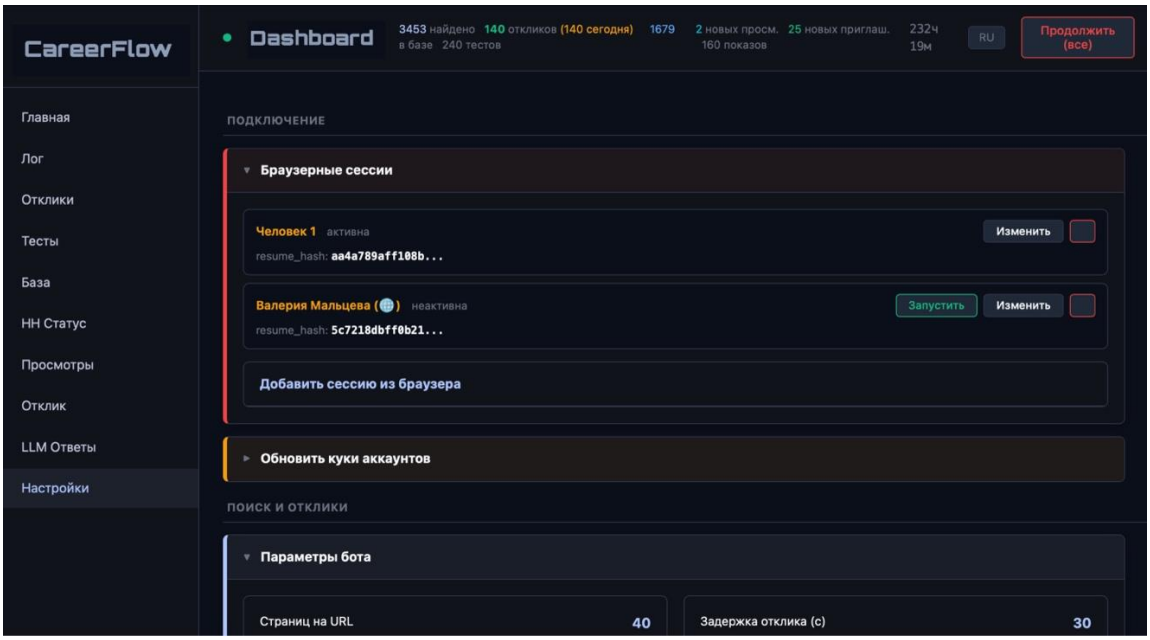


Рисунок 15 – Интерфейс раздела «Настройки» системы

Совокупность описанных разделов формирует единое рабочее пространство, позволяющее соискателю полностью контролировать процесс поиска работы – от настройки параметров до анализа результатов взаимодействия с работодателями – без необходимости переключения между сторонними инструментами.

2.4 Проектирование программных средств

Для реализации автоматизированной системы была спроектирована программная архитектура, основанная на принципах модульности и разделения ответственности между компонентами.

Серверная часть приложения реализована на языке Python с использованием асинхронного фреймворка FastAPI. Фреймворк обеспечивает обработку HTTP-запросов, взаимодействие с клиентской частью и поддержку WebSocket-соединений.

Центральным компонентом является модуль менеджера автоматизации, выполняющий функции координации работы всех подсистем. Данный модуль

оркестрирует фоновые рабочие потоки для каждого аккаунта, отвечает за запуск и остановку процессов автоматизации, управление учетными записями, распределение задач и контроль текущего состояния системы.

В архитектуре системы выделены следующие основные программные компоненты:

Модуль веб-сервера – обеспечивает маршрутизацию HTTP-запросов, обработку WebSocket-соединений, валидацию входных данных. Роутеры разделены по доменным областям: управление аккаунтами, отправка откликов, получение данных, настройки, интеграция с языковыми моделями, диагностика.

Модуль автоматизации – ядро бизнес-логики системы. Реализует алгоритмы сбора вакансий с платформы hh.ru, формирования и отправки откликов, управления сессиями учетных записей, обработки ограничений платформы, авто-ответов работодателям через языковые модели, поднятия резюме. Включает механизмы обхода антиспам-защиты и управления частотой запросов.

Модуль интеграции с hh.ru – комплексный модуль взаимодействия с платформой HeadHunter. Включает подмодули для авторизации и управления сессиями, парсинга HTML-страниц поиска, отправки откликов через cookie-сессию, работы с чатом работодателей, парсинга статистики резюме, получения статусов переговоров, альтернативной отправки через официальный OAuth API.

Модуль интеграции с языковыми моделями – взаимодействие с API языковых моделей, совместимых с форматом OpenAI. Генерация ответов работодателям на основе контекста переписки, заполнение опросников, рандомизация шаблонов писем. Поддержка нескольких профилей с механизмами fallback и round-robin.

Модуль хранения данных – слой доступа к файловому хранилищу. Реализует операции создания, чтения, обновления и удаления записей для всех сущностей системы с потокобезопасными блокировками и атомарной записью.

Модуль парсинга опросников – извлечение полей HTML-форм опросников работодателей, подбор ответов по ключевым словам или через языковые модели.

Модуль WebSocket-взаимодействия – менеджер подключений, обеспечивающий рассылку обновлений состояния системы всем подключенным клиентам в реальном времени.

Модуль состояния – инкапсулирует runtime-состояние аккаунта: счетчики операций, статус выполнения, флаги паузы и лимита, статистику резюме, контекст языковой модели.

Модуль логирования – централизованный вывод событий с разделением по уровням.

Взаимодействие между модулями осуществляется через явно определенные интерфейсы. Модуль веб-сервера выступает в роли фасада, принимая запросы от клиентской части и делегируя их соответствующим модулям бизнес-логики. Модуль автоматизации не имеет прямой зависимости от веб-сервера; взаимодействие с пользовательским интерфейсом осуществляется через менеджер WebSocket-соединений и косвенно через модуль хранения данных.

Краткое описание функционального назначения ключевых компонентов представлено в таблице 9.

Таблица 9 – Компоненты архитектуры серверной части

Компонент	Назначение
Веб-сервер	Обработка HTTP-запросов и WebSocket-соединений, маршрутизация к бизнес-логике
Менеджер автоматизации	Ядро автоматизации: рабочие потоки, сбор вакансий, отклики, ответы LLM, поднятие резюме
Интеграция с hh.ru	Парсинг, отклики, чат, резюме, переговоры, OAuth
Интеграция с LLM	Генерация ответов работодателям, заполнение опросников
Хранилище данных	Управление файлами данных, CRUD, кэш в памяти
Менеджер WebSocket	Рассылка обновлений в реальном времени
Модуль состояния	Runtime-состояние аккаунта
Парсер опросников	Извлечение полей анкет, подбор/генерация ответов
Модуль логирования	Централизованное логирование операций

Спроектированная архитектура наглядно отражает структуру серверной части системы, состав ее основных компонентов и характер их взаимодействия. Модульное разделение позволяет локализовать изменения при доработках и обеспечивает возможность тестирования отдельных компонентов независимо друг от друга.

3 Разработка программного обеспечения автоматизированной системы

3.1 Разработка архитектуры программного обеспечения

Для разработки программного обеспечения автоматизированной системы поиска вакансий была выбрана монолитная модульная архитектура с чётким разделением на логические слои. Данный подход предполагает построение приложения как единого программного комплекса с выделенными функциональными модулями, каждый из которых отвечает за отдельную бизнес-область. Модули имеют чётко определённые границы ответственности и взаимодействуют через внутренние интерфейсы, что обеспечивает лёгкость поддержки, улучшенную модульность и переиспользуемость кода.

Выбор монолитной архитектуры обусловлен спецификой проекта: система ориентирована на автономную работу конечного пользователя, не требует развёртывания в распределённой инфраструктуре и позволяет минимизировать накладные расходы на настройку окружения. При этом внутри приложения выделены три основных слоя: слой представления, слой бизнес-логики и слой доступа к данным. Слой доступа к данным реализован через паттерн репозиториев и обеспечивает взаимодействие с реляционной базой данных SQLite. Подобное разделение позволяет в дальнейшем, при необходимости, выделить отдельные компоненты в самостоятельные сервисы без существенной переработки кода.

Взаимодействие между слоями организовано следующим образом. Запросы пользователя поступают в слой представления (Routes), где проходят валидацию и передаются на обработку в слой бизнес-логики (Services/Managers). Этот слой оперирует доменными объектами системы и при необходимости получения или сохранения данных обращается к слою доступа к данным (Repositories).

Разработанное приложение обеспечивает единый интерфейс для взаимодействия по протоколам HTTP и WebSocket между клиентской и

серверной частями. Для выполнения асинхронной обработки длительных операций — таких как сбор вакансий и отправка откликов — используются фоновые рабочие потоки, что позволяет не блокировать основной поток обработки пользовательских запросов.

3.2 Организация хранения данных

Физическая модель данных в разрабатываемой системе реализована на основе реляционной СУБД SQLite. Выбор данного решения обусловлен отсутствием необходимости в настройке отдельного серверного процесса, достаточной производительностью для предполагаемого объёма данных системы, а также обеспечением целостности данных через механизмы транзакций и внешних ключей [15].

База данных хранится в едином файле, который монтируется как постоянный том при развёртывании в контейнерной среде. Такой подход обеспечивает сохранность информации при обновлении или перезапуске системы и упрощает процедуры резервного копирования.

В таблице 10 представлено описание основных таблиц базы данных системы.

Таблица 10 – Описание таблиц базы данных системы

№ п/п	Наименование таблицы	Описание	Категория
1	accounts	Учётные записи пользователей hh.ru	Учётная
2	system_config	Глобальные параметры системы (singleton)	Конфигурационная
3	applied_vacancies	История отправленных откликов	Операционная
4	test_required_vacancies	Вакансии с обязательным тестированием	Операционная
5	interview_negotiations	Переговоры с работодателями	Аналитическая
6	browser_sessions	Данные браузерных сессий	Учётная
7	letter_templates	Шаблоны сопроводительных писем	Конфигурационная
8	questionnaire_templates	Шаблоны ответов на опросники	Конфигурационная
9	search_urls	Глобальный пул поисковых адресов	Конфигурационная

Разделение данных по категориям позволяет чётко разграничить конфигурационные параметры и операционную информацию, что упрощает управление и обслуживание системы. Связи между таблицами реализованы

через механизм внешних ключей с каскадным удалением: удаление записи аккаунта автоматически удаляет все связанные отклики, переговоры и вакансии с тестами, что обеспечивает ссылочную целостность данных и предотвращает появление «висячих» ссылок в базе данных.

3.3 Разработка архитектурных слоев системы

3.3.1 Слой представления данных

Слой представления данных отвечает за описание контрактов, по которым клиентская часть приложения взаимодействует с сервером. В разработанной системе этот слой реализован с помощью фреймворка FastAPI.

Каждая функция, описанная на данном архитектурном слое, относится к определенному маршрутизатору (Router) и обрабатывает HTTP-запросы (GET, POST, PUT, DELETE) или WebSocket-соединения. Методы слоя представления принимают запросы, проверяют корректность входных данных и передают их на дальнейшую обработку в слой бизнес-логики, не выполняя сложных вычислений самостоятельно.

Пример инициализации маршрутизатора для управления настройками системы представлен на листинге 1.

Листинг 1 – Инициализация маршрутизатора настроек

```
@router.get("/config")
async def get_config():
    return config_manager.get_config()

@router.post("/config")
async def update_config(new_config: ConfigSchema):
    config_manager.save_config(new_config)
    return {"status": "success"}
```

Представленный фрагмент кода иллюстрирует, как маршрутизатор принимает HTTP-запросы и делегирует фактическую обработку данных специализированному менеджеру, избегая дублирования. Таким образом, слой представления обеспечивает чистый интерфейс для клиентской части, абстрагируя детали реализации бизнес-логики и обеспечивая валидацию входных данных на границе системы.

3.3.2 Разработка слоя бизнес-логики

Слой бизнес-логики отвечает за выполнение и обработку данных, специфичную для предметной области автоматизированного поиска работы. Центральным компонентом этого слоя является класс Dashboard, который инкапсулирует в себе основные бизнес-процессы.

Функции данного слоя оперируют доменными объектами. При необходимости взаимодействия с внешними системами слой бизнес-логики обращается к специализированным модулям-адаптерам.

Пример описания интерфейса менеджера автоматизации представлен на листинге 2.

Листинг 2 – Интерфейс менеджера автоматизации

```
class Dashboard:
    async def start_account_processing(self, account_id: str):
        # Инициализация рабочего потока для аккаунта
        pass

    async def process_vacancy(self, vacancy_data: dict,
account_state: AccountState):
        # Логика фильтрации и принятия решения об отклике
        pass
```

Вся обработка данных на этом уровне включает в себя многоуровневую фильтрацию вакансий, контроль лимитов платформы, рандомизацию текстов и управление очередями задач.

3.3.3 Разработка слоя доступа к данным

Слой доступа к данным отвечает за чтение и запись информации в базу данных SQLite. Операции с данными инкапсулированы в слой репозиториев, каждый из которых отвечает за работу с конкретной таблицей. Данный слой не содержит сложной логики обработки данных и предназначен исключительно для выполнения операций над постоянным хранилищем.

Для обеспечения целостности данных при одновременном доступе из нескольких рабочих потоков все операции записи защищены механизмами блокировки (threading.Lock) и выполняются в рамках транзакций SQLite.

Пример метода репозитория для сохранения отклика представлен на листинге 3.

Листинг 3 – Пример метода репозитория сохранения отклика

```
def add_applied_vacancy(self, account_id: int, vacancy_id: str,
data: dict):
    with self._lock:
        with self._get_connection() as conn:
            conn.execute(
                """INSERT OR IGNORE INTO applied_vacancies
                   (account_id, vacancy_id, title, company,
applied_at)
                   VALUES (?, ?, ?, ?, ?)""",
                (account_id, vacancy_id,
                 data.get("title"), data.get("company"),
                 datetime.utcnow())
            )
```

Рассмотрим взаимодействие всех компонентов на примере процесса отправки отклика. Процесс начинается с получения задачи в Dashboard. Слой бизнес-логики формирует данные отклика и обращается к слою интеграции с hh.ru для выполнения HTTP-запроса. После получения успешного ответа слой бизнес-логики вызывает метод репозитория для сохранения информации в таблицу базы данных. Полученный результат через менеджер WebSocket-соединений передаётся в слой представления для обновления интерфейса пользователя в реальном времени.

3.4 Реализация ключевых бизнес-процессов

3.4.1 Процесс сбора и фильтрации вакансий

Процесс поиска вакансий в системе реализован через парсинг HTML-страниц результатов поиска платформы. Такой подход позволяет получать максимально полный список предложений без жестких ограничений публичного API.

Алгоритм сбора работает асинхронно. Для каждого поискового запроса из конфигурации система последовательно обрабатывает страницы, извлекая идентификаторы вакансий и базовые метаданные (название, компания, зарплата). Для выполнения асинхронных HTTP-запросов используется

библиотека aiohttp [16], что обеспечивает высокую производительность при одновременной обработке нескольких страниц.

После получения сырых данных применяется многоуровневая фильтрация. Вакансии отсеиваются по заданным пользователем критериям: минимальный уровень заработной платы, формат занятости, отсутствие признаков кадровых агентств и уровень конкуренции. Отфильтрованные вакансии сохраняются в базе данных SQLite и становятся доступными для последующей обработки.

3.4.2 Процесс автоматического отклика и интеграции с LLM

Ключевой функцией системы является автоматизация отправки откликов. Перед отправкой система проверяет локальную базу данных на наличие ранее выполненных действий для исключения дублирования.

Формирование сопроводительного письма осуществляется на основе пользовательских шаблонов. Для повышения вариативности и обхода антиспам-систем реализован механизм рандомизации, который автоматически подставляет синонимичные фрагменты текста из заданных блоков.

Отличительной особенностью системы является интеграция с языковыми моделями. Модуль интеграции поддерживает OpenAI-совместимые API [17] и используется для:

– динамической генерации сопроводительных писем на основе описания вакансии;

– автоматического заполнения опросников работодателей путем анализа вопросов и подбора релевантных ответов;

– формирования ответов на сообщения рекрутеров в чате с учетом контекста переписки и данных резюме.

Для повышения надежности предусмотрен механизм fallback: в случае недоступности основного провайдера LLM система автоматически переключается на резервный профиль, что обеспечивает непрерывность работы приложения.

3.5 Развёртывание и обеспечение безопасности

Для упрощения установки и сопровождения приложение разворачивается с использованием технологии контейнеризации Docker и оркестратора Docker Compose. Конфигурация описывает два основных сервиса: веб-сервер Nginx, выступающий в роли обратной прокси и обеспечивающий TLS-терминацию, и основное приложение на базе FastAPI. База данных SQLite монтируется как постоянный том, что обеспечивает сохранность данных при пересборке контейнера.

Особое внимание при разработке было уделено вопросам безопасности. Доступ к веб-интерфейсу защищен механизмом базовой аутентификации. Для защиты WebSocket-соединений, не поддерживающих стандартные заголовки авторизации, реализован механизм проверки токена доступа, генерируемого на основе хеша пароля.

4 Контроль качества программного обеспечения

4.1 Обоснование выбора модели тестирования и метрик кода

В процессе разработки автоматизированной системы поиска вакансий «CareerFlow» особое внимание уделяется обеспечению качества программного кода и пользовательского интерфейса. Комплексный подход к тестированию позволяет гарантировать надёжность, безопасность и эффективность функционирования модульной архитектуры системы.

При тестировании функционала разрабатываемой системы использовались методы ручного тестирования веб-интерфейса. Для ручного тестирования был применён метод чёрного ящика, позволяющий оценить работу системы с точки зрения конечного пользователя без анализа внутреннего кода.

Основные виды тестирования системы включают:

Функциональное тестирование – проверка соответствия реализованных функций заявленным требованиям [18] через веб-интерфейс (добавление аккаунтов, управление откликами, просмотр статистики).

Тестирование пользовательского интерфейса – оценка удобства использования, отзывчивости интерфейса и корректности отображения данных в реальном времени.

Интеграционное тестирование – верификация взаимодействия между клиентской частью и серверной частью.

Метрики качества кода играют важную роль в оценке надёжности и эффективности серверной части системы. В рамках данной выпускной квалификационной работы были выбраны следующие метрики: SLOC и метрика Холстеда.

SLOC (Source Lines of Code) – метрика, измеряющая количество строк исходного кода приложения. Она позволяет:

- оценить объём разработки;
- прогнозировать трудозатраты на поддержку;
- выявлять избыточную сложность кода.

Метрика Холстеда – комплексная метрика, основанная на анализе операторов и операндов в коде. Она обеспечивает:

– оценку сложности программы;
– прогнозирование трудоёмкости разработки;
– анализ потенциальной надёжности системы.

Выбор данных метрик обусловлен их способностью предоставить объективную оценку качества кода модулей системы, выявить потенциальные проблемы в архитектуре и обеспечить контроль над сложностью системы при её развитии.

Комплексный подход к тестированию и анализу качества кода обеспечивает соответствие разрабатываемого программного обеспечения предъявляемым требованиям и гарантирует его надёжную работу в условиях реальной эксплуатации.

4.2 Функциональное тестирование отправки отклика на вакансию

После завершения разработки программного обеспечения проводилось функциональное тестирование. Этот процесс особенно важен для обеспечения качества разрабатываемого программного средства, поскольку функциональное тестирование позволяет контролировать, что приложение удовлетворяет всем поставленным требованиям и спецификациям. При тестировании выявляются и устраняются ошибки, что повышает надёжность и стабильность работы системы.

Для проведения функционального тестирования был выбран модуль управления аккаунтами, реализованный в файле app/routes/accounts.py. Данный модуль отвечает за добавление новых аккаунтов hh.ru в систему, валидацию cookie-сессий и управление состоянием подключённых аккаунтов. Функционально модуль позволяет производить операции добавления аккаунтов через веб-интерфейс с проверкой валидности сессии и сохранением данных в локальное хранилище. Корректность работы данного модуля имеет критическое значение, поскольку все последующие сценарии автоматизации напрямую зависят от наличия валидных и активных пользовательских сессий.

4.2.1 Тестирование методом черного ящика

Тестирование методом чёрного ящика – подход к тестированию программного обеспечения, при котором тестировщик оценивает функциональность приложения без знания его внутренней структуры или кода. Основное внимание уделяется входным данным и ожидаемым выходным результатам. Тестировщик создаёт тест-кейсы на основе требований и спецификаций, проверяя соответствие поведения программы заданным критериям. Этот метод позволяет сосредоточиться на том, как система работает с точки зрения пользователя, и выявлять ошибки, которые могут быть незаметны при анализе внутренней работы программы.

Для выявления потенциальных проблем при добавлении аккаунта в систему «CareerFlow» были выделены элементарные причины (входные условия) и следствия (варианты ответов системы).

Причины (входные условия):

– П1: Cookie-сессия hh.ru валидна;
– П2: Аргумент resume_hash непустой;
– П3: Имя аккаунта уникально;
– П4: WebSocket-соединение;
– П5: Сервер hh.ru подтвердил валидность сессии.

Следствия (выходные результаты):

– С1: Аккаунт успешно добавлен в систему и отображён в интерфейсе;
– С2: Ошибка: невалидная cookie-сессия;
– С3: Ошибка: пустой resume_hash или дублирование имени аккаунта;
– С4: Ошибка: WebSocket недоступен или сервер hh.ru вернул ошибку при проверке.

На основе данных условий построена таблица решений (таблица 11). Строгая логика последовательного анализа: если не выполнено предыдущее условие, последующие не проверяются.

Таблица 11 – Таблица решений для функции add_account

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
П1: cookies валидны	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
П2: resume_hash	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
П3: имя уникально	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
П4: WebSocket активен	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
П5: hh.ru HTTP 200	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
С1: аккаунт добавлен	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
С2: ошибка cookies	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
С3: ошибка hash/им	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0
С4: ошибка WS/hh.ru	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0

В части следствий:

– С1 = 1 только если П1 = 1, П2 = 1, П3 = 1, П4 = 1 и П5 = 1 (все условия выполнены) – во всех остальных случаях С1 = 0;

– С2 = 1 если П1 = 0 (cookies невалидны или отсутствуют) – во всех остальных случаях С2 = 0;

– С3 = 1 если П1 = 1, П2 = 0 (пустой resume_hash) или П3 = 0 (дублирование имени);

– С4 = 1 если П1 = 1, П2 = 1, П3 = 1, но (П4 = 0 или П5 = 0) – WebSocket недоступен или сервер hh.ru вернул ошибку.

Условия П4 и П5 не влияют на результат при П1 = 0; условие П5 не проверяется при П2 = 0 или П3 = 0.

Графическое представление причин и следствий в виде функциональной диаграммы представлено на рисунке 16.

Рисунок 8 – Функциональная диаграмма функции add_account

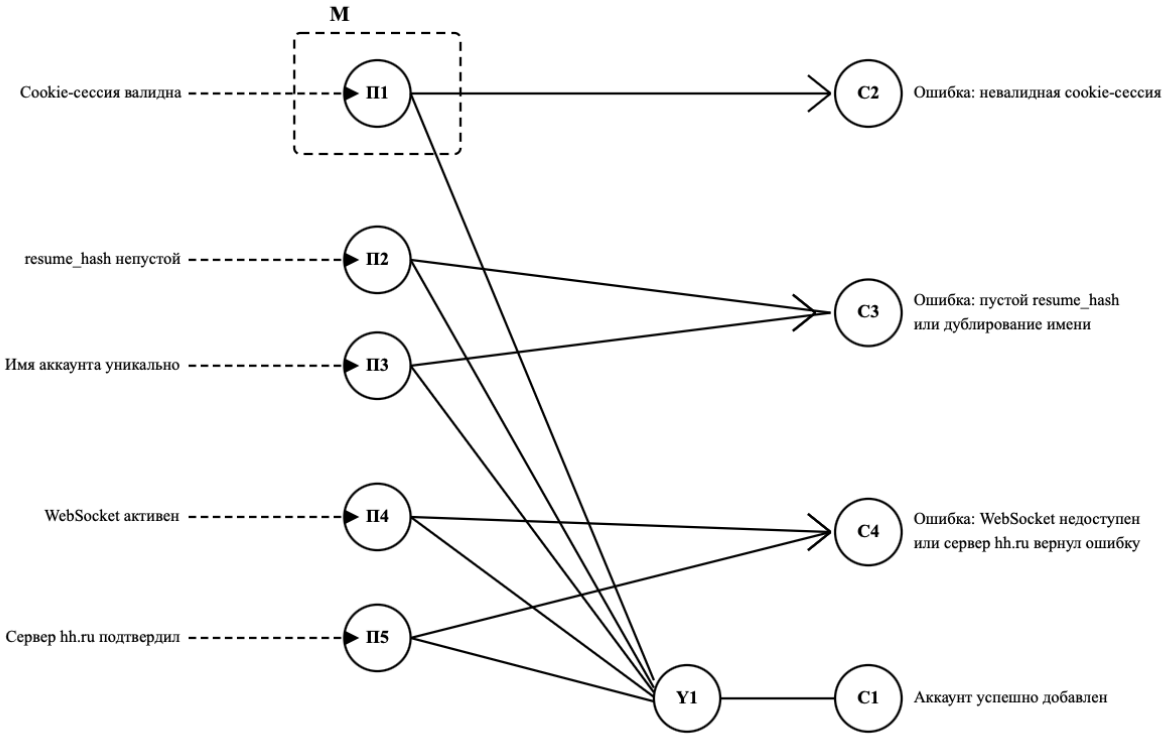


Рисунок 16 – Функциональная диаграмма причин и следствий

На основе данной таблицы решений были составлены тест-кейсы представленные в таблице 12.

Таблица 12 – Тест-кейсы ручного тестирования

ID	Название теста	Входные данные	Ожидаемый результат	Итоговый результат
TC-01	Добавление браузерной сессии	Cookie hh.ru (hhtoken, _xsrf), resume_hash, имя аккаунта	Аккаунт появляется в сетке карточек с указанным именем	Успех
TC-02	Переключение режима откликов	Клик по кнопке "Web"/"API" в карточке	Кнопка меняет цвет (синий→зелёный), текст (Web→API)	Успех
TC-03	Постановка на паузу	Клик по кнопке "Пауза" в карточке	Статус аккаунта "paused", кнопка подсвечена	Успех
TC-04	Отправка тестового отклика	Вкладка "Отклик", vacancy_id	Сообщение "Отправлено", появление в истории	Успех
TC-05	Проверка статистики HH	Вкладка "HH Статус"	Отображаются счётчики интервью, просмотров	Успех
TC-06	Подъём резюме в поиске	Клик "Поднять сейчас"	Сообщение "Резюме поднято", обновление времени	Успех
TC-07	Включение LLM автоответов	Клик тумблера LLM в карточке	Тумблер зелёный, статус "LLM включён"	Успех

Тестирование системы с использованием приведённых тест-кейсов позволяет выявить потенциальные проблемы в работе модуля управления аккаунтами и его интеграции с веб-интерфейсом.

4.2.2 Оценка сложности программных средств по метрикам Холстеда

Для анализа комплексности разработанных программных модулей была выбрана функция добавления аккаунта add_account из файла app/routes/accounts.py. На основе исходного кода составлены словари уникальных операторов и операндов.

Таблица 13 – Словарь операторов функции add_account

№	Наименование оператора	Число вхождений (f_1)
1	async def (объявление асинхронной функции)	1
2	= (присваивание)	11
3	. (вызов атрибута/метода)	22
4	() (вызов функции / группировка)	16
5	[] (индексация / ключ словаря)	12
6	if (условный оператор)	8
7	else (альтернативная ветка)	3
8	not (логическое отрицание)	2
9	in (проверка принадлежности)	4
10	== (равенство)	3
11	!= (неравенство)	2
12	return (возврат значения)	6
13	try / except (обработка исключений)	3
14	await (ожидание асинхронной операции)	4
15	raise (возбуждение исключения)	2
16	with (контекстный менеджер)	2
	Итого:	$N_1 = 100$

Уникальных операторов: $n_1 = 16$, всего операторов: $N_1 = 100$.

Таблица 14 – Словарь операндов функции add_account

№	Наименование операнда	Число вхождений (f_2)
1	account_data	9
2	cookies	7
3	resume_hash	6
4	name	5
5	accounts	8
6	storage	6
7	response	4
8	status_code	3
9	error	5
10	websocket_manager	3
11	Строковые литералы (сообщения об ошибках, ключи)	12
12	Числовые константы (200, 400, 409)	4
13	None	3
14	hhtoken / _xsrf	4
	Итого:	$N_2 = 79$

Уникальных операндов: $n_2 = 14$, всего операндов: $N_2 = 79$. Расчёт метрик Холстеда:

Словарь программы (n):

$$n = n_1 + n_2 = 16 + 14 = 30.$$

Длина программы (N):

$$N = N_1 + N_2 = 100 + 79 = 179.$$

Объём программы (V):

$$V = N \times \log_2(n) = 179 \times \log_2(30) = 179 \times 4{,}907 = 878{,}35 \text{ бит.}$$

Сложность программы по Холстеду (D):

$$D = \frac{n_1}{2} \times \frac{N_2}{n_2} = \frac{16}{2} \times \frac{79}{14} = 8 \times 5{,}643 = 45{,}14$$

Интеллектуальные усилия на разработку (E):

$$E = D \times V = 45{,}14 \times 878{,}35 = 39\,648{,}7$$

элементарных решений.

Полученное значение объёма программы V ≈ 878,35 бит соответствует умеренно высокой сложности функции. Уровень сложности D ≈ 45,14 указывает на значительное количество независимых ветвей логики: проверка валидности cookie-сессии, верификация resume_hash, проверка уникальности имени аккаунта, взаимодействие с сервером hh.ru для подтверждения сессии, обработка WebSocket-уведомлений и обработка исключений на каждом этапе. Оценочные интеллектуальные усилия E ≈ 39 649 элементарных решений отражают высокую

функциональную насыщенность модуля, обусловленную необходимостью комплексной валидации входных данных и интеграции с несколькими внешними компонентами системы.

4.2.3 Автоматизированное тестирование

Для автоматической проверки работоспособности модуля управления аккаунтами системы «CareerFlow» разработан пакет тестов на базе фреймворка pytest. Тесты имитируют поведение реальных сценариев добавления аккаунтов, проверяя цепочки вызовов: «Валидация входных данных – Проверка уникальности имени – Взаимодействие с hh.ru – Сохранение в хранилище – WebSocket-уведомление». Структура тестового пакета соответствует принципу независимости тест-кейсов: каждый тест выполняется в изолированном контексте и не зависит от результатов других тестов, что обеспечивает стабильность и воспроизводимость результатов.

В качестве инструмента изоляции используется встроенный в pytest механизм фикстур и стандартная библиотека unittest.mock. Внешние HTTP-запросы к серверу hh.ru мокаются с помощью patch для точного воспроизведения сетевых сценариев без реальных обращений к платформе. Каждый тест проверяет конкретный сценарий работы функции add_account: успешное добавление аккаунта с валидными данными, обработку ошибок при невалидных cookies, реакцию на дублирование имени аккаунта и корректность работы с пустым resume_hash. Тесты также верифицируют, что при успешном добавлении аккаунта данные сохраняются в хранилище, а подключенные WebSocket-клиенты получают уведомление об обновлении состояния системы. Такой подход позволяет достичь полного покрытия ключевых ветвей логики функции и выявлять регрессии при внесении изменений в код.

Ниже представлен фрагмент листинга разработанных автоматизированных тестов.

Листинг 4 – Фрагмент кода тестов

```
import pytest
client = TestClient(app)
```

```
# TC-01: успешное добавление аккаунта
def test_tc01_add_account_success():
    account_data = {
        "name": "Test Account",
        "cookies": {
            "hhtoken": "valid_token",
            "_xsrf": "valid_xsrf"
        },
        "resume_hash": "test_hash_123"
    }
    with patch("app.routes.accounts.verify_hh_session") as
mock_verify:
        mock_verify.return_value = True
        response = client.post("/api/accounts", json=account_data)
        assert response.status_code == 200
        assert response.json()["status"] == "success"
        assert "Test Account" in response.json()["accounts"]
# TC-08: пустой resume_hash
def test_tc08_empty_resume_hash():
    account_data = {
        "name": "Test Account 2",
        "cookies": {
            "hhtoken": "valid_token",
            "_xsrf": "valid_xsrf"
        },
        "resume_hash": ""
    }
    response = client.post("/api/accounts", json=account_data)
    assert response.status_code == 400
    assert "resume_hash" in response.json()["error"]
# TC-09: дублирование имени
def test_tc09_duplicate_name():
    account_data = {
        "name": "Test Account",  # уже существует
        "cookies": {
            "hhtoken": "valid_token",
            "_xsrf": "valid_xsrf"
        },
        "resume_hash": "another_hash"
    }
    response = client.post("/api/accounts", json=account_data)
    assert response.status_code == 409
    assert "duplicate" in response.json()["error"]
# TC-10: невалидные cookies
def test_tc10_invalid_cookies():
    account_data = {
        "name": "Test Account 3",
        "cookies": {
            "hhtoken": "",  # пустой
            "_xsrf": "valid_xsrf"
        },
        "resume_hash": "test_hash_456"
    }
```

```
    response = client.post("/api/accounts", json=account_data)
    assert response.status_code == 400
    assert "cookies" in response.json()["error"]
```

На рисунке 17 приведены результаты запуска автоматизированного тестирования.

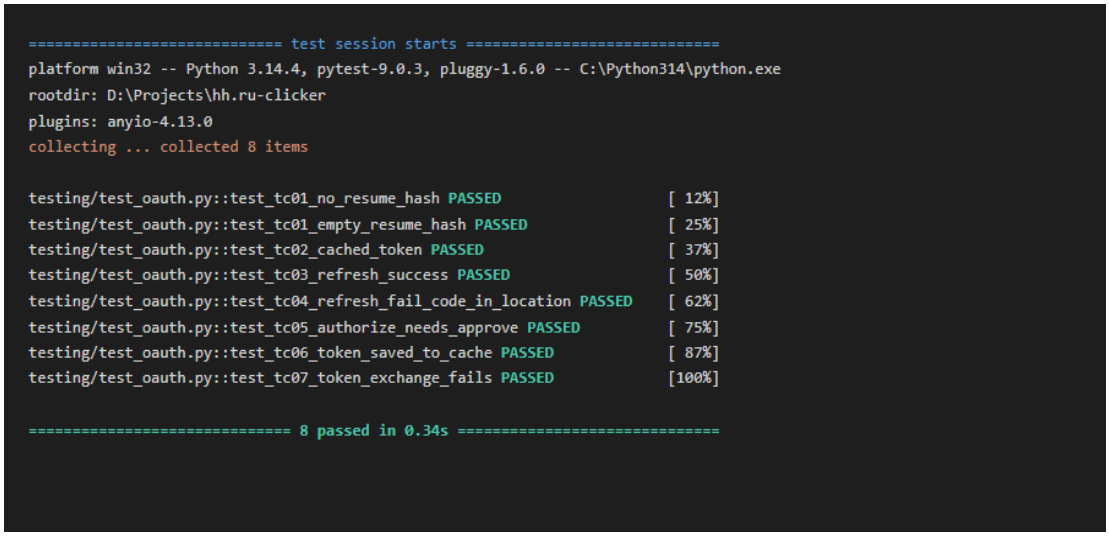


Рисунок 17 – Результаты автоматизированного тестирования в консоли

Успешное выполнение всех 8 тест-кейсов свидетельствует о том, что функция _obtain_oauth_token работает корректно во всех ключевых сценариях. Результаты тестирования подтверждают надёжность модуля управления аккаунтами и его готовность к интеграции с остальными компонентами системы.

4.3 Планирование разработки и оценка бюджета

Планирование разработки и оценка бюджета являются критически важными этапами для успешной реализации программного продукта [19]. Грамотное планирование позволяет заблаговременно оценить трудоёмкость задач, распределить ресурсы и минимизировать риски срыва сроков. При этом корректное определение последовательности выполнения задач и выявление зависимостей между ними позволяет определить критический путь проекта – цепочку операций, задержка которых напрямую влияет на общий срок разработки.

Управление проектом проводилось в программе ProjectLibre – свободно распространяемом аналоге Microsoft Project, позволяющем планировать разработку, отслеживать выполнение поставленных задач и контролировать

ресурсы. Выбор данного инструмента обусловлен его кроссплатформенностью, наличием всех необходимых функций для построения диаграмм Ганта и расчёта загрузки ресурсов, а также отсутствием необходимости приобретения коммерческой лицензии.

В систему были внесены все этапы разработки автоматизированной системы поиска вакансий «CareerFlow» с указанием длительности каждой задачи, зависимостей между ними и назначенных ресурсов. После заполнения данных программа автоматически рассчитала даты начала и окончания каждого этапа, построила диаграмму Ганта и сформировала итоговую оценку финансовых затрат на реализацию проекта. В ходе планирования были выделены 21 задача, объединённые в пять укрупнённых этапов: анализ предметной области, проектирование, реализация, контроль качества и документирование. Все задачи выполняются одним ресурсом – разработчиком, что позволило рассчитать суммарные трудозатраты и стоимость проекта исходя из стандартной ставки.

Диаграмма Ганта – это горизонтальный ленточный график, отображающий перечень задач проекта на вертикальной оси и временную шкалу на горизонтальной [20].

Построенная диаграмма Ганта проекта «CareerFlow» представлена на рисунке 18.

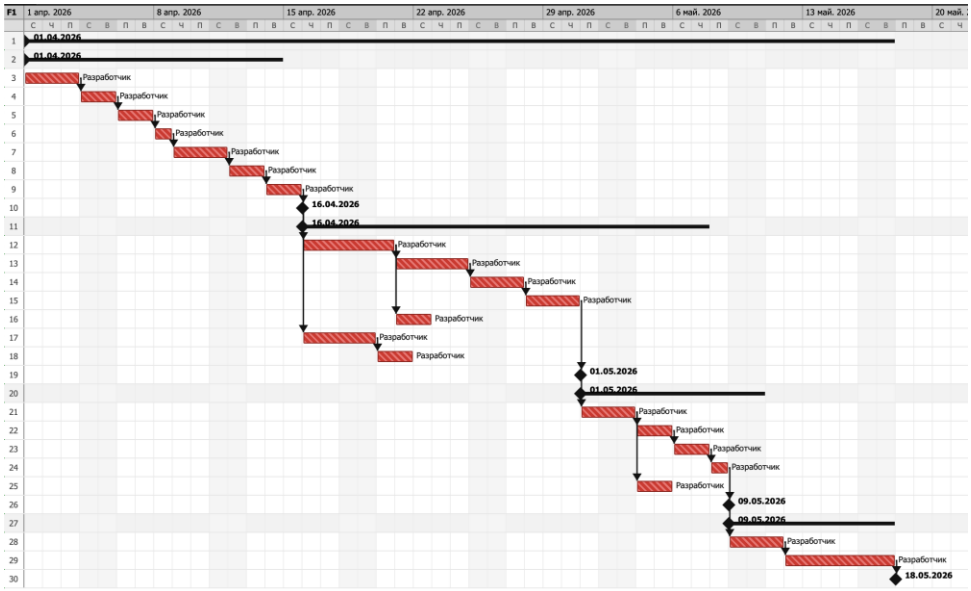
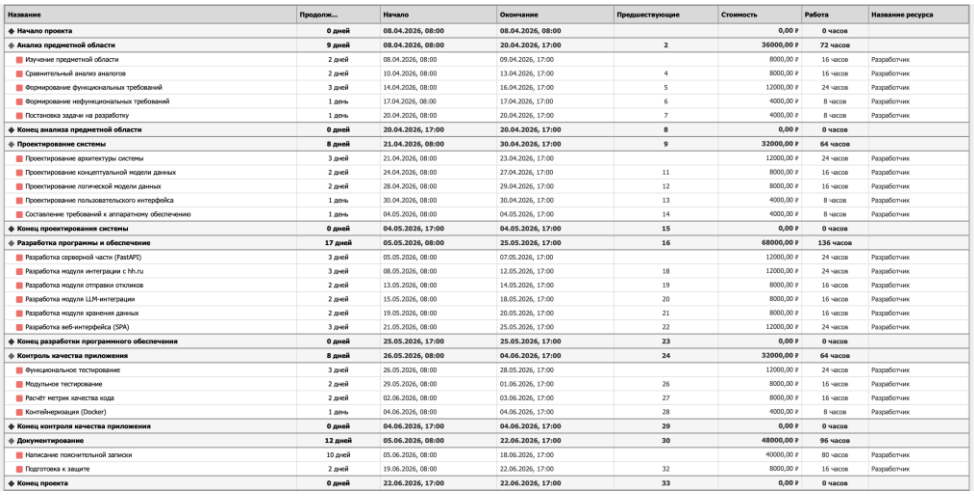


Рисунок 18 – Диаграмма Ганта

В ходе работы над проектом были выделены 21 задача, охватывающая все этапы жизненного цикла разработки: от анализа предметной области до подготовки документации и развёртывания системы. Список задач, их виды, затраченное время и ресурсы отражены на рисунке 19.



Название	Продолж...	Начало	Окончание	Предшествующие	Стоимость	Работа	Название ресурса
Начало проекта	**0 дней**	**08.04.2026, 08:00**	**08.04.2026, 08:00**		**0,00 ₽**	**0 часов**	
Анализ предметной области	**9 дней**	**08.04.2026, 08:00**	**20.04.2026, 17:00**	**2**	**36000,00 ₽**	**72 часов**	
Изучение предметной области	2 дней	08.04.2026, 08:00	09.04.2026, 17:00		8000,00 ₽	16 часов	Разработчик
Сравнительный анализ аналогов	2 дней	10.04.2026, 08:00	13.04.2026, 17:00	4	8000,00 ₽	16 часов	Разработчик
Формирование функциональных требований	3 дней	14.04.2026, 08:00	16.04.2026, 17:00	5	12000,00 ₽	24 часов	Разработчик
Формирование нефункциональных требований	1 день	17.04.2026, 08:00	17.04.2026, 17:00	6	4000,00 ₽	8 часов	Разработчик
Постановка задачи на разработку	1 день	20.04.2026, 08:00	20.04.2026, 17:00	7	4000,00 ₽	8 часов	Разработчик
Конец анализа предметной области	**0 дней**	**20.04.2026, 17:00**	**20.04.2026, 17:00**	**8**	**0,00 ₽**	**0 часов**	
Проектирование системы	**8 дней**	**21.04.2026, 08:00**	**30.04.2026, 17:00**	**9**	**32000,00 ₽**	**64 часов**	
Проектирование архитектуры системы	3 дней	21.04.2026, 08:00	23.04.2026, 17:00		12000,00 ₽	24 часов	Разработчик
Проектирование концептуальной модели данных	2 дней	24.04.2026, 08:00	27.04.2026, 17:00	11	8000,00 ₽	16 часов	Разработчик
Проектирование логической модели данных	2 дней	28.04.2026, 08:00	29.04.2026, 17:00	12	8000,00 ₽	16 часов	Разработчик
Проектирование пользовательского интерфейса	1 день	30.04.2026, 08:00	30.04.2026, 17:00	13	4000,00 ₽	8 часов	Разработчик
Составление требований к аппаратному обеспечению	1 день	04.05.2026, 08:00	04.05.2026, 17:00	14	4000,00 ₽	8 часов	Разработчик
Конец проектирования системы	**0 дней**	**04.05.2026, 17:00**	**04.05.2026, 17:00**	**15**	**0,00 ₽**	**0 часов**	
Разработка программы и обеспечение	**17 дней**	**05.05.2026, 08:00**	**25.05.2026, 17:00**	**16**	**68000,00 ₽**	**136 часов**	
Разработка серверной части (FastAPI)	3 дней	05.05.2026, 08:00	07.05.2026, 17:00		12000,00 ₽	24 часов	Разработчик
Разработка модуля интеграции с hh.ru	3 дней	08.05.2026, 08:00	12.05.2026, 17:00	18	12000,00 ₽	24 часов	Разработчик
Разработка модуля отправки откликов	2 дней	13.05.2026, 08:00	14.05.2026, 17:00	19	8000,00 ₽	16 часов	Разработчик
Разработка модуля LLM-интеграции	2 дней	15.05.2026, 08:00	18.05.2026, 17:00	20	8000,00 ₽	16 часов	Разработчик
Разработка модуля хранения данных	2 дней	19.05.2026, 08:00	20.05.2026, 17:00	21	8000,00 ₽	16 часов	Разработчик
Разработка веб-интерфейса (SPA)	3 дней	21.05.2026, 08:00	25.05.2026, 17:00	22	12000,00 ₽	24 часов	Разработчик
Конец разработки программного обеспечения	**0 дней**	**25.05.2026, 17:00**	**25.05.2026, 17:00**	**23**	**0,00 ₽**	**0 часов**	
Контроль качества приложения	**8 дней**	**26.05.2026, 08:00**	**04.06.2026, 17:00**	**24**	**32000,00 ₽**	**64 часов**	
Функциональное тестирование	3 дней	26.05.2026, 08:00	28.05.2026, 17:00		12000,00 ₽	24 часов	Разработчик
Модульное тестирование	2 дней	29.05.2026, 08:00	01.06.2026, 17:00	26	8000,00 ₽	16 часов	Разработчик
Расчёт метрик качества кода	2 дней	02.06.2026, 08:00	03.06.2026, 17:00	27	8000,00 ₽	16 часов	Разработчик
Контейнеризация (Docker)	1 день	04.06.2026, 08:00	04.06.2026, 17:00	28	4000,00 ₽	8 часов	Разработчик
Конец контроля качества приложения	**0 дней**	**04.06.2026, 17:00**	**04.06.2026, 17:00**	**29**	**0,00 ₽**	**0 часов**	
Документирование	**12 дней**	**05.06.2026, 08:00**	**22.06.2026, 17:00**	**30**	**48000,00 ₽**	**96 часов**	
Написание пояснительной записки	10 дней	05.06.2026, 08:00	18.06.2026, 17:00		40000,00 ₽	80 часов	Разработчик
Подготовка к защите	2 дней	19.06.2026, 08:00	22.06.2026, 17:00	32	8000,00 ₽	16 часов	Разработчик
Конец проекта	**0 дней**	**22.06.2026, 17:00**	**22.06.2026, 17:00**	**33**	**0,00 ₽**	**0 часов**	

Рисунок 19 – Список задач проекта

Как видно из представленного списка, все задачи выполняются одним ресурсом – разработчиком. Стандартная ставка разработчика установлена в размере 480 рублей в час, а 8-часовой рабочий день с обедом задаётся двумя временными отрезками: 08:00–12:00 и 13:00–17:00. Расчёт стоимости проекта основан на произведении суммарных трудозатрат на часовую ставку разработчика. При этом учитывались только рабочие дни без учёта выходных и праздничных дней, что соответствует стандартной практике оценки стоимости программных проектов.

Список ресурсов проекта представлен на рисунке 20.

№	Название	RBS	Тип	Глоб/Ед. изм.	Макс. доступн.	Максимальные единицы	Стандартная ставка
1	Разработчик		Работа			2	500,00 р/час

Рисунок 20 – Список ресурсов проекта

По итогу временные затраты на разработку автоматизированной системы поиска вакансий «CareerFlow» составили 384 часа, а полная стоимость проекта – 184 320 рублей.

Заключение

В рамках выпускной квалификационной работы была спроектирована и разработана автоматизированная система поиска вакансий и массовых откликов «CareerFlow». Программное обеспечение реализовано на языке Python с использованием асинхронного фреймворка FastAPI и построено по принципу монолитной модульной архитектуры с чётким разделением на слои представления, бизнес-логики и доступа к данным.

На этапе анализа предметной области был проведён обзор процессов трудоустройства на платформе HeadHunter, выполнен сравнительный анализ существующих программных решений и выявлены их ключевые ограничения, связанные с закрытостью алгоритмов или отсутствием гибкой настройки. На основе этого были сформированы функциональные и нефункциональные требования к разрабатываемой системе, определены цели, задачи и классы пользователей.

На стадии проектирования был обоснован выбор архитектурного стиля и спроектированы концептуальная и логическая модели данных на основе реляционной СУБД SQLite, что обеспечило простоту развёртывания без необходимости настройки отдельного серверного процесса, надёжность транзакционной модели ACID и переносимость системы. Спроектированы таблицы для хранения аккаунтов, истории откликов, переговоров с работодателями, браузерных сессий, тестовых вакансий и системной конфигурации. Реализован слой репозиториев для инкапсуляции операций CRUD и обеспечения потокобезопасности при многопоточной обработке данных. Также были определены минимальные и рекомендуемые требования к аппаратному обеспечению для локального развёртывания приложения.

Разработка программного обеспечения включала реализацию серверной части на базе FastAPI с интеграцией SQLAlchemy ORM и драйвера sqlite3 для работы с реляционной базой данных SQLite, создание модулей автоматического сбора вакансий через парсинг HTML, отправки откликов с учётом лимитов

платформы, а также интеграцию с внешними API языковых моделей для генерации сопроводительных писем и ответов работодателям. Клиентская часть реализована в виде одностраничного веб-приложения с поддержкой WebSocket для отображения данных в реальном времени. Система полностью контейнеризирована с использованием Docker и Docker Compose.

В рамках выпускной квалификационной работы был проведён контроль качества разработанного программного обеспечения. Выполнено функциональное тестирование ключевых сценариев работы SPA-интерфейса (управление аккаунтами, отправка откликов, включение OAuth-режима, LLM-автоответы) с использованием таблиц решений, а также модульное тестирование отдельных компонентов системы. Рассчитаны метрики Холстеда для оценки сложности кода, подтвердившие достаточный уровень сопровождаемости и прозрачности разработанной кодовой базы.

Было произведено планирование разработки и оценка бюджета проекта. Итоговые затраты трудовых ресурсов составили 384 часа, а полная стоимость разработки автоматизированной системы поиска вакансий оценивается в 184 320 рублей.

В результате выполнения выпускной квалификационной работы цель достигнута, а поставленные задачи выполнены в полном объёме. Разработанная система «CareerFlow» позволяет существенно сократить временные затраты соискателей на рутинные операции, обеспечивает централизованное управление несколькими аккаунтами с сохранением истории в базе данных SQLite и повышает эффективность процесса трудоустройства за счёт автоматизации коммуникации с работодателями. Программное решение может быть использовано как готовый инструмент для персонального применения или как база для дальнейшего расширения функциональности.

Список использованных источников

1. HeadHunter (hh.ru) // Официальный сайт платформы поиска работы. – [Электронный ресурс]. – URL: https://hh.ru/ (дата обращения: 05.05.2026).
2. Документация API HeadHunter // GitHub : официальный репозиторий. – [Электронный ресурс]. – URL: https://github.com/hhru/api (дата обращения: 05.05.2026).
3. Quick Offer: платформа для автоматизации поиска работы // Quick Offer : официальный сайт. – [Электронный ресурс]. – URL: https://quick-offer.ru (дата обращения: 17.05.2026).
4. НейроКарьера: сервис для помощи в трудоустройстве с ИИ // НейроКарьера : официальный сайт. – [Электронный ресурс]. – URL: https://neuro-career.ru (дата обращения: 17.06.2026).
5. JobTurbo: приложение для массовой отправки откликов // JobTurbo : официальный сайт. – [Электронный ресурс]. – URL: https://jobturbo.ru (дата обращения: 17.06.2026).
6. Автоотклик на вакансии: как настроить и использовать // HeadHunter: справочный центр. – [Электронный ресурс]. – URL: https://penza.hh.ru/article/31249 (дата обращения: 17.06.2026).
7. Саати, Т. Л. Принятие решений при зависимостях и обратных связях: аналитические сети / Т. Л. Саати. – М. : Librocom, 2008. – 360 с.
8. UML – диаграмма вариантов использования (use case diagram) // Habr: сообщество IT-специалистов. – [Электронный ресурс]. – URL: https://habr.com/ru/articles/47940/ (дата обращения: 17.05.2026).
9. FastAPI // Официальная документация фреймворка FastAPI. – [Электронный ресурс]. – URL: https://fastapi.tiangolo.com/ (дата обращения: 17.06.2026).
10. Docker Documentation // Docker : официальная документация. – [Электронный ресурс]. – URL: https://docs.docker.com/ (дата обращения: 17.06.2026).

11. Казаков, А. Организация модульного монолита / А. Казаков // Medium : платформа для публикации статей. – [Электронный ресурс]. – URL: https://medium.com/@kazakov.andr.v/организация-модульного-монолита-cacc0aaf5de4 (дата обращения: 17.06.2026).

12. Верников Г. Основы методологии IDEF1X // Проект «Корпоративный менеджмент» [Электронный ресурс]. URL: https://www.cfin.ru/vernikov/idef/idef1x.shtml/ (дата обращения 20.05.2026)

13. Просто о сложном. Концептуальная модель // MLKV : блог о базах данных. – [Электронный ресурс]. – URL: https://mlkv.ru/sample-page/материалы-по-базам-данных/концептуальная-модель/ (дата обращения: 18.05.2026).

14. The WebSocket Protocol: RFC 6455 // IETF Datatracker. – [Электронный ресурс]. – 2011. – URL: https://datatracker.ietf.org/doc/html/rfc6455 (дата обращения: 17.06.2026).

15. SQLite Documentation // SQLite : официальная документация. – [Электронный ресурс]. – URL: https://sqlite.org/docs.html (дата обращения: 17.06.2026).

16. aiohttp: асинхронный HTTP-фреймворк для Python // aio-libs : официальная документация. – [Электронный ресурс]. – URL: https://docs.aiohttp.org/ (дата обращения: 17.06.2026).

17. OpenAI API Documentation // OpenAI Platform : официальная документация. – [Электронный ресурс]. – URL: https://platform.openai.com/docs/api-reference (дата обращения: 17.06.2026).

18. Фундаментальная теория тестирования // Habr : публикация. – [Электронный ресурс]. – URL: https://habr.com/ru/articles/549054/ (дата обращения: 05.06.2026).

19. Иванов, П.В. Управление проектами : учебное пособие / П.В. Иванов. – Ростов н/Д : Феникс, 2018. – 102 с.

20. Создаём диаграмму Ганта // Unisender : официальный блог. – [Электронный ресурс]. – URL: https://www.unisender.com/ru/blog/chto-takoe-diagramma-ganta-i-kak-ee-sdelat/ (дата обращения: 05.06.2026).

Приложение А

(справочное)

Глоссарий

Автоматизированная система поиска вакансий – программная система, предназначенная для автоматизации процессов поиска вакансий, отправки откликов, обработки ответов работодателей и анализа результатов трудоустройства.

Учётная запись hh.ru – профиль пользователя на платформе HeadHunter, содержащий данные авторизации, резюме и историю взаимодействия с работодателями.

Языковая модель (LLM, Large Language Model) – модель искусственного интеллекта, предназначенная для генерации и обработки текстовой информации на естественном языке.

WebSocket – сетевой протокол, обеспечивающий двусторонний обмен данными между клиентом и сервером в режиме реального времени без необходимости постоянного обновления страницы.

REST API (Representational State Transfer Application Programming Interface) – программный интерфейс взаимодействия между компонентами системы посредством HTTP-запросов.

FastAPI – веб-фреймворк для языка Python, используемый для реализации серверной части разрабатываемой системы.

SPA (Single Page Application) – одностраничное веб-приложение, в котором обновление данных выполняется без полной перезагрузки страницы.

Фоновый процесс (Worker) – программный компонент, выполняющий длительные операции, такие как поиск вакансий, отправка откликов и обработка статистики, независимо от пользовательского интерфейса.

SQLite – встраиваемая реляционная система управления базами данных, используемая для хранения данных приложения.

Docker – платформа контейнеризации программного обеспечения, обеспечивающая развёртывание приложения в изолированной среде.

Docker Compose – инструмент управления многоконтейнерными приложениями, используемый для запуска компонентов системы.

Nginx – веб-сервер и обратный прокси-сервер, обеспечивающий обработку входящих запросов и передачу их приложению.

OAuth 2.0 – протокол авторизации, позволяющий получать доступ к ресурсам внешнего сервиса без передачи учётных данных пользователя.

Cookie-файл (Cookies) – набор служебных данных, используемых для хранения параметров авторизации и поддержания пользовательской сессии.

Логирование – процесс регистрации событий, возникающих в ходе работы программной системы.

Пользовательская сессия – состояние авторизованного взаимодействия пользователя с платформой hh.ru, необходимое для выполнения операций поиска вакансий и отправки откликов.

Приложение Б

(рекомендуемое)

Спецификация атрибутов таблиц

Таблица Б.1 – Спецификация атрибутов таблицы accounts

№ п/п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор пользователя	integer	15	PK	Да		
2	name	Имя пользователя	text	255		Да	unique	пустая строка
3	short_name	Короткое имя аккаунта	text	50		Да		пустая строка
4	color	Цветовая метка в дашборде	text	20				пустая строка
5	resume_hash	Хеш резюме на hh.ru	text	50		Да	unique	пустая строка
6	cookies_json	Cookies авторизации в формате JSON	text	2000				пустая строка
7	cover_letter	Сопроводительное письмо	text	2000				пустая строка
8	use_oauth	Флаг использования OAuth	boolean	1				false
9	apply_tests	Флаг пропуска вакансий с тестами	boolean	1				false
10	urls_json	Список поисковых URL в формате JSON	text	2000				[]
11	created_at	Дата создания записи	timestamp	20				CURRENT_TIMESTAMP
12	updated_at	Дата последнего обновления	timestamp	20				CURRENT_TIMESTAMP

Таблица Б.2 – Спецификация атрибутов таблицы system_config

№ п/п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор конфигурации	integer	15	PK	Да		1
2	pages_per_url	Количество страниц парсинга на URL	integer	3		Да		40
3	max_concurrent	Максимум параллельных потоков	integer	3		Да		20
4	response_delay	Задержка между откликами (сек)	integer	3		Да		1
5	pause_between_cycles	Пауза между циклами (сек)	integer	5		Да		60
6	min_salary	Минимальная зарплата фильтра (руб)	integer	10				0
7	auto_pause_errors	Авто-пауза после N ошибок	integer	2				5
8	daily_apply_limit	Суточный лимит откликов	integer	5				0
9	llm_enabled	Флаг включения автоответа LLM	boolean	1				false
10	llm_api_key	API-ключ LLM-сервиса	text	255				Пустая строка
11	llm_base_url	Базовый URL LLM API	text	255				https://api.openai.com/v1
12	llm_model	Модель LLM	text	50				gpt-4o-mini
13	updated_at	Дата последнего изменения	timestamp	20				CURRENT_TIMESTAMP

Таблица Б.3 – Спецификация атрибутов таблицы search_urls

№ п/п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор URL	integer	15	PK	Да		
2	url	Поисковый URL hh.ru	text	500		Да		пустая строка
3	pages	Количество страниц для парсинга	integer	3		Да		40
4	Created_at	Дата добавления	timestamp	20				CURRENT_TIMESTAMP

Таблица Б.4 – Спецификация атрибутов таблицы applied_vacacies

№ п/п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор записи	integer	15	PK	Да		
2	account_id	Идентификатор аккаунта	integer	15	FK	Да		
3	vacancy_id	Идентификатор вакансии на hh.ru	text	20		Да		пустая строка
4	vacancy_url	Ссылка на вакансию	text	255				пустая строка
5	title	Название вакансии	text	255				пустая строка
6	company	Название компании	text	255				пустая строка
7	salary_from	Зарплата от	integer	10				
8	salary_to	Зарплата до	integer	10				
9	applied_at	Дата и время отклика	timestamp	20		Да		CURRENT_TIMESTAMP

Таблица Б.5 – Спецификация атрибутов таблицы test_required_vacancies

№ п/п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор записи	integer	15	PK	Да		
2	vacancy_id	Идентификатор вакансии на hh.ru	text	20		Да	unique	пустая строка
3	vacancy_url	Ссылка на вакансию	text	255				пустая строка
4	title	Название вакансии	text	255				пустая строка
5	company	Название компании	text	255				пустая строка
6	account_id	Идентификатор аккаунта	integer	15	FK	Да		
7	resume_hash	Хеш резюме	text	50				пустая строка
8	detected_at	Дата обнаружения теста	timestamp	20				CURRENT_TIMESTAMP

Таблица Б.6 – Спецификация атрибутов таблицы interview_negotiations

№ п/п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения	Значение по умолчанию
1	neg_id	Идентификатор переговоров hh.ru	text	50	PK	Да	unique	
2	account_id	Идентификатор аккаунта	integer	15	FK	Да		
3	account_color	Цветовая метка аккаунта	text	20				пустая строка
4	employer	Название работодателя	text	255				пустая строка
5	vacancy_title	Название вакансии	text	255				пустая строка
6	first_seen	Дата первого обнаружения	timestamp	20				CURRENT_TIMESTAMP
7	last_seen	Дата последнего обновления	timestamp	20				CURRENT_TIMESTAMP
8	employer_last_msg	Последнее сообщение работодателя	text	1000				пустая строка
9	employer_last_msg_date	Дата последнего сообщения	timestamp	20				
10	needs_reply	Требуется ответ	boolean	1				false
11	llm_reply	Сгенерированный черновик ответа	text	2000				пустая строка
12	llm_reply_date	Дата генерации ответа	timestamp	20				
13	llm_sent	Флаг отправки LLM-ответа	boolean	1				false
14	chat_not_found	Чат недоступен (409)	boolean	1				false
15	chat_status	Внутренний статус чата	text	50				пустая строка
16	status	Статус переговоров	text	20				pending_reply

Таблица Б.7 – Спецификация атрибутов таблицы browser_sessions

№ п/п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор сессии	integer	15	PK	Да		
2	session_id	UUID браузерной сессии	text	100		Да	unique	пустая строка
3	session_data_json	Данные сессии в формате JSON	text	2000				пустая строка
4	created_at	Дата создания	timestamp	20				CURRENT_TIMESTAMP
5	updated_at	Дата обновления	timestamp	20				CURRENT_TIMESTAMP

Таблица Б.8 – Спецификация атрибутов таблицы letter_templates

№ п/п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор шаблона	integer	15	PK	Да		
2	template_name	Наименование шаблона	text	100		Да		Стандартное
3	template_text	Текст сопроводительного письма	text	2000		Да		пустая строка
4	created_at	Дата создания	timestamp	20				CURRENT_TIMESTAMP

Таблица Б.9 – Спецификация атрибутов таблицы questionnaire_templates

№ п/п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор шаблона	integer	15	PK	Да		
2	keywords_json	Ключевые слова для сопоставления в формате JSON	text	500				[]
3	answer_text	Текст ответа на опросник	text	500		Да		Готова рассказать подробнее на собеседовании
4	is_default	Флаг ответа по умолчанию	boolean	1				false
5	created_at	Дата создания	timestamp	20				CURRENT_TIMESTAMP

Приложение В

(обязательное)

Объекты обращения к базам данных

Листинг В.1 – Объекты аккаунтов

```
from dataclasses import dataclass, field
from typing import Optional
from datetime import datetime

@dataclass
class Account:
    """Объект таблицы accounts (Таблица Б.1)"""
    id: int
    name: str
    short_name: str
    color: Optional[str] = None
    resume_hash: str = ""
    cookies_json: str = "{}"          # поле cookies_json
    cover_letter: str = ""
    use_oauth: bool = False
    apply_tests: bool = False
    urls_json: str = "[]"             # список URL в JSON
    created_at: datetime = field(default_factory=datetime.now)
    updated_at: datetime = field(default_factory=datetime.now)
```

Листинг В.2 – Объекты глобальной конфигурации

```
@dataclass
class SystemConfig:
    """Объект таблицы system_config (Таблица Б.2)"""
    id: int = 1
    pages_per_url: int = 40
    max_concurrent: int = 20
    response_delay: int = 1
    pause_between_cycles: int = 60
    min_salary: int = 0
    auto_pause_errors: int = 5
    daily_apply_limit: int = 0
    llm_enabled: bool = False
    llm_api_key: str = ""
    llm_base_url: str = "https://api.openai.com/v1"
    llm_model: str = "gpt-4o-mini"
    updated_at: datetime = field(default_factory=datetime.now)
```

Листинг В.3 – Объекты поисковых URL

```
@dataclass
class SearchUrl:
    """Объект таблицы search_urls (Таблица Б.3)"""
    id: int
    url: str
    pages: int = 40

    created_at: datetime = field(default_factory=datetime.now)
```

Листинг В.4 – Объекты отправленных откликов

```
from typing import Optional
```

```
@dataclass
class AppliedVacancy:
    """Объект таблицы applied_vacancies (Таблица Б.4)"""
    id: int
    account_id: int                # FK → accounts.id
    vacancy_id: str
    vacancy_url: str = ""
    title: str = ""
    company: str = ""
    salary_from: Optional[int] = None
    salary_to: Optional[int] = None
    applied_at: datetime = field(default_factory=datetime.now)
```

Листинг В.5 – Объекты вакансий с тестами

```
@dataclass
class TestRequiredVacancy:
    """Объект таблицы test_required_vacancies (Таблица Б.5)"""
    id: int
    vacancy_id: str                # unique
    vacancy_url: str = ""
    title: str = ""
    company: str = ""
    account_id: int = 0            # FK → accounts.id
    resume_hash: str = ""

    detected_at: datetime = field(default_factory=datetime.now)
```

Листинг В.6 – Объекты переговоров

```
from typing import Optional
@dataclass
class InterviewNegotiation:
    """Объект таблицы interview_negotiations (Таблица Б.6)"""
    neg_id: str                    # PK – ID переговоров hh.ru
    account_id: int                # FK → accounts.id
    account_color: str = ""
    employer: str = ""
    vacancy_title: str = ""
    first_seen: datetime = field(default_factory=datetime.now)
    last_seen: datetime = field(default_factory=datetime.now)
    employer_last_msg: str = ""
    employer_last_msg_date: Optional[datetime] = None
    needs_reply: bool = False
    llm_reply: str = ""
    llm_reply_date: Optional[datetime] = None
    llm_sent: bool = False
    chat_not_found: bool = False
    chat_status: str = ""
    status: str = "pending_reply"
```

Листинг В.7 – Объекты браузерных сессий

```
@dataclass
```

```
class BrowserSession:
    """Объект таблицы browser_sessions (Таблица Б.7)"""
    id: int
    session_id: str                # unique UUID
    session_data_json: str = "{}"  # cookie / localStorage
    created_at: datetime = field(default_factory=datetime.now)
    updated_at: datetime = field(default_factory=datetime.now)
```

Листинг В.8 – Объекты шаблонов сопроводительных писем

```
@dataclass
class LetterTemplate:
    """Объект таблицы letter_templates (Таблица Б.8)"""
    id: int
    template_name: str = "Стандартное"
    template_text: str = ""
    created_at: datetime = field(default_factory=datetime.now)
```

Листинг В.9 – Объекты шаблонов ответов на опросники

```
@dataclass
class QuestionnaireTemplate:
    """Объект таблицы questionnaire_templates (Таблица Б.9)"""
    id: int
    keywords_json: str = "[]"      # список ключевых слов в JSON
    answer_text: str = "Готова рассказать подробнее на
собеседовании."
    is_default: bool = False
    created_at: datetime = field(default_factory=datetime.now)
```

Приложение Г

(обязательное)

SQL-скрипты

Листинг Г.1 – Скрипты создания таблиц

```
CREATE TABLE IF NOT EXISTS accounts
(
    id            SERIAL PRIMARY KEY,
    name          TEXT NOT NULL DEFAULT '',
    short_name    TEXT NOT NULL DEFAULT '',
    color         TEXT DEFAULT '',
    resume_hash   TEXT NOT NULL DEFAULT '',
    cookies_json  JSONB DEFAULT '{}',
    cover_letter  TEXT DEFAULT '',
    use_oauth     BOOLEAN DEFAULT FALSE,
    apply_tests   BOOLEAN DEFAULT FALSE,
    urls_json     JSONB DEFAULT '[]',
    created_at    TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at    TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT uq_accounts_name UNIQUE (name),
    CONSTRAINT uq_accounts_resume_hash UNIQUE (resume_hash)
);

CREATE TABLE IF NOT EXISTS system_config
(
    id                   INTEGER PRIMARY KEY DEFAULT 1 CHECK (id =
1),
    pages_per_url        INTEGER NOT NULL DEFAULT 40,
    max_concurrent       INTEGER NOT NULL DEFAULT 20,
    response_delay       INTEGER NOT NULL DEFAULT 1,
    pause_between_cycles INTEGER NOT NULL DEFAULT 60,
    min_salary           INTEGER DEFAULT 0,
    auto_pause_errors    INTEGER DEFAULT 5,
    daily_apply_limit    INTEGER DEFAULT 0,
    llm_enabled          BOOLEAN DEFAULT FALSE,
    llm_api_key          TEXT DEFAULT '',
    llm_base_url         TEXT DEFAULT 'https://api.openai.com/v1',
    llm_model            TEXT DEFAULT 'gpt-4o-mini',
    updated_at           TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE IF NOT EXISTS search_urls
(
    id         SERIAL PRIMARY KEY,
    url        TEXT NOT NULL DEFAULT '',
    pages      INTEGER NOT NULL DEFAULT 40,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE IF NOT EXISTS applied_vacancies
(
    id          SERIAL PRIMARY KEY,
    account_id  INTEGER NOT NULL REFERENCES accounts(id) ON DELETE
CASCADE,
    vacancy_id  TEXT NOT NULL DEFAULT '',
    vacancy_url TEXT DEFAULT '',
```

```
    title       TEXT DEFAULT '',
    company     TEXT DEFAULT '',
    salary_from INTEGER,
    salary_to   INTEGER,
    applied_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE IF NOT EXISTS test_required_vacancies
(
    id           SERIAL PRIMARY KEY,
    vacancy_id   TEXT NOT NULL UNIQUE DEFAULT '',
    vacancy_url  TEXT DEFAULT '',
    title        TEXT DEFAULT '',
    company      TEXT DEFAULT '',
    account_id   INTEGER NOT NULL REFERENCES accounts(id) ON
DELETE CASCADE,
    resume_hash  TEXT DEFAULT '',
    detected_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE IF NOT EXISTS interview_negotiations
(
    neg_id                  VARCHAR(50) PRIMARY KEY,
    account_id              INTEGER NOT NULL REFERENCES
accounts(id) ON DELETE CASCADE,
    account_color           TEXT DEFAULT '',
    employer                TEXT DEFAULT '',
    vacancy_title           TEXT DEFAULT '',
    first_seen              TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    last_seen               TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    employer_last_msg       TEXT DEFAULT '',
    employer_last_msg_date  TIMESTAMP,
    needs_reply             BOOLEAN DEFAULT FALSE,
    llm_reply               TEXT DEFAULT '',
    llm_reply_date          TIMESTAMP,
    llm_sent                BOOLEAN DEFAULT FALSE,
    chat_not_found          BOOLEAN DEFAULT FALSE,
    chat_status             TEXT DEFAULT '',
    status                  TEXT DEFAULT 'pending_reply'
);

CREATE TABLE IF NOT EXISTS browser_sessions
(
    id                SERIAL PRIMARY KEY,
    session_id        TEXT NOT NULL UNIQUE DEFAULT '',
    session_data_json JSONB DEFAULT '{}',
    created_at        TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at        TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE IF NOT EXISTS letter_templates
(
    id             SERIAL PRIMARY KEY,
```

```
    template_name  TEXT NOT NULL DEFAULT 'Стандартное',
    template_text  TEXT NOT NULL DEFAULT '',
    created_at     TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE IF NOT EXISTS questionnaire_templates
(
    id             SERIAL PRIMARY KEY,
    keywords_json  JSONB DEFAULT '[]',
    answer_text    TEXT NOT NULL DEFAULT 'Готова рассказать
подробнее на собеседовании.',
    is_default     BOOLEAN DEFAULT FALSE,
    created_at     TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Листинг Г.2 – Скрипты вставки и обновления данных

```
-- Вставка нового аккаунта
INSERT INTO accounts (name, short_name, color, resume_hash,
cookies_json,
                      cover_letter, use_oauth, apply_tests,
urls_json)
VALUES (:name, :short_name, :color, :resume_hash, :cookies_json,
        :cover_letter, :use_oauth, :apply_tests, :urls_json)
RETURNING id;

-- Обновление конфигурации (singleton)
INSERT INTO system_config (id, pages_per_url, max_concurrent,
response_delay,
                           pause_between_cycles, min_salary,
auto_pause_errors,
                           daily_apply_limit, llm_enabled,
llm_api_key,
                           llm_base_url, llm_model)
VALUES (1, :pages_per_url, :max_concurrent, :response_delay,
        :pause_between_cycles, :min_salary, :auto_pause_errors,
        :daily_apply_limit, :llm_enabled, :llm_api_key,
        :llm_base_url, :llm_model)
ON CONFLICT (id) DO UPDATE SET
    pages_per_url = EXCLUDED.pages_per_url,
    max_concurrent = EXCLUDED.max_concurrent,
    response_delay = EXCLUDED.response_delay,
    pause_between_cycles = EXCLUDED.pause_between_cycles,
    min_salary = EXCLUDED.min_salary,
    auto_pause_errors = EXCLUDED.auto_pause_errors,
    daily_apply_limit = EXCLUDED.daily_apply_limit,
    llm_enabled = EXCLUDED.llm_enabled,
    llm_api_key = EXCLUDED.llm_api_key,
    llm_base_url = EXCLUDED.llm_base_url,
    llm_model = EXCLUDED.llm_model,
    updated_at = CURRENT_TIMESTAMP;

-- Добавление поискового URL в глобальный пул
INSERT INTO search_urls (url, pages)
```

```
VALUES (:url, :pages)
RETURNING id;

-- Регистрация отправленного отклика
INSERT INTO applied_vacancies (account_id, vacancy_id,
vacancy_url,
                               title, company, salary_from,
salary_to)
VALUES (:account_id, :vacancy_id, :vacancy_url,
        :title, :company, :salary_from, :salary_to)
RETURNING id;

-- Регистрация вакансии, требующей теста
INSERT INTO test_required_vacancies (vacancy_id, vacancy_url,
title, company,
                                     account_id, resume_hash)
VALUES (:vacancy_id, :vacancy_url, :title, :company,
        :account_id, :resume_hash)
ON CONFLICT (vacancy_id) DO NOTHING;

-- Создание или обновление записи переговоров
INSERT INTO interview_negotiations (neg_id, account_id,
account_color,
                                    employer, vacancy_title,
first_seen,
                                    last_seen, employer_last_msg,
                                    employer_last_msg_date,
needs_reply,
                                    llm_reply, llm_reply_date,
llm_sent,
                                    chat_not_found, chat_status,
status)
VALUES (:neg_id, :account_id, :account_color,
        :employer, :vacancy_title, CURRENT_TIMESTAMP,
        CURRENT_TIMESTAMP, :employer_last_msg,
        :employer_last_msg_date, :needs_reply,
        :llm_reply, :llm_reply_date, :llm_sent,
        :chat_not_found, :chat_status, :status)
ON CONFLICT (neg_id) DO UPDATE SET
    last_seen = CURRENT_TIMESTAMP,
    employer_last_msg = EXCLUDED.employer_last_msg,
    employer_last_msg_date = EXCLUDED.employer_last_msg_date,
    needs_reply = EXCLUDED.needs_reply,
    llm_reply = EXCLUDED.llm_reply,
    llm_reply_date = EXCLUDED.llm_reply_date,
    llm_sent = EXCLUDED.llm_sent,
    chat_not_found = EXCLUDED.chat_not_found,
    chat_status = EXCLUDED.chat_status,
    status = EXCLUDED.status;

-- Сохранение браузерной сессии
INSERT INTO browser_sessions (session_id, session_data_json)
VALUES (:session_id, :session_data_json)
```

```
ON CONFLICT (session_id) DO UPDATE SET
    session_data_json = EXCLUDED.session_data_json,
    updated_at = CURRENT_TIMESTAMP
RETURNING id;

-- Добавление шаблона сопроводительного письма
INSERT INTO letter_templates (template_name, template_text)
VALUES (:template_name, :template_text)
RETURNING id;

-- Добавление шаблона ответа на опросник
INSERT INTO questionnaire_templates (keywords_json, answer_text,
is_default)
VALUES (:keywords_json, :answer_text, :is_default)
RETURNING id;
```

Листинг Г.3 – Скрипты получения данных

```
-- Получение списка всех аккаунтов
SELECT id, name, short_name, color, resume_hash,
       use_oauth, apply_tests, created_at, updated_at
FROM accounts;

-- Получение аккаунта по resume_hash
SELECT id, name, short_name, color, resume_hash,
       cookies_json, cover_letter, use_oauth, apply_tests,
urls_json
FROM accounts
WHERE resume_hash = :resume_hash;

-- Получение текущей конфигурации
SELECT id, pages_per_url, max_concurrent, response_delay,
       pause_between_cycles, min_salary, auto_pause_errors,
       daily_apply_limit, llm_enabled, llm_api_key,
       llm_base_url, llm_model, updated_at
FROM system_config
WHERE id = 1;

-- Получение глобального пула поисковых URL
SELECT id, url, pages, created_at
FROM search_urls;

-- Получение истории откликов по аккаунту
SELECT id, vacancy_id, vacancy_url, title, company,
       salary_from, salary_to, applied_at
FROM applied_vacancies
WHERE account_id = :account_id
ORDER BY applied_at DESC;

-- Проверка: откликался ли аккаунт на вакансию
SELECT 1
FROM applied_vacancies
WHERE account_id = :account_id AND vacancy_id = :vacancy_id;
```

```
-- Получение списка вакансий с тестами
SELECT id, vacancy_id, vacancy_url, title, company,
       account_id, resume_hash, detected_at
FROM test_required_vacancies
ORDER BY detected_at DESC;

-- Получение переговоров по аккаунту с сортировкой
SELECT neg_id, account_id, account_color, employer, vacancy_title,
       first_seen, last_seen, employer_last_msg,
       employer_last_msg_date, needs_reply, llm_reply,
       llm_reply_date, llm_sent, chat_not_found,
       chat_status, status
FROM interview_negotiations
WHERE account_id = :account_id
ORDER BY
    CASE status
        WHEN 'pending_reply' THEN 0
        WHEN 'draft' THEN 1
        WHEN 'replied' THEN 2
        WHEN 'chat_closed' THEN 3
        ELSE 4
    END,
    last_seen DESC;

-- Получение статистики откликов по всем аккаунтам
SELECT account_id,
       COUNT(*) AS total_applied,
       COUNT(*) FILTER (WHERE applied_at >= CURRENT_DATE) AS
today_applied
FROM applied_vacancies
GROUP BY account_id;

-- Получение количества вакансий с тестами
SELECT COUNT(*) AS test_count
FROM test_required_vacancies;

-- Получение данных для дашборда:
-- объединение откликов и тест-вакансий с привязкой к аккаунтам
SELECT a.name AS account_name,
       COUNT(DISTINCT av.vacancy_id) AS total_sent,
       COUNT(DISTINCT tv.vacancy_id) AS total_tests
FROM accounts a
LEFT JOIN applied_vacancies av ON a.id = av.account_id
LEFT JOIN test_required_vacancies tv ON a.id = tv.account_id
GROUP BY a.id, a.name;

-- Получение шаблонов сопроводительных писем
SELECT id, template_name, template_text, created_at
FROM letter_templates;

-- Получение шаблонов ответов на опросники
SELECT id, keywords_json, answer_text, is_default, created_at
FROM questionnaire_templates;
```

Приложение Д

(обязательное)

Структуры слоя представления данных

Листинг Д.1 – Слой представления core-service

```
# app/routes/core.py
router = APIRouter()

@router.on_event("startup")
async def startup()

@router.get("/")
async def index()

@router.websocket("/ws")
async def websocket_endpoint(ws: WebSocket)

@router.post("/api/pause")
async def api_pause()

@app.get("/api/ws_token")        # в app/routes/__init__.py
async def api_ws_token()
```

Листинг Д.2 – Слой представления account-service

```
# app/routes/accounts.py
router = APIRouter()

@router.post("/api/account/{idx}/pause")
async def api_account_pause(idx: int)

@router.post("/api/account/{idx}/llm_toggle")
async def api_account_llm_toggle(idx: int)

@router.post("/api/account/{idx}/resume_touch")
async def api_resume_touch(idx: int)

@router.post("/api/account/{idx}/resume_touch_toggle")
async def api_resume_touch_toggle(idx: int)

@router.post("/api/account/{idx}/set_urls")
async def api_set_urls(idx: int, request: Request)

@router.post("/api/account/{idx}/set_letter")
async def api_set_letter(idx: int, request: Request)

@router.post("/api/account/{idx}/update_cookies")
async def api_update_cookies(idx: int, body: dict)

@router.post("/api/account/{idx}/profile")
async def api_account_profile(idx: int, request: Request)

@router.post("/api/account/{idx}/credentials")
async def api_account_credentials(idx: int, request: Request)

@router.post("/api/account/{idx}/autologin")
async def api_account_autologin(idx: int)
```

```
@router.post("/api/accounts/add")
async def api_account_add(request: Request)

@router.delete("/api/account/{idx}/delete")
async def api_account_delete(idx: int)

@router.post("/api/account/{idx}/apply_tests")
async def api_apply_tests(idx: int)

@router.get("/api/account/{idx}/resume_text")
async def api_resume_text(idx: int)

@router.get("/api/account/{idx}/resume_views")
async def api_resume_views(idx: int)

@router.post("/api/account/{idx}/oauth_token")
async def api_oauth_token(idx: int)

@router.get("/api/account/{idx}/oauth_status")
async def api_oauth_status(idx: int)

@router.post("/api/account/{idx}/oauth_touch")
async def api_oauth_touch(idx: int)

@router.get("/api/account/{idx}/test_llm_questionnaire")
async def api_test_llm_questionnaire(idx: int, vacancy_id: str =
"")

@router.get("/api/account/{idx}/resume_audit")
async def api_resume_audit(idx: int, extra_terms: str = "")

@router.get("/api/account/{idx}/hot_leads")
async def api_hot_leads(idx: int)

@router.get("/api/account/{idx}/remindable")
async def api_remindable(idx: int)

@router.post("/api/account/{idx}/clone_resume")
async def api_clone_resume(idx: int, request: Request)

@router.post("/api/account/{idx}/edit_resume")
async def api_edit_resume(idx: int, request: Request)

@router.get("/api/account/{idx}/all_resumes")
async def api_all_resumes(idx: int)

@router.post("/api/account/{idx}/decline_discards")
async def api_decline_discards(idx: int)
```

Листинг Д.3 – Слой представления session-service

```
# app/routes/sessions.py
router = APIRouter()
```

```
@router.get("/api/sessions")
async def api_sessions()

@router.post("/api/session/add")
async def api_session_add(body: dict)

@router.patch("/api/session/{idx}")
async def api_session_patch(idx: int, body: dict)

@router.post("/api/session/{idx}/activate")
async def api_session_activate(idx: int)

@router.post("/api/session/{idx}/refresh")
async def api_session_refresh(idx: int)

@router.delete("/api/session/{idx}")
async def api_session_delete(idx: int)

@router.post("/api/session/{idx}/profile")
async def api_session_profile(idx: int, request: Request)
```

Листинг Д.4 – Слой представления apply-service

```
# app/routes/apply.py
router = APIRouter()

@router.post("/api/apply/check")
async def api_apply_check(body: dict)

@router.post("/api/apply/submit")
async def api_apply_submit(body: dict)
```

Листинг Д.5 – Слой представления data-service

```
# app/routes/data.py
router = APIRouter()

@router.get("/api/applied")
async def api_applied(limit: int = 300)

@router.get("/api/tests")
async def api_tests(limit: int = 300)

@router.get("/api/interviews")
async def api_interviews(acc: str = "", limit: int = 2000, status:
str = "")

@router.get("/api/vacancies")
async def api_vacancies(limit: int = 3000)

@router.delete("/api/vacancy/{vacancy_id}")
async def api_vacancy_delete(vacancy_id: str, account: str = "")
```

```
@router.get("/api/hr_contacts")
async def api_hr_contacts()
```

Листинг Д.6 – Слой представления settings-service

```
# app/routes/settings.py
router = APIRouter()

@router.post("/api/settings")
async def api_settings(update: ConfigUpdate)

@router.get("/api/raw/config")
async def api_raw_config_get()

@router.post("/api/raw/config")
async def api_raw_config_set(request: Request)

@router.get("/api/raw/accounts")
async def api_raw_accounts_get()

@router.post("/api/raw/accounts")
async def api_raw_accounts_set(request: Request)
```

Листинг Д.7 – Слой представления llm-service

```
# app/routes/llm.py
router = APIRouter()

@router.post("/api/llm_profiles")
async def api_llm_profiles(request: Request)

@router.post("/api/llm_toggle")
async def api_llm_toggle()

@router.post("/api/llm_config")
async def api_llm_config(request: Request)

@router.post("/api/llm_run_now")
async def api_llm_run_now()

@router.post("/api/llm_reset_replied")
async def api_llm_reset_replied()

@router.post("/api/llm_detect")
async def api_llm_detect(request: Request)
```

Листинг Д.8 – Слой представления debug-service

```
# app/routes/debug.py
router = APIRouter()

@router.get("/api/debug/session/{idx}")
async def api_debug_session(idx: int)

@router.get("/api/debug")
```

```
async def api_debug()

@router.get("/api/debug/neg_ids/{idx}")
async def api_debug_neg_ids(idx: int)

@router.get("/api/debug/thread/{idx}/{chat_id}")
async def api_debug_thread(idx: int, chat_id: str)

@router.get("/api/debug/thread_raw/{idx}/{chat_id}")
async def api_debug_thread_raw(idx: int, chat_id: str)
```

Приложение Е

(обязательное)

Структуры слоя бизнес-логики

Листинг Е.1 – Слой бизнес-логики auth-service

```
# app/hh_auth.py / app/oauth.py

def login_and_get_cookies(login: str, password: str) ->
Optional[dict]

def _obtain_oauth_token(acc: dict) -> str

def _oauth_apply(acc: dict, vid: str, message: str = "") -> tuple

def _oauth_touch_resume(acc: dict) -> tuple
```

Листинг Е.2 – Слой бизнес-логики apply-service

```
# app/hh_apply.py

async def send_response_async(acc: dict, vid: str) -> tuple

async def fill_and_submit_questionnaire(acc: dict, vid: str,
                                        vacancy_title: str = "",
company: str = "") -> tuple

def _check_vacancy_before_apply(acc: dict, vid: str) -> dict

def check_limit(acc: dict) -> bool


def touch_resume(acc: dict) -> tuple
```

Листинг Е.3 – Слой бизнес-логики questionnaire-service

```
# app/questionnaire.py

def get_questionnaire_answer(question_text: str) -> str

def _parse_questionnaire_fields(html: str) -> tuple

def _parse_questionnaire_rich(html: str) -> list
```

Листинг Е.4 – Слой бизнес-логики llm-service

```
# app/llm.py

def _randomize_text(template: str) -> str

def generate_llm_reply(conversation: list, employer_name: str =
"",
                       cover_letter: str = "", resume_text: str =
"") -> str

def generate_llm_questionnaire_answers(rich_questions: list,
                                       vacancy_title: str = "",
company: str = "",
```

```
                                        resume_text: str = "") ->
dict
```

Листинг Е.5 – Слой бизнес-логики manager-service

```
# app/manager.py

class Dashboard:
    def __init__(self)
    def start(self)
    def stop(self)
    def toggle_pause(self)
    def toggle_account_pause(self, idx: int)
    def activate_session(self, temp_idx: int) -> bool
    def _build_session_urls(self, resume_hash: str) -> list[str]
    def _run_account_worker(self, idx: int, state: AccountState)
    def _fetch_hh_stats_worker(self, idx: int, state:
AccountState)
```

Листинг Е.6 – Слой бизнес-логики state-service

```
# app/state.py

class AccountState:
    def __init__(self, acc_data: dict)
    # Атрибуты: name, short, color, status, sent, tests, errors,
    #   daily_sent, hard_stopped, paused, limit_exceeded,
    #   resume_views_7d, resume_invitations_7d, llm_enabled,
    #   llm_replied_count, action_history, ...
```

Листинг Е.7 – Слой бизнес-логики storage-service

```
# app/storage.py

def _load_cache()

def add_applied(account_name: str, vacancy_id: str, info: dict =
None)

def add_test_vacancy(vacancy_id: str, title: str = "", company:
str = "",
                     account_name: str = "", resume_hash: str =
"")

def is_applied(account_name: str, vacancy_id: str) -> bool

def is_test(vacancy_id: str) -> bool

def get_stats() -> dict

def get_applied_list(limit: int = 300) -> list

def get_test_list(limit: int = 300) -> list

def get_vacancy_db(limit: int = 3000) -> list
```

```
def upsert_interview(neg_id: str, acc: str, ...) -> None

def get_interviews_list(acc: str = "", limit: int = 2000, status:
str = "") -> list

def load_browser_sessions() -> list

def save_browser_sessions(sessions: list)
```

Приложение Ж

(обязательное)

Структуры слоя обращения к базе данных

Листинг Ж.1 – Слой обращения к базе данных accounts-service

```
# app/repositories/accounts_repository.py
from typing import Optional
from bson import ObjectId
from pymongo.collection import Collection
class AccountsRepository:
    """Репозиторий коллекции accounts."""
    def __init__(self, collection: Collection):
        self._collection = collection
    def get_by_id(self, account_id: str) -> Optional[dict]:
        return self._collection.find_one({"_id":
ObjectId(account_id)})
    def get_by_resume_hash(self, resume_hash: str) ->
Optional[dict]:
        return self._collection.find_one({"resume_hash":
resume_hash})
    def get_by_name(self, name: str) -> Optional[dict]:
        return self._collection.find_one({"name": name})
    def create(self, account: dict) -> str:
        result = self._collection.insert_one(account)
        return str(result.inserted_id)
    def update(self, account_id: str, update: dict) -> bool:
        result = self._collection.update_one(
            {"_id": ObjectId(account_id)}, {"$set": update}
        )
        return result.modified_count > 0
    def delete(self, account_id: str) -> bool:
        result = self._collection.delete_one({"_id":
ObjectId(account_id)})
        return result.deleted_count > 0
    def list_all(self) -> list[dict]:
        return list(self._collection.find())
```

Листинг Ж.2 – Слой обращения к базе данных config-service

```
# app/repositories/config_repository.py
from typing import Optional
from pymongo.collection import Collection

class ConfigRepository:
    """Репозиторий коллекции system_config (singleton-
документ)."""

    def __init__(self, collection: Collection):
        self._collection = collection

    def get_config(self) -> Optional[dict]:
        return self._collection.find_one({"_id": 1})

    def save_config(self, config: dict) -> bool:
        config["_id"] = 1
        result = self._collection.replace_one(
            {"_id": 1}, config, upsert=True
```

```
        )
        return result.upserted_id is not None or
result.modified_count > 0

    def update_field(self, field: str, value) -> bool:
        result = self._collection.update_one(
            {"_id": 1}, {"$set": {field: value}}, upsert=True
        )
        return result.modified_count > 0 or result.upserted_id is
not None
```

Листинг Ж.3 – Слой обращения к базе данных applied-vacancies-service

```
# app/repositories/applied_vacancies_repository.py
from typing import Optional
from bson import ObjectId
from pymongo.collection import Collection

class AppliedVacanciesRepository:
    """Репозиторий коллекции applied_vacancies."""

    def __init__(self, collection: Collection):
        self._collection = collection

    def add_applied(self, account_id: str, vacancy_id: str, info:
dict) -> str:
        doc = {
            "account_id": ObjectId(account_id),
            "vacancy_id": vacancy_id,
            "vacancy_url": info.get("url",
f"https://hh.ru/vacancy/{vacancy_id}"),
            "title": info.get("title", ""),
            "company": info.get("company", ""),
            "salary_from": info.get("salary_from"),
            "salary_to": info.get("salary_to"),
            "applied_at": info.get("at")
        }
        result = self._collection.insert_one(doc)
        return str(result.inserted_id)

    def is_applied(self, account_id: str, vacancy_id: str) ->
bool:
        return self._collection.find_one({
            "account_id": ObjectId(account_id), "vacancy_id":
vacancy_id
        }) is not None

    def get_by_account(self, account_id: str, limit: int = 300) ->
list[dict]:
        return list(
            self._collection
            .find({"account_id": ObjectId(account_id)})
            .sort("applied_at", -1)
```

```
            .limit(limit)
        )

    def get_stats(self) -> dict:
        total = self._collection.count_documents({})
        pipeline = [
            {"$group": {"_id": "$account_id", "count": {"$sum":
1}}}
        ]
        by_acc = {str(r["_id"]): r["count"] for r in
self._collection.aggregate(pipeline)}
        return {"total": total, "by_acc": by_acc}
```

Листинг Ж.4– Слой обращения к базе данных test-vacancies-service

```
# app/repositories/test_vacancies_repository.py
from typing import Optional
from pymongo.collection import Collection
class TestVacanciesRepository:
    """Репозиторий коллекции test_required_vacancies."""
    def __init__(self, collection: Collection):
        self._collection = collection
    def add_test(self, vacancy_id: str, title: str = "", company:
str = "",
                 account_id: str = "", resume_hash: str = "") ->
str:
        doc = {
            "vacancy_id": vacancy_id,
            "vacancy_url": f"https://hh.ru/vacancy/{vacancy_id}",
            "title": title,
            "company": company,
            "account_id": account_id,
            "resume_hash": resume_hash,
            "detected_at": ""
        }
        result = self._collection.insert_one(doc)
        return str(result.inserted_id)
    def is_test(self, vacancy_id: str) -> bool:
        return self._collection.find_one({"vacancy_id":
vacancy_id}) is not None
    def list_all(self, limit: int = 300) -> list[dict]:
        return list(self._collection.find().sort("detected_at", -
1).limit(limit))
```

Листинг Ж.5 – Слой обращения к базе данных interview-service

```
# app/repositories/interview_repository.py
from typing import Optional, Set
from bson import ObjectId
from pymongo.collection import Collection
class InterviewRepository:
    """Репозиторий коллекции interview_negotiations."""
    def __init__(self, collection: Collection):
        self._collection = collection
```

```
    def upsert_interview(self, neg_id: str, account_id: str,
                         employer: str = "", vacancy_title: str =
"",
                         employer_last_msg: str = None,
                         needs_reply: bool = None,
                         llm_reply: str = None,
                         llm_sent: bool = None,
                         chat_not_found: bool = None,
                         chat_status: str = None) -> bool:
        update = {
            "$set": {
                "neg_id": neg_id,
                "account_id": ObjectId(account_id),
                "employer": employer,
                "vacancy_title": vacancy_title,
                "last_seen": ""
            },
            "$setOnInsert": {"first_seen": ""}
        }
        if employer_last_msg is not None:
            update["$set"]["employer_last_msg"] =
employer_last_msg
        if needs_reply is not None:
            update["$set"]["needs_reply"] = needs_reply
        if llm_reply is not None:
            update["$set"]["llm_reply"] = llm_reply
        if llm_sent is not None:
            update["$set"]["llm_sent"] = llm_sent
        if chat_not_found is True:
            update["$set"]["chat_not_found"] = True
        if chat_status is not None:
            update["$set"]["chat_status"] = chat_status
        result = self._collection.update_one(
            {"neg_id": neg_id}, update, upsert=True
        )
        return result.matched_count > 0 or result.upserted_id is
not None
    def get_by_status(self, status: str, account_id: str = "",
                      limit: int = 2000) -> list[dict]:
        query = {"status": status}
        if account_id:
            query["account_id"] = ObjectId(account_id)
        return list(
            self._collection
            .find(query)
            .sort("last_seen", -1)
            .limit(limit)
        )
    def get_no_chat_neg_ids(self) -> Set[str]:
        return {
            r["neg_id"] for r in self._collection.find(
                {"chat_not_found": True}, {"neg_id": 1}
            )}
```

Листинг Ж.6 – Слой обращения к базе данных session-service

```
# app/repositories/session_repository.py
from typing import Optional
from bson import ObjectId
from pymongo.collection import Collection

class SessionRepository:
    """Репозиторий коллекции browser_sessions."""

    def __init__(self, collection: Collection):
        self._collection = collection

    def get_all(self) -> list[dict]:
        return list(self._collection.find())

    def get_by_id(self, session_id: str) -> Optional[dict]:
        return self._collection.find_one({"session_id":
session_id})

    def create(self, session: dict) -> str:
        result = self._collection.insert_one(session)
        return str(result.inserted_id)

    def update(self, session_id: str, update: dict) -> bool:
        result = self._collection.update_one(
            {"session_id": session_id}, {"$set": update}
        )
        return result.modified_count > 0

    def delete(self, session_id: str) -> bool:
        result = self._collection.delete_one({"session_id":
session_id})
        return result.deleted_count > 0
```

Приложение И

(рекомендуемое)

Листинг модульных тестов

Листинг И.1 – Файл testing/test_oauth.py

```
import time
from unittest.mock import patch, MagicMock

import pytest

import app.oauth as oauth_module
from app.oauth import _obtain_oauth_token


# TC-01: отсутствует resume_hash
def test_tc01_no_resume_hash(mock_acc_no_hash):
    result = _obtain_oauth_token(mock_acc_no_hash)
    assert result == ""


# TC-02: используется актуальный токен из кэша
def test_tc02_cached_token(mock_acc, cached_token_data):
    with patch.dict(
            oauth_module._oauth_tokens,
            {"abc123def456": cached_token_data}
    ):
        with patch("app.oauth.requests.post") as mp, \
             patch("app.oauth.requests.get") as mg:

            result = _obtain_oauth_token(mock_acc)

            assert result == "cached_access_token_xyz"

            mp.assert_not_called()
            mg.assert_not_called()


# TC-03: успешное обновление токена через refresh_token
def test_tc03_refresh_success(mock_acc, expired_token_data):

    resp = MagicMock()
    resp.status_code = 200
    resp.json.return_value = {
        "access_token": "new_token",
        "refresh_token": "new_refresh",
        "expires_in": 1209599
    }

    with patch.dict(
            oauth_module._oauth_tokens,
            {"abc123def456": expired_token_data}
    ):
        with patch("app.oauth.requests.post", return_value=resp):

            result = _obtain_oauth_token(mock_acc)
```

```
            assert result == "new_token"


# TC-04: ошибка при обновлении токена
def test_tc04_refresh_failed(mock_acc, expired_token_data):

    resp = MagicMock()
    resp.status_code = 401

    with patch.dict(
            oauth_module._oauth_tokens,
            {"abc123def456": expired_token_data}
    ):
        with patch("app.oauth.requests.post", return_value=resp):

            result = _obtain_oauth_token(mock_acc)

            assert result == ""


# TC-05: успешное получение authorization code
def test_tc05_authorize_success(mock_acc):

    auth_resp = MagicMock()
    auth_resp.status_code = 302
    auth_resp.headers = {
        "Location": "hhandroid://oauth?code=GOOD_CODE"
    }

    token_resp = MagicMock()
    token_resp.status_code = 200
    token_resp.json.return_value = {
        "access_token": "access_token",
        "refresh_token": "refresh_token",
        "expires_in": 1209599
    }

    with patch.dict(oauth_module._oauth_tokens, {}, clear=True):
        with patch(
                "app.oauth.requests.get",
                return_value=auth_resp
        ):
            with patch(
                    "app.oauth.requests.post",
                    return_value=token_resp
            ):

                result = _obtain_oauth_token(mock_acc)

                assert result == "access_token"


# TC-06: ошибка получения authorization code
```

```
def test_tc06_authorize_failed(mock_acc):

    auth_resp = MagicMock()
    auth_resp.status_code = 500

    with patch.dict(oauth_module._oauth_tokens, {}, clear=True):
        with patch(
                "app.oauth.requests.get",
                return_value=auth_resp
        ):

            result = _obtain_oauth_token(mock_acc)

            assert result == ""


# TC-07: ошибка обмена authorization code на access token
def test_tc07_token_exchange_fails(mock_acc):

    auth_resp = MagicMock()
    auth_resp.status_code = 302
    auth_resp.headers = {
        "Location": "hhandroid://oauth?code=BAD"
    }
    bad_resp = MagicMock()
    bad_resp.status_code = 401

    with patch.dict(oauth_module._oauth_tokens, {}, clear=True):
        with patch(
                "app.oauth.requests.get",
                return_value=auth_resp
        ):
            with patch(
                    "app.oauth.requests.post",
                    return_value=bad_resp
            ):

                result = _obtain_oauth_token(mock_acc)

                assert result == ""
# TC-08: проверка истечения срока действия токена
def test_tc08_expired_token_cleanup(mock_acc, expired_token_data):

    expired_token_data["expires_at"] = time.time() - 100

    with patch.dict(
            oauth_module._oauth_tokens,
            {"abc123def456": expired_token_data}
    ):
        result = _obtain_oauth_token(mock_acc)

        assert result == ""
```

Приложение К

(рекомендуемое)

Иллюстрационные материалы к тестированию API

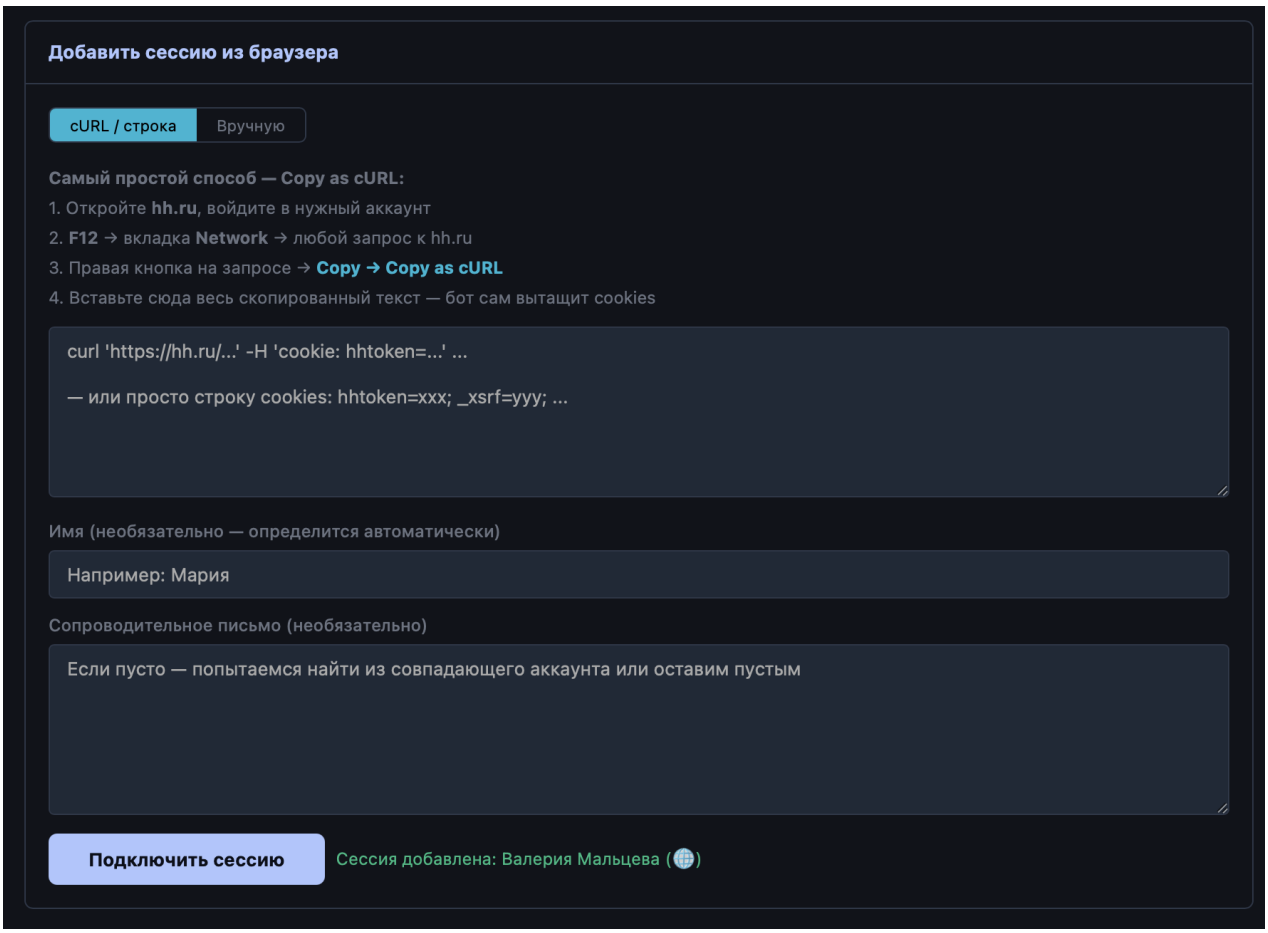


Рисунок К.1 – Добавление браузерной сессии

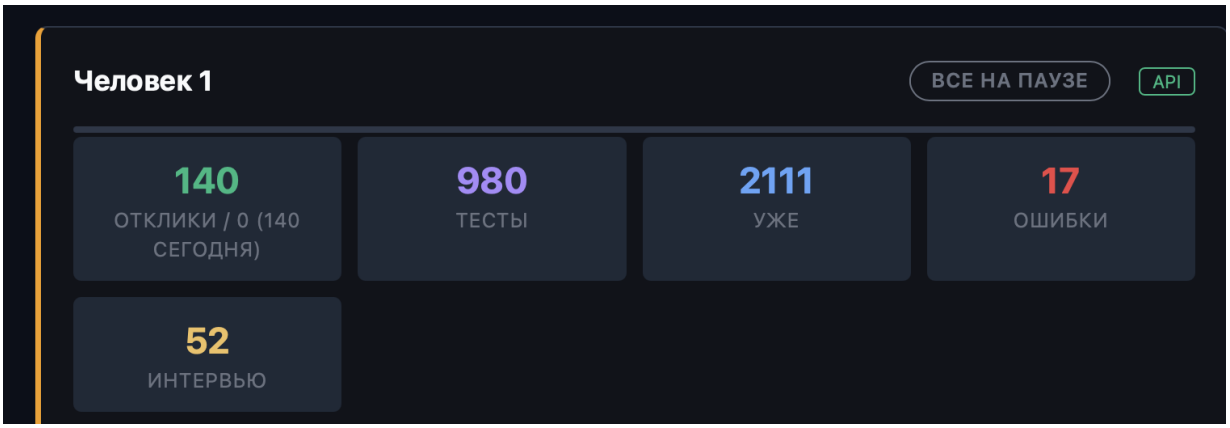


Рисунок К.2 – Переключение режима откликов

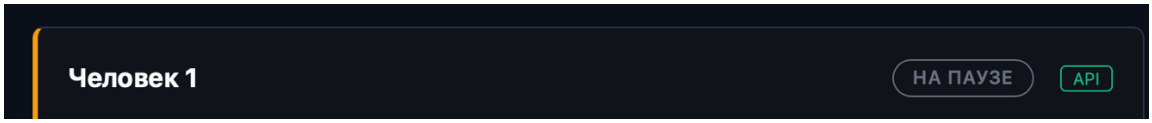


Рисунок К.3 – Постановка на паузу

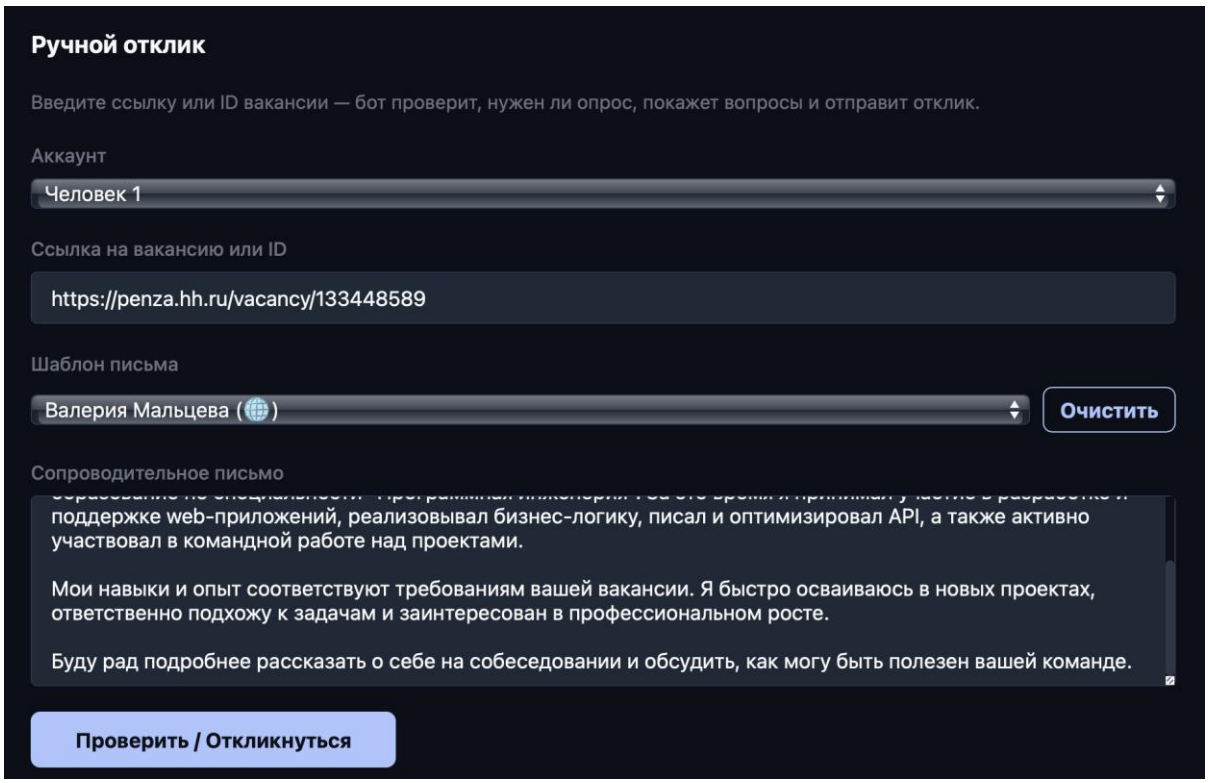


Рисунок К.4 – Отправка тестового отклика

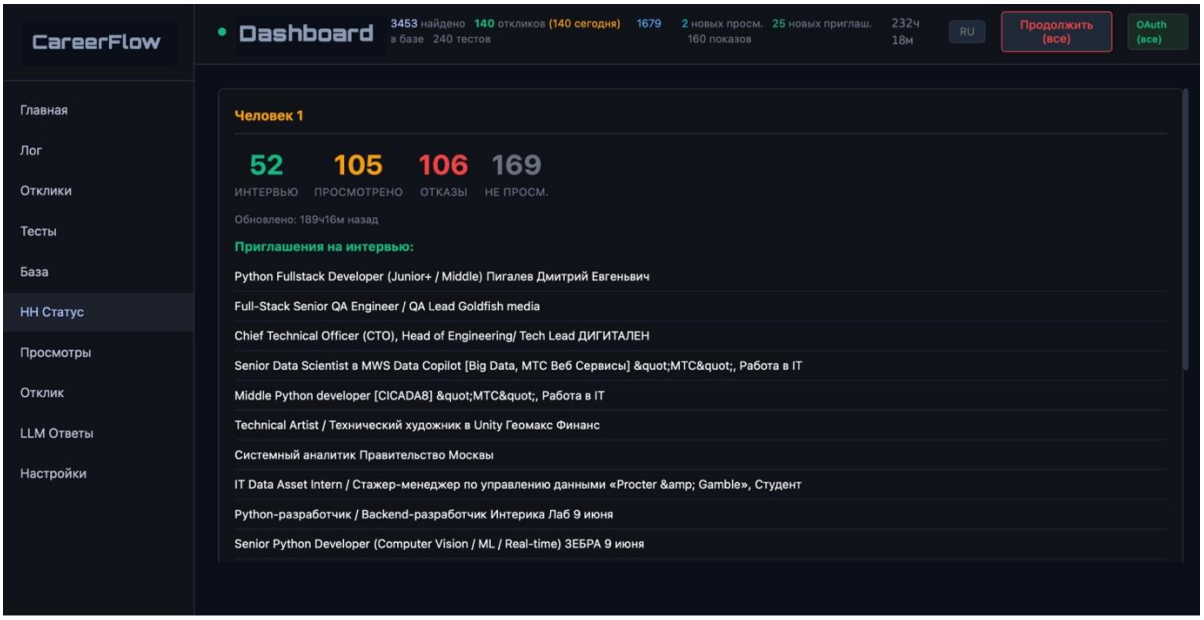


Рисунок К.5 – Проверка статистики HH

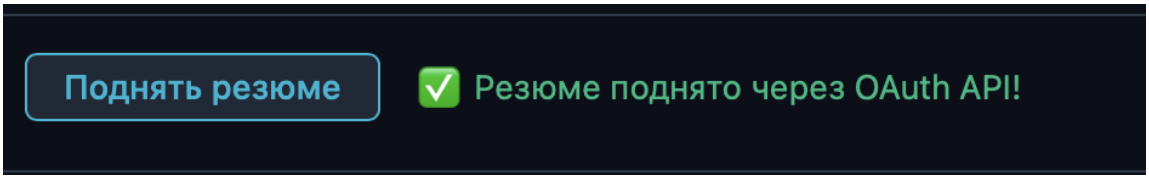


Рисунок К.6 – Подъём резюме в поиске

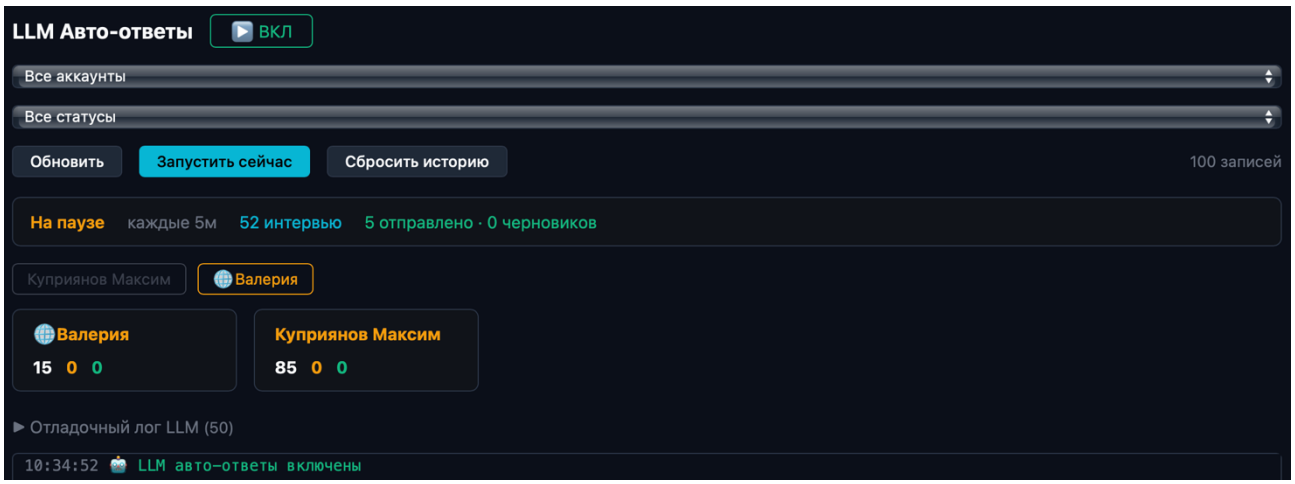


Рисунок К.7 – Включение LLM автоответов