МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»

Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой ЭВМ
_____ Д.В. Топольский
«___»_____ 2025 г.

# Создание цифровой платформы кадровых ресурсов

## ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
## К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
## ЮУРГУ-090301.2025.405 ПЗ ВКР

Руководитель работы,
к. т. н., доцент каф. ЭВМ
_____ Д.В. Топольский
«___»_____ 2025 г.

Автор работы,
студент группы КЭ-405
_____ В.Д. Палкин
«___»_____ 2025 г.

Нормоконтролёр,
ст. преподаватель каф. ЭВМ
_____ С.В. Сяськов
«___»_____ 2025 г.

Челябинск-2025

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

УТВЕРЖДАЮ
Заведующий кафедрой ЭВМ
_____ Д.В. Топольский
«___»_____2025 г.

# ЗАДАНИЕ

**на выпускную квалификационную работу бакалавра**

студенту группы КЭ-405
Палкину Василию Дмитриевичу
обучающемуся по направлению
09.03.01 «Информатика и вычислительная техника»

1. **Тема работы:** «Создание цифровой платформы кадровых ресурсов» утверждена приказом по университету от «21» апреля 2025 г. №648-13/12 (приложение №9).

2. **Срок сдачи студентом законченной работы:** 1 июня 2025 г.

3. **Исходные данные к работе:**

В рамках настоящей работы требуется разработать цифровую HR-платформу на основе фреймворка Django и системы управления базами данных PostgreSQL. Платформа должна включать следующие функциональные модули: управление персоналом, подбор сотрудников, организация обучения и документооборот. Для анализа кадровых данных необходимо предусмотреть использование методов машинного обучения с применением библиотеки Scikit-learn. В процессе разработки

предполагается использование технологий Python, Django, PostgreSQL, HTML, CSS, JavaScript и Scikit-learn.

4. **Перечень подлежащих разработке вопросов:**

1. Аналитический обзор литературы.
2. Проектирование архитектуры цифровой платформы кадровых ресурсов.
3. Разработка цифровой платформы кадровых ресурсов.
4. Проведение тестирования программного модуля.

5. **Дата выдачи задания:** 2 декабря 2024 г.

Руководитель работы _____/ *Д. В. Топольский* /

Студент _____/ *В. Д. Палкин* /

КАЛЕНДАРНЫЙ ПЛАН

| Этап | Срок сдачи | Подпись руководителя |
|---|---|---|
| Аналитический обзор литературы | 03.03.2025 | |
| Проектирование архитектуры цифровой платформы кадровых ресурсов | 22.03.2025 | |
| Разработка программного модуля цифровой платформы кадровых ресурсов | 12.04.2025 | |
| Проведение тестирования программного модуля | 26.04.2025 | |
| Компоновка текста работы и сдача на нормоконтроль | 22.05.2025 | |
| Подготовка презентации и доклада | 30.05.2025 | |

Руководитель работы _____/ *Д. В. Топольский* /

Студент _____/ *В. Д. Палкин* /

Аннотация

В.Д. Палкин. Создание цифровой платформы кадровых ресурсов. – Челябинск: ФГАОУ ВО «ЮУрГУ (НИУ)», ВШ ЭКН; 2025, 142 с., 22 ил., библиогр. список – 35 наим.

В работе выполнен аналитический обзор современной литературы и практических решений, посвящённых цифровизации HR-процессов, People Analytics и электронному документообороту. На основе сравнительного анализа отечественных и зарубежных систем обоснован выбор стека Python / Django / PostgreSQL и предложена архитектура веб-платформы, объединяющей управление персоналом, подбор сотрудников, машинное обучение и работы с документами.

В ходе исследования сформулированы требования к функционалу, спроектирована реляционная модель данных, разработан прототип с ролями администратора, HR-менеджера, сотрудника и кандидата. Платформа обеспечивает централизованное хранение кадровых данных, автоматизацию маршрутов согласования документов, аналитику текучести и ранжирование кандидатов, что позволяет сократить время найма, снизить ошибочность операций и уменьшить затраты на бумажный документооборот.

Результаты соответствуют актуальным направлениям развития информационных технологий в сфере управления человеческими ресурсами и повышения цифровой зрелости предприятий. Предложенное решение может служить основой для дальнейшего расширения функциональности и внедрения в организациях малого и среднего масштаба, требующих гибкой и экономичной HR-платформы.

# ОГЛАВЛЕНИЕ

# ВВЕДЕНИЕ

Стремительные изменения на рынке труда, широкое распространение удалённых форм занятости и необходимость оптимизации внутренних процессов делают цифровую трансформацию HR-служб критически важной для любой современной организации. Традиционные методы управления персоналом, предполагающие ручной учёт и бумажный документооборот, становятся малопригодными в условиях цифровой экономики. Поэтому всё больше компаний ищут эффективные инструменты автоматизации, способные централизованно обрабатывать большие объёмы данных о сотрудниках и вакансиях, а также упрощать взаиморасчёты и процесс принятия управленческих решений.

Актуальность темы данной дипломной работы определяется потребностью в создании гибкой и доступной платформы, оптимизирующей базовые HR-процессы: подбор персонала, хранение данных о сотрудниках, ведение электронного документооборота и взаимодействие с кандидатами. Существующие комплексные решения (SAP SuccessFactors, Oracle HCM, 1С:ЗУП и др.) нередко требуют крупных инвестиций и сложны в интеграции, что делает их не всегда целесообразными для организаций со средним штатом и ограниченными ресурсами.

Цель работы – разработать прототип веб-приложения «Цифровая платформа для кадровых ресурсов», демонстрирующий ключевые возможности автоматизации HR-функций:

- регистрацию пользователей и управление ролями (администратор, HR-менеджер, сотрудник);
- хранение и обновление цифровых профилей персонала;
- управление вакансиями, откликами и проведением собеседований;
- возможность расширения за счёт интеграции электронного документооборота и обучения сотрудников.

Практическая значимость состоит в том, что прототип может служить основой для создания внутрикорпоративных HR-платформ небольшого и среднего масштаба. Автоматизация позволяет упростить рутинные операции (оформление документов, обработку заявок на вакансии, хранение досье сотрудников), сократить временные затраты и минимизировать ошибки, обусловленные человеческим фактором. Система способствует повышению прозрачности и эффективности HR-процессов, а также сокращению расходов на бумажный документооборот.

Структура работы включает введение, где обосновывается актуальность исследования и ставятся основные задачи, главу с анализом предметной области и описанием методологии IDEF, раздел о проектировании и реализации прототипа, заключение и список литературы. Данная последовательность призвана продемонстрировать логику исследования: от теоретических основ и моделирования процессов до практической проверки и итоговой оценки разработанного решения.

# 1. АНАЛИТИЧЕСКИЙ ОБЗОР ЛИТЕРАТУРЫ

В России с 2017 года действует «Стратегия развития информационного общества на 2017-2030 гг.» и одноимённая нацпрограмма «Цифровая экономика» (этап II – 2025 г.) – они задали ориентир на тотальную цифровизацию процессов, перенос документооборота в электронную среду и внедрение ИИ в бизнес-практики. К концу 2024 г. 93 % домохозяйств подключены к широкополосному интернету, а бюджет исполнения нацпроекта достиг 99,7% плановых назначений – 145,7 млрд руб. Правительство дополнило эти инициативы отраслевыми «Стратегиями цифровой трансформации» (промышленность, культура и др.) и «Стратегией научно-технологического развития – 2030», которые обязывают предприятия переходить к data-driven подходу и локальным ИТ-решениям.

Государственный курс формирует острую потребность в отечественных HR-инструментах, которые:

- полностью хранят персональные данные внутри РФ (152-ФЗ);
- интегрируются с российскими госсервисами (ЕСИА, КЭДО);
- содержат People Analytics/ИИ-модули, снижая зависимость от западных SaaS.

Создание собственной платформы отвечает этим требованиям и вписывается в приоритеты «цифровой зрелости» кадровых процессов по методологии Минцифры.

## 1.1. Исторические и современные тенденции цифровизации HR

Стремительная эволюция информационных технологий радикально изменила подходы к управлению человеческими ресурсами ещё в конце XX века. Переход от классических бумажных картотек к электронным

таблицам ознаменовал I цифровую волну в HR сфере. Со второй половины 2000 х годов началось формирование полноценных систем Human Resource Management System (HRMS) и Human Capital Management (HCM), поддерживающих многоуровневые процессы: от учёта кадрового состава до расчёта заработной платы. Третья волна (2010) привнесла облачные решения Software as a Service (SaaS), позволившие компаниям быстро масштабировать сервисы и платить «по подписке» без капитальных вложений в инфраструктуру.

Сегодня на пороге четвёртой волны цифровизации HR определяющими факторами являются большие данные, машинное обучение и демократизация разработческих инструментов. Ключевые тенденции сводятся к:

- переходу от учётных систем к экосистемам взаимодействия сотрудников и руководителей, ориентированным на пользовательский опыт (Employee Experience);
- интеграции искусственного интеллекта для автоматизированного подбора персонала, прогнозирования текучести кадров и формирования карьерных рекомендаций;
- широкому распространению удалённой работы и, как следствие, необходимости надёжных онлайн средств для электронного документооборота и дистанционного обучения;
- концепции People Analytics, когда HR служба становится дата ориентированным подразделением, способным принимать решения на основании количественных моделей, а не интуиции.

Историческая динамика показывает: каждая новая технологическая итерация не обнуляет, а расширяет предыдущие слои автоматизации, образуя многослойные цифровые платформы, где ядро учёта кадров дополняется сервисами аналитики, обучения и вовлечённости.

## 1.2. Анализ функционала и возможностей существующих цифровых HR систем

Функциональный срез рынка 2020 х демонстрирует консолидацию четырёх базовых доменов: ядро HRIS/HRMS, рекрутинг, обучение и документооборот. В наиболее зрелых продуктах (SAP SuccessFactors, Oracle HCM, Workday) эти домены тесно интегрированы, образуя единую экосистему. В то же время на рынке присутствуют нишевые игроки, решающие отдельные задачи с помощью узкоспециализированных платформ (Skillaz – рекрутинг с ИИ подбором, HRlink – кадровый электронный документооборот). Сравнение существующих решений, разбитое по доменам, приведено в таблице 1.

Таблица 1 – Сравнение текущих существующих на рынке систем

| **Домен** | **Ключевые модули** | **Примеры систем** | **Ценность для бизнеса** |
|---|---|---|---|
| Учёт персонала (Core HR) | Личные карточки, движения кадров, расчёт стажа | 1С:ЗУП, БИТ.ФИНАНС HR | Централизованные и точные данные о составе штата |
| Подбор и адаптация (Talent Acquisition) | ATS, карьерные сайты, HR-чат-боты | Skillaz, Huntflow, Greenhouse | Ускорение найма, снижение стоимости закрытия вакансий |
| Обучение и развитие (LMS/TMS) | Каталоги курсов, трекинг прохождения, геймификация | iSpring Learn, Mirapolis LMS | Поддержка непрерывного обучения, развитие компетенций |

Продолжение таблицы 1

| | | | |
|---|---|---|---|
| Электронный документооборот (КЭДО) | Приказы, заявления, ЭЦП, маршруты согласования | HRlink, Диадок, Контур.Кадры | Сокращение бумажных затрат, юридическая значимость документов |
| HR-аналитика | Дашборды, People Analytics, прогнозы текучести | SAP Analytics Cloud, Oracle OAC | Принятие решений на основе данных, оценка ROI HR-инициатив |

Интегральным критерием эффективности становится бесшовность между модулями: единый интерфейс, сквозная авторизация, общая модель данных. Компании нередко комбинируют несколько систем, связывая их через REST API или микросервисную шину событий.

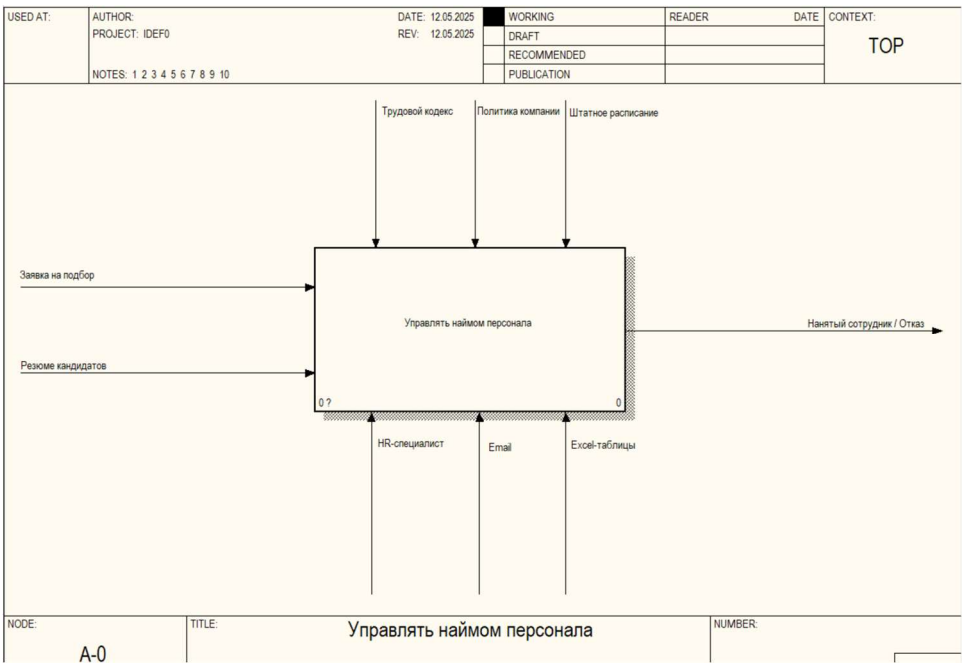



Рисунок 1 – IDEF0-диаграмма текущего процесса подбора персонала (A-0, AS-IS).

Диаграмма создана при помощи пакета программного обеспечения Ramus

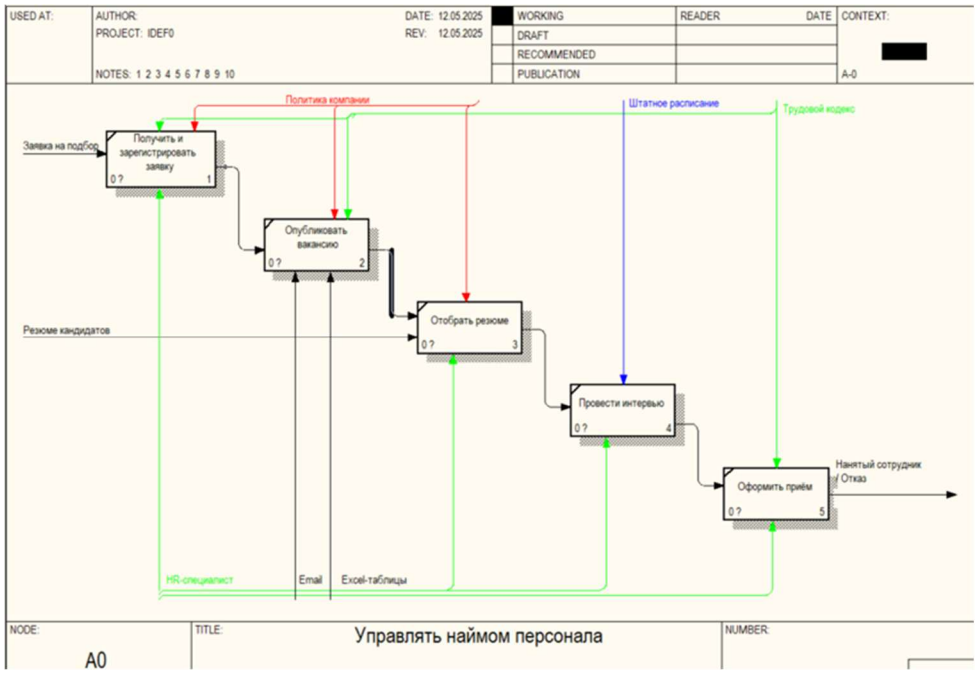



Рисунок 2 – Декомпозиция IDEF0-диаграммы текущего процесса подбора персонала (узел A0, AS-IS).

Диаграмма создана при помощи пакета программного обеспечения Ramus

### 1.3. Анализ функционала и возможностей существующих цифровых HR систем

Используя массивы данных об откликах, оценках эффективности, зарплатных метриках и вовлечённости, человеко-центричные алгоритмы строят предиктивные модели:

- прогноз текучести (Churn Prediction) – алгоритмы градиентного бустинга или XGBoost оценивают вероятность увольнения сотрудника в горизонте 3-6 месяцев, учитывая историю повышения, отпусков, результаты перформанс ревью;
- ранжирование кандидатов (Resume Scoring) – модели трансформеров, обученные на множестве вакансий, автоматически сопоставляют компетенции соискателя с требованиями должности;

- персонализированное обучение (Adaptive Learning) – системы рекомендаций, аналогичные Netflix, предлагают сотруднику курсы, усиливающие именно его дефицитные навыки [35];
- оптимизация графиков (Workforce Scheduling) – стохастические методы минимизируют излишнее время простоя персонала в розничных сетях и колл-центрах.

Внедрение People Analytics трансформирует HR подразделение из сервисного офиса в центр стратегической экспертизы, а также требует новых компетенций: аналитики данных, визуализации и интерпретации ML результатов.

Такие расширенные инструменты позволяют HR-команде оперативно реагировать на проблемы, выстраивать проактивные коммуникации и стратегически управлять талантами, снижая операционные риски и повышая удовлетворённость персонала.

## 1.4. Сравнительный анализ существующих HR-решений

В таблицах 2 и 3 представлены сравнения отечественных и зарубежных HR-решений.

Таблица 2 – Сравнительный анализ HR-решений по стране происхождения, функционалу и дополнительным возможностям

| Платформа | Страна происхождения | Основной функционал | Дополнительные возможности |
|---|---|---|---|
| SAP SuccessFactors | Германия, США | Core HR, рекрутинг, обучение | People Analytics, Succession Planning |
| Workday | США | HCM, планирование, финансовый модуль | Adaptive Planning, Prism |
| 1С:ЗУП | Россия | Учёт кадров, зарплата | КЭДО, отчётность ФНС |
| Skillaz | Россия | ИИ-подбор, ATS | Адаптация, онбординг |
| iSpring Learn | Россия | LMS | Геймификация, готовая библиотека |
| Mirapolis HCM | Россия | HCM, LMS, KPI | Портал, тесты |

Таблица 3 – Сравнительный анализ HR-решений по преимуществам, ограничениям, локализации и комплаентности с местным законодательством

| Платформа | Преимущества | Ограничения | Локали-зация | Соот-ветс-твие 152-ФЗ |
|---|---|---|---|---|
| SAP SuccessFactors | Комплексность, лучшая аналитика | Высокая стоимость, сложность внедрения | Русский язык (частично переведено) | – |
| Workday | SaaS-масштабируе-мость, UX | Нет локальных дата-центров | Английский язык | – |
| 1С:ЗУП | Глубокая интеграция с бухгалтерией | Устаревший UI, слабый рекрутинг | Русский язык | + |
| Skillaz | ML-алгоритмы ликвидируют ручной поиск | Узкая ниша, нет расчёта ЗП | Русский язык | + |
| iSpring Learn | Быстрый запуск, SCORM | Раздельная оплата модулей | Русский язык | + |
| Mirapolis HCM | Модульность, масштабируемо сть | Отсутствует мобильное приложение | Русский язык | + |

Сравнение показывает: иностранные решения выигрывают в зрелости People Analytics, но требуют локализации хранилищ данных и значительных затрат на соответствие отечественному законодательству. Российские

продукты быстрее адаптируются к регуляторным нормам, но зачастую уступают в UX и глубине интеграции ML-сервисов.

## 1.5. Выбор технологического стека для разработки собственной HR-платформы

Опираясь на выводы предыдущих пунктов, был выбран следующий стек технологий для разработки нашей HR-платформы.

Серверная часть (backend):

- Python (Django) - высокоуровневый фреймворк с богатой экосистемой и встроенной ORM;
- Django REST Framework - для реализации API-интерфейсов;
- PostgreSQL - надежная и масштабируемая СУБД с поддержкой сложных типов данных;
- pandas, scikit-learn - инструменты для аналитики и машинного обучения.

Клиентская часть (frontend):

- HTML5, CSS3, JavaScript – стандартные технологии веб-разработки;
- Bootstrap – фреймворк для создания адаптивного и современного интерфейса.

Такой выбор обеспечивает высокую скорость прототипирования и разработки, готовую основу для People Analytics (pandas, scikit-learn), масштабируемость решения и хорошие характеристики безопасности (XSS/CSRF защита, RBAC, JWT-аутентификация).

Схема архитектуры программного решения представлена на рисунке 3.

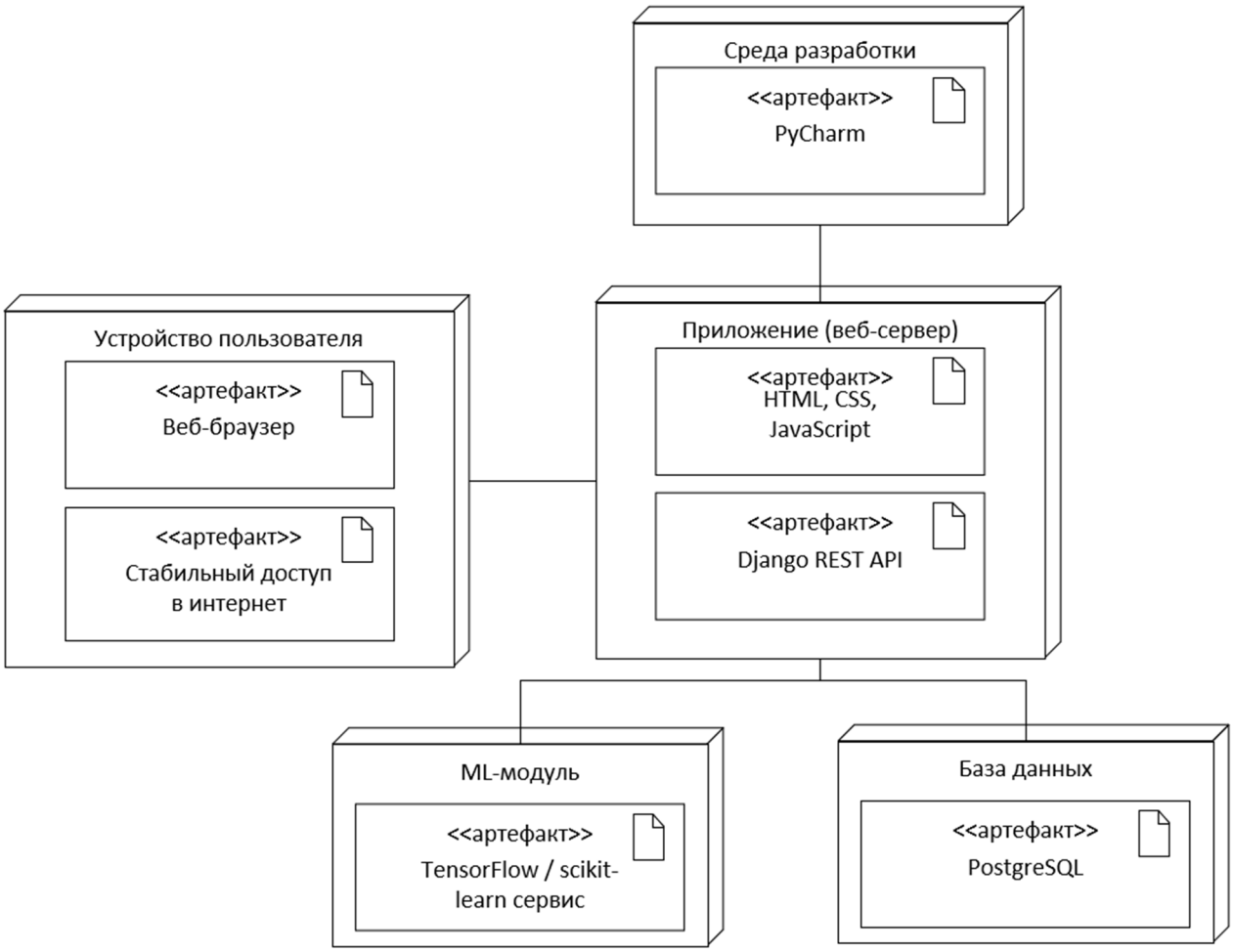



Рисунок 3 – UML диаграмма архитектуры программного решения

## 1.6. Выводы по итогам обзора литературы

В первой главе систематизирована эволюция цифровых HR решений, проанализированы ключевые функциональные домены и рассмотрено влияние People Analytics на трансформацию роли кадровой службы. Сравнительный обзор показал преимущества и ограничения отечественных и зарубежных платформ, а также подтвердил жизнеспособность комбинации Django/React технологий для разработки гибкой HR системы. Сформулирована технологическая основа, отвечающая современным трендам цифровизации HR процессов.

# 2. ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ЦИФРОВОЙ ПЛАТФОРМЫ КАДРОВЫХ РЕСУРСОВ

В данном разделе рассматриваются этапы проектирования цифровой HR-платформы, начиная с постановки требований и заканчивая выбором технологического стека и описанием ключевых компонентов системы. Особое внимание уделяется структурированному подходу, который позволяет учесть все особенности будущего веб-приложения и заложить основу для дальнейшего развития.

## 2.1. Требования к платформе

Формирование требований имеет ключевое значение при разработке любой информационной системы, так как оно помогает четко определить цели проекта и установить границы функционала. Убедительно прописанные требования минимизируют риски недоразумений между заказчиком и разработчиками на всех стадиях создания и внедрения приложения. Важно учитывать потенциальные варианты использования HR-платформы (они приведены на рисунке 4.

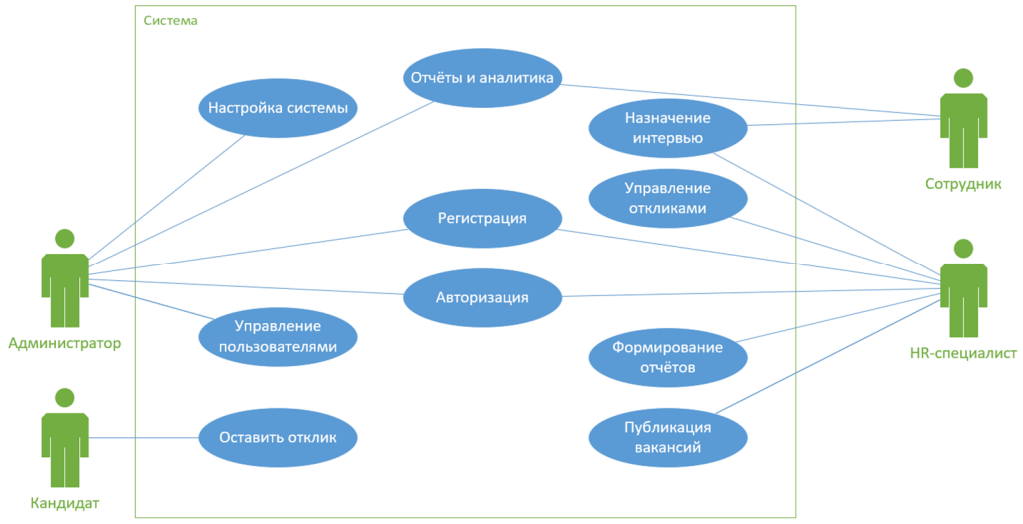



Рисунок 4 – UML диаграмма вариантов использования HR-платформы

Что касается функциональных требований, то необходимо предусмотреть создание учётных записей с разграничением ролей, таких как администратор, HR-менеджер, сотрудник и кандидат. Также важна реализация механизма сброса пароля и двухфакторной аутентификации (опционально). Управление профилями сотрудников должно включать возможность выполнять CRUD-операции с записями о сотрудниках, учитывая историю изменений должности, отдела и других параметров. Помимо этого, требуется функционал для фильтрации, поиска и сортировки информации по таким критериям, как отдел, должность, навыки, стаж и т. д. Пример интерфейса, на котором реализован такой функционал, приведён на рисунке 5.

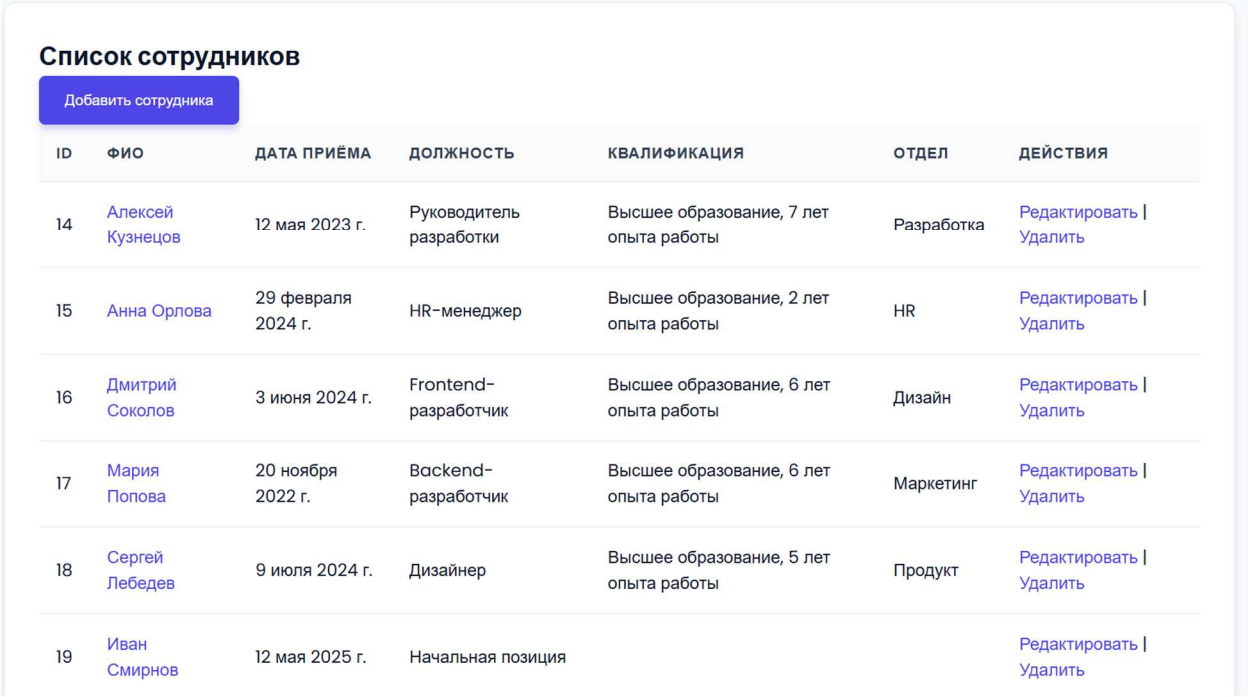



Рисунок 5 – Экранная форма, демонстрирующая список сотрудников в рамках HR-платформы

Важно предоставить возможность загрузки и хранения документов, включая сканы удостоверений, рекомендаций и сертификатов.

По части управления вакансиями, необходимо предоставить возможность создавать и редактировать карточки вакансий, указав

требования, условия труда и сроки. Также важно предусмотреть функционал для подачи откликов через личный кабинет кандидата или внешнюю форму, хранение резюме и ведение статусов отклика, таких как новый отклик, собеседование, тестовое задание, отказ или оффер. Дополнительно, должна быть возможность прикрепления комментариев и результатов тестов или собеседований.

Документооборот должен включать электронное оформление кадровых документов, таких как приказы о приёме, переводе и увольнении, а также заявления на отпуск и другие. Важно также настроить маршруты согласования с участием руководителей, бухгалтерии и юристов. При этом необходимо обеспечить безопасное хранение электронных версий документов и логирование действий.

Что касается аналитики и отчётов, приложение должно формировать отчёты о текучести кадров, динамике найма и статистике выполнения KPI. Необходимо также обеспечить графическую визуализацию данных для удобства восприятия, с использованием диаграмм и графиков. Экспорт отчётных данных в популярные форматы, такие как Excel и PDF, также является важным требованием.

Интеграция с внешними сервисами предполагает возможность обмена данными с ERP или CRM через REST/API, включая информацию о сотрудниках, доступ к календарям и финансовым показателям. Также должно быть предусмотрено возможность публикации вакансий на сторонних job-порталах непосредственно из платформы.

Нефункциональные требования включают требования к надёжности и масштабируемости системы. Приложение должно корректно работать при пиковых нагрузках, когда одновременно работают несколько сотен пользователей. Поддержка горизонтального или вертикального масштабирования на уровне веб-сервера, кэширования и балансировки нагрузки также является необходимостью.

Безопасность приложения должна соответствовать требованиям защиты персональных данных, таким как GDPR и ФЗ-152. Шифрование передаваемой информации по протоколу HTTPS (SSL/TLS) также должно быть реализовано. Настройка ролей и разрешений должна учитывать принципы минимально необходимого доступа, реализуя RBAC.

Для удобства использования интерфейс приложения должен быть адаптивным и работать в распространённых браузерах и на мобильных устройствах. Навигация должна быть интуитивно понятной, включая такие разделы, как главная панель, личный кабинет и отчёты. Также важно предусмотреть возможность локализации интерфейса, с поддержкой нескольких языков.

Приложение должно обеспечивать стабильную работу при больших объёмах данных, до нескольких сотен тысяч записей в базе. Среднее время отклика сервера должно быть не более 2–3 секунд при штатной нагрузке.

Важной частью системы должна быть интегрируемость, для чего потребуется наличие API для подключения к другим системам компании и внешним веб-сервисам. Также необходимо предусмотреть стандартизированные методы авторизации для сторонних модулей, такие как OAuth 2.0 и JWT.

Поддерживаемость приложения требует, чтобы код был структурирован понятным образом и документирован на уровне модулей с помощью docstrings и ReadMe. Также должны быть предусмотрены инструменты для логирования и мониторинга, такие как Sentry или ELK Stack.

## 2.2. Выбор инструментов разработки

При проектировании системы важно учитывать современные технологии, которые обеспечивают надёжность, удобство разработки и дальнейшую модульную расширяемость. Серверная часть приложения будет

реализована с использованием языка программирования Python, который обладает широкой экосистемой библиотек, простотой интеграции с модулями аналитики и поддержкой машинного обучения. В качестве фреймворка для разработки будет использован Django, который благодаря встроенной ORM упрощает работу с данными и позволяет быстро генерировать административную часть. Для реализации RESTful API и возможных внешних интеграций будет применяться Django Rest Framework (DRF). В качестве системы управления базами данных (СУБД) выбрана PostgreSQL, известная своей поддержкой транзакций, индексов и JSON-типов данных, что обеспечивает высокую надёжность и производительность. Схема базы данных в рамках разрабатываемой HR-платформы приведена на рисунке 6.

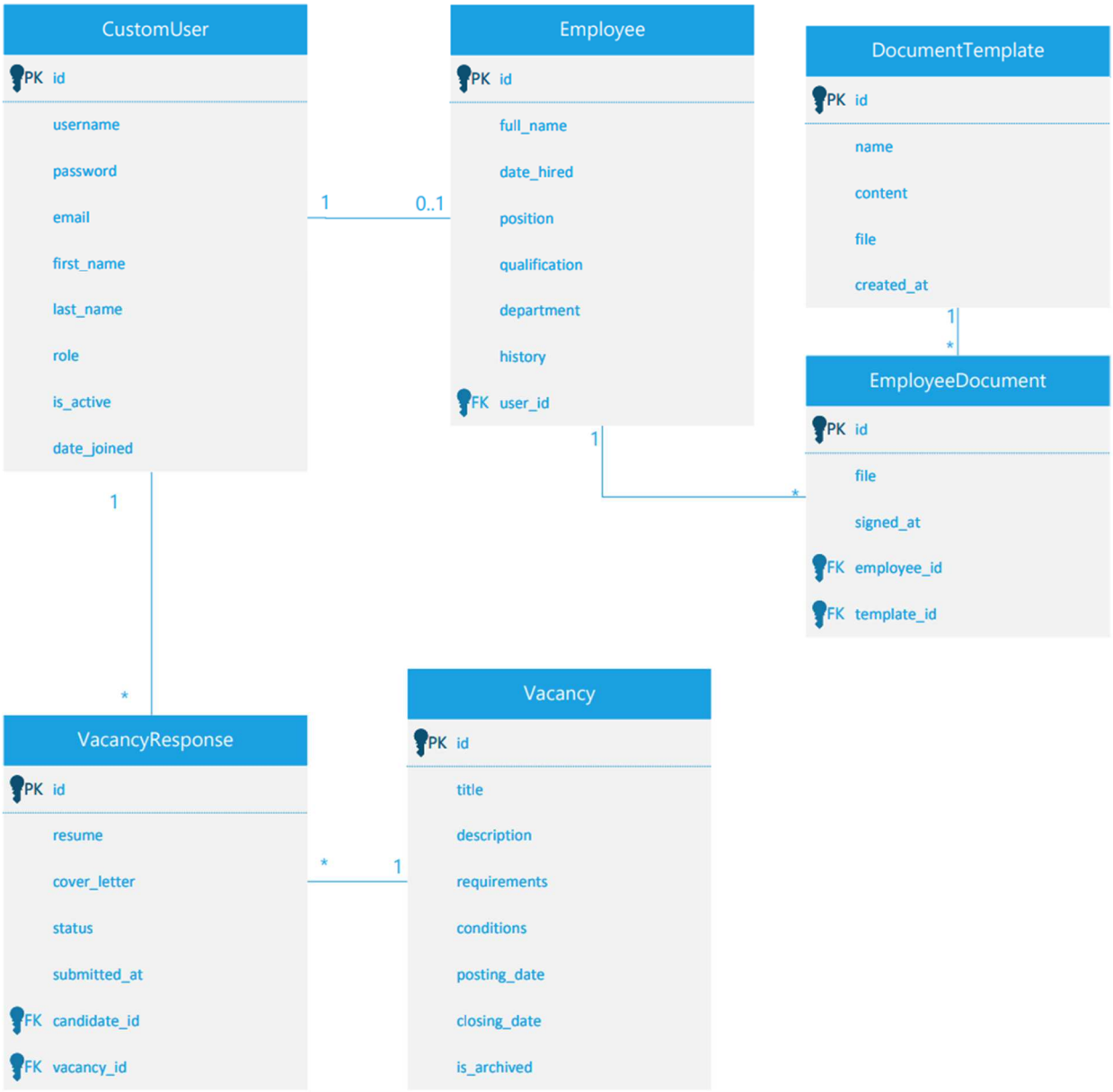



Рисунок 6 – Схема базы данных (изображение создано при помощи пакета программного обеспечения Microsoft Visio)

Для фронтенда используется стандартный стек технологий: HTML, CSS и JavaScript. Основным инструментом для создания пользовательского интерфейса станет фреймворк React или Vue.js, который позволит создавать интерактивные модули, панели управления и удобные формы для работы с документами, а также обеспечит легкость настройки маршрутизации и повторного использования компонентов. В качестве CSS-библиотек для

облегчения адаптивной верстки и использования готовых элементов интерфейса будут использоваться Bootstrap или Tailwind CSS.

Для организации командной работы и отслеживания изменений будет использоваться система управления версиями Git. Стандартизация окружения и упрощение процесса запуска системы будет обеспечена с помощью инструментов Docker и Docker Compose. Для аналитики будут применяться Python-библиотеки, такие как Pandas и Matplotlib, либо сторонние BI-платформы, например, Metabase или Grafana.

Что касается архитектуры приложения, то правильное проектирование упрощает внедрение нового функционала и поддержку существующего кода. В рассматриваемом проекте целесообразно выделить несколько основных подсистем, таких как подсистема управления пользователями и доступом, включающая разделение на роли (администратор, HR-менеджер, сотрудник, кандидат) и определение прав доступа. Механизмы аутентификации будут включать пароли и OAuth/JWT, с возможной интеграцией с LDAP или Active Directory. Подсистема профилей и кадровых данных будет обеспечивать хранение и редактирование сведений о сотрудниках, их перемещении между отделами и навыках, а также логирование изменений. Подсистема вакансий и подбора кадров позволит создавать и редактировать карточки вакансий, регистрировать отклики, хранить резюме и отслеживать статусы собеседований, а также оставлять комментарии HR-менеджерам и руководителям.

Подсистема электронного документооборота будет включать шаблоны кадровых документов, их заполнение, хранение и подписание, а также управление маршрутами согласования для различных типов документов. Интеграция с системой цифровых подписей будет возможна при необходимости. Модуль аналитики и отчётности будет отображать сводные показатели по вакансиям, динамике текучести кадров и структуре штата через дашборды, с возможностью экспорта данных в Excel и PDF, а также хранения

готовых отчётов в базе. Визуализация данных может быть реализована как встроенными средствами, так и с помощью внешних платформ.

Проектирование модуля управления пользователями будет включать структуру таблиц, таких как User, Role, Permission. Регистрация пользователей будет осуществляться через короткую форму с опциональной верификацией электронной почты, а аутентификация будет обеспечиваться с использованием безопасных хэш-функций, таких как bcrypt или Argon2. В модуле управления вакансиями будет создана таблица Vacancy, содержащая ключевые поля, такие как название должности, требования, статус и дата создания. Таблица Application будет обеспечивать связь резюме соискателей с вакансиями и статусы их обработки. Интерфейс HR-менеджера позволит просматривать список вакансий с фильтрами по отделам, должностям и навыкам.

Модуль документооборота будет включать классы и таблицы для хранения документов, таких как Document и DocumentType, а также настройку маршрутов согласования. Интеграция с провайдером электронных подписей будет предусмотрена при необходимости. В модуле аналитики будут рассчитываться основные метрики, такие как текучесть кадров, время закрытия вакансий, средняя зарплата по отделам и динамика найма. Интерфейс отчётов обеспечит возможность выбора периода, формата диаграммы и экспорта данных. Дополнительные возможности, такие как расчёт прогноза текучести кадров, будут возможны при добавлении ML-модулей.

## 2.3. Построение архитектуры приложения. Проектирование компонентов и связей между ними

Грамотная архитектура приложения играет ключевую роль в упрощении внедрения нового функционала и поддержке уже существующего кода. В

рассматриваемом проекте целесообразно выделить несколько основных подсистем, каждая из которых будет выполнять определённую роль в общей системе.

Подсистема управления пользователями и доступом будет отвечать за разделение на роли, такие как администратор, HR-менеджер, сотрудник и кандидат, а также за определение прав и уровней доступа для каждой роли. Важной частью этой подсистемы станут механизмы аутентификации, включая использование паролей, а также OAuth и JWT, с возможностью интеграции с LDAP или Active Directory для управления доступом и аутентификацией пользователей.

Подсистема профилей и кадровых данных будет обеспечивать хранение и редактирование информации о сотрудниках, включая данные о их перемещении между отделами и о навыках. Также будет реализовано логирование изменений данных, позволяя отслеживать, кто, когда и какую информацию обновил.

Подсистема вакансий и подбора кадров будет включать создание и редактирование карточек вакансий, с возможностью фиксации требований, а также регистрацию откликов на вакансии, хранение резюме и отслеживание статусов собеседований. Дополнительно, пользователи смогут оставлять комментарии для HR-менеджеров и руководителей.

Подсистема электронного документооборота будет включать шаблоны для различных кадровых документов, таких как приказы, заявления и другие, с функциями их заполнения, хранения и подписания. Также будет предусмотрено управление маршрутами согласования для разных типов документов и интеграция с системой цифровой подписи, если это потребуется.

Модуль аналитики и отчётности будет представлять информацию в виде дашбордов, отображающих сводные показатели по вакансиям, динамику текучести кадров и структуру штата. Также будет реализована возможность экспорта данных в форматы Excel и PDF, а отчёты будут храниться в базе

данных для дальнейшего использования. Визуализация данных будет осуществляться как встроенными средствами, так и с помощью внешних инструментов.

Целевую модель процесса цифрового найма можно увидеть на рисунке 7, а детальная декомпозиция этого процесса представлена на рисунке 8.

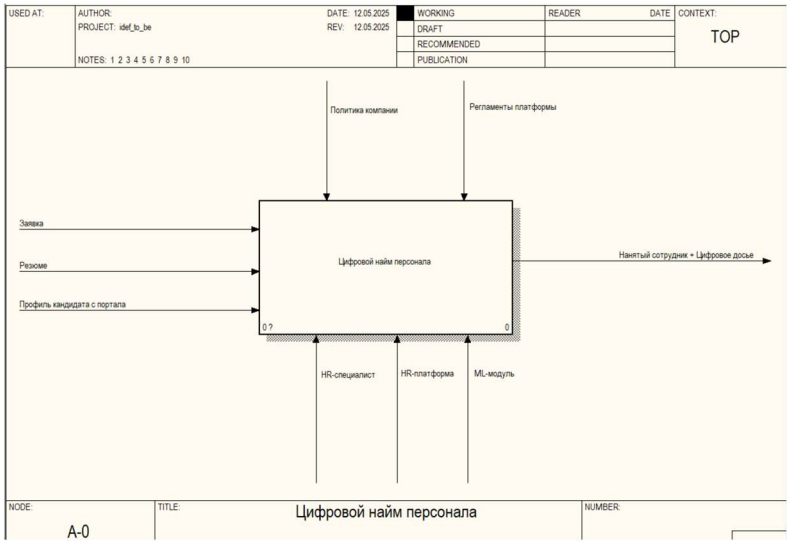



Рисунок 7 – IDEF0-диаграмма целевого процесса цифрового найма персонала (A-0, TO-BE).

Диаграмма создана при помощи пакета программного обеспечения Ramus

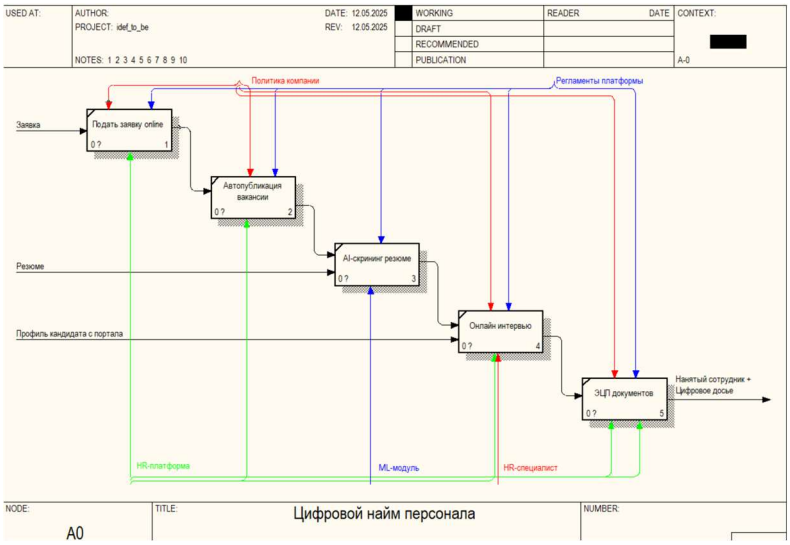



Рисунок 8 – Декомпозиция IDEF0-диаграммы целевого процесса цифрового найма (узел A0, TO-BE)

Диаграмма создана при помощи пакета программного обеспечения Ramus

Проектирование модуля управления пользователями включает в себя создание структуры таблиц, таких как User, Role и Permission. Регистрация пользователей будет осуществляться через короткую форму с опциональной верификацией электронной почты. Для аутентификации паролей будут использоваться безопасные хэш-функции, такие как bcrypt или Argon2, что обеспечит высокий уровень безопасности данных.

Модуль управления вакансиями будет включать таблицу Vacancy, где будут храниться ключевые поля, такие как название должности, требования, статус вакансии и дата её создания. Важной частью этого модуля станет таблица Application, которая будет связывать резюме соискателей с вакансиями и отслеживать статусы обработки, такие как принятый отклик, отказ или статус собеседования. Интерфейс для HR-менеджеров будет позволять просматривать список вакансий с возможностью фильтрации по таким параметрам, как отдел, должность или навыки кандидатов.

Проектирование модуля документооборота включает создание классов и таблиц для хранения документов, таких как Document, DocumentType и ApprovalRoute. Модуль будет предусматривать настройку маршрутов согласования документов, начиная от отдела кадров и заканчивая руководителями. При необходимости будет реализована интеграция с системой цифровых подписей, что обеспечит безопасность и юридическую значимость кадровых документов.

Модуль аналитики будет собирать и отображать ключевые метрики, такие как текучесть кадров, время закрытия вакансий, среднюю зарплату по отделам и динамику найма. Интерфейс отчётов позволит пользователю выбрать период, формат диаграммы и экспортировать данные. В перспективе будет добавлена возможность расчёта прогноза текучести кадров, если в систему будут интегрированы модули машинного обучения.

## 2.4. Выводы по итогам проектирования архитектуры

Во втором разделе дипломной работы сформированы основные функциональные и нефункциональные требования к цифровой HR-платформе, позволяющей комплексно управлять персоналом и процессами найма. Рассмотрены современные инструменты разработки: Django и PostgreSQL для серверной части, React (или Vue.js) для фронтенда, а также описаны средства интеграции и аналитики.

На базе проведённого анализа выбрана многоуровневая архитектура, в рамках которой выделены несколько ключевых подсистем (управление пользователями, кадровыми профилями, вакансиями и откликами, электронный документооборот и аналитика). Подробно рассмотрены структуры таблиц, принципы маршрутизации документов и механизмы авторизации. Для наглядной иллюстрации структуры приложения и интерфейсов в тексте приведены соответствующие рисунки, демонстрирующие макеты страниц, блок-схемы и ER-диаграммы.

Заложенные архитектурные решения и выбранный технологический стек позволяют создать надёжную и гибкую платформу, способную масштабироваться и расширяться с учётом меняющихся потребностей в сфере управления персоналом.

# 3. РАЗРАБОТКА ПРОГРАММНОГО МОДУЛЯ ЦИФРОВОЙ ПЛАТФОРМЫ КАДРОВЫХ РЕСУРСОВ

## 3.1. Реализация функционала управления персоналом и найма

В данном разделе описывается реализация основных модулей HR-платформы, включая управление пользователями, сотрудниками, вакансиями и откликами. Разработка велась в соответствии с модульным подходом, где каждый функциональный блок представлен отдельным Django-приложением. Полный программный код указан в листинге В.1 приложения А.

Модуль управления пользователями реализован на основе Django-приложения accounts. Для расширения стандартной модели пользователя Django и добавления возможности ролевого доступа была создана модель CustomUser.

Реализация модели пользователя приведена в листинге А.1 настоящего приложения А.

Для защиты представлений был реализован декоратор контроля ролей. Его реализация приведена в листинге А.2 приложения А.

На рисунке 9 представлена форма регистрации нового пользователя, а. на рисунке 10 – форма авторизации в системе.

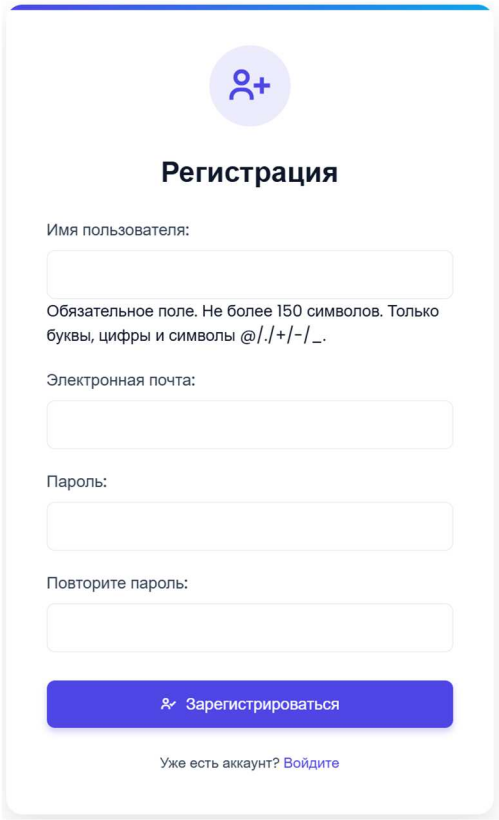



Рисунок 9 – Экранная форма регистрации нового пользователя

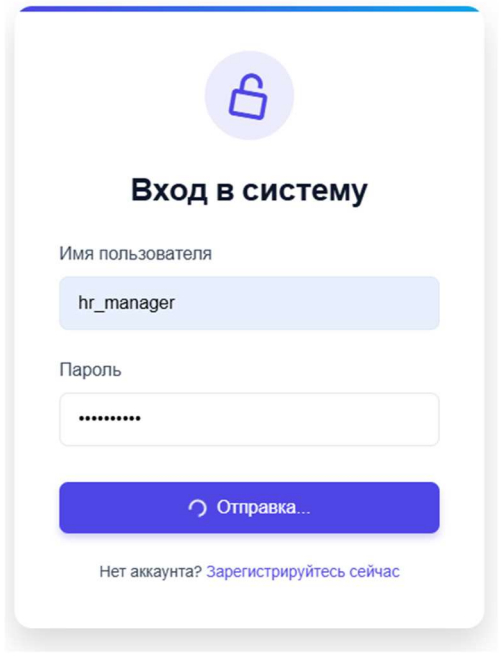



Рисунок 10 – Экранная форма, предназначенная для авторизации пользователя

На рисунке 11 представлена главная страница HR-платформы.

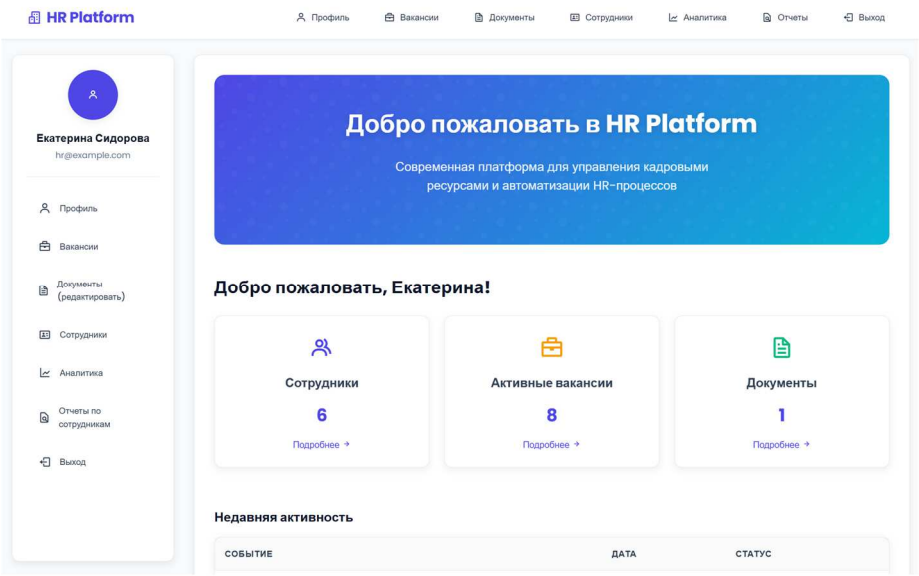



Рисунок 11 – Экранная форма, содержащая главную страницу HR-платформы

Модуль управления вакансиями реализован в приложении vacancies. Сопутствующие экранные формы приведены на рисунках 12 и 13. Основная модель вакансии приведена в листинге А.3 приложения А.

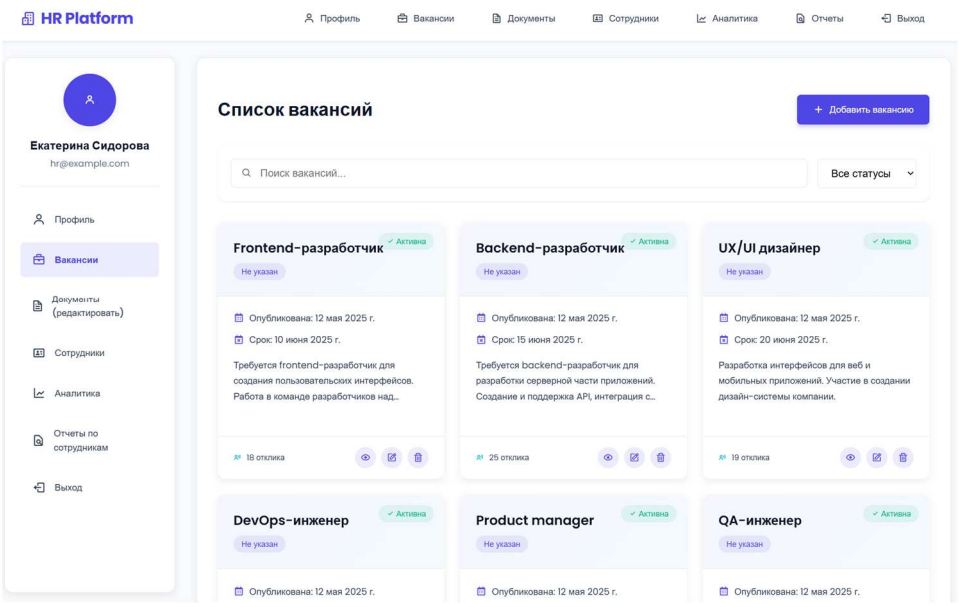



Рисунок 12 – Экранная форма модуля управления вакансиями

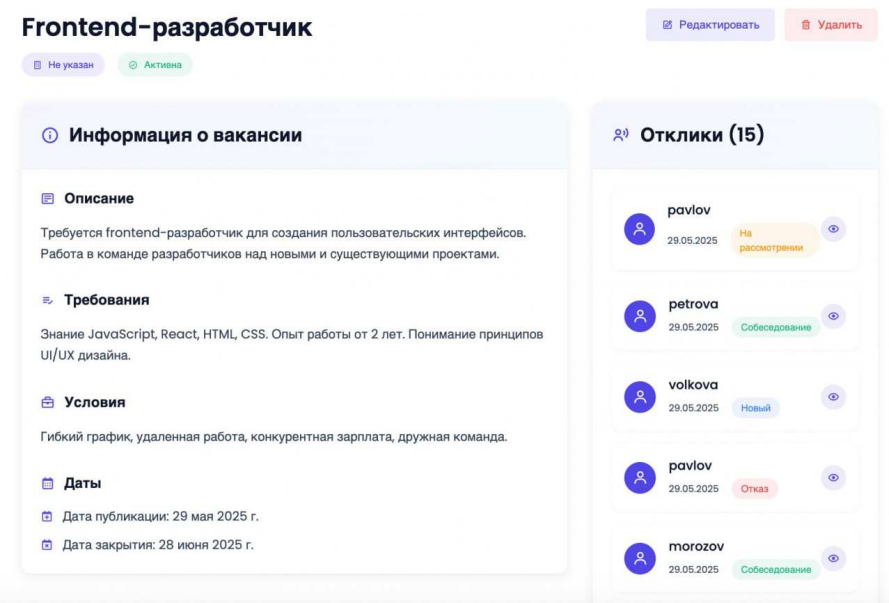



Рисунок 13 – Экранная форма, содержащая подробную информацию о вакансии

Модель откликов на вакансии приведена в листинге А.4 приложения А.

Сопутствующие этой модели экранные формы приведены на рисунках 14 и 15.

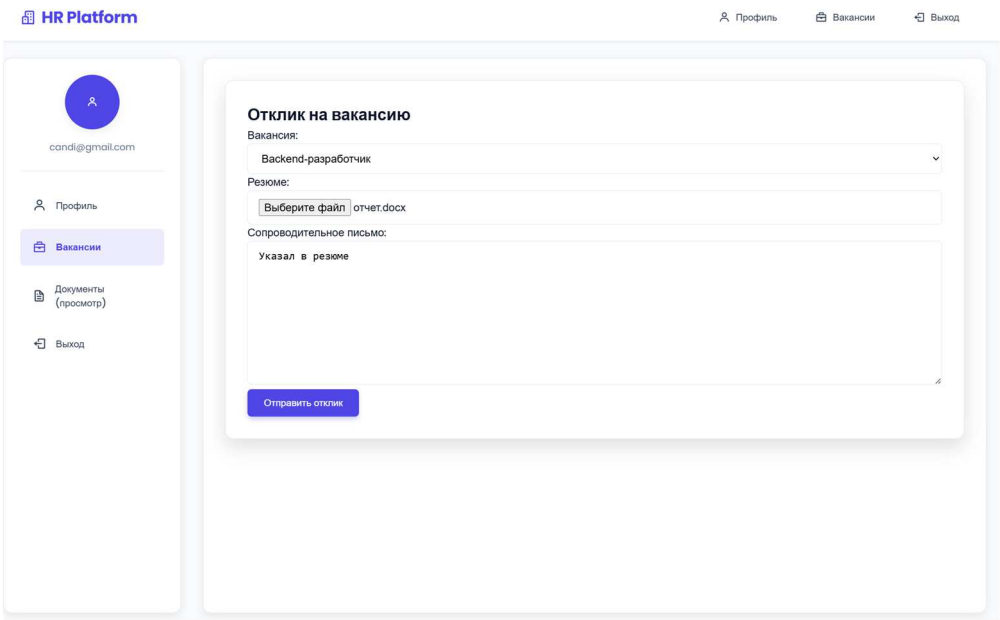



Рисунок 14 – Экранная форма отклика на вакансию с возможностью загрузки резюме

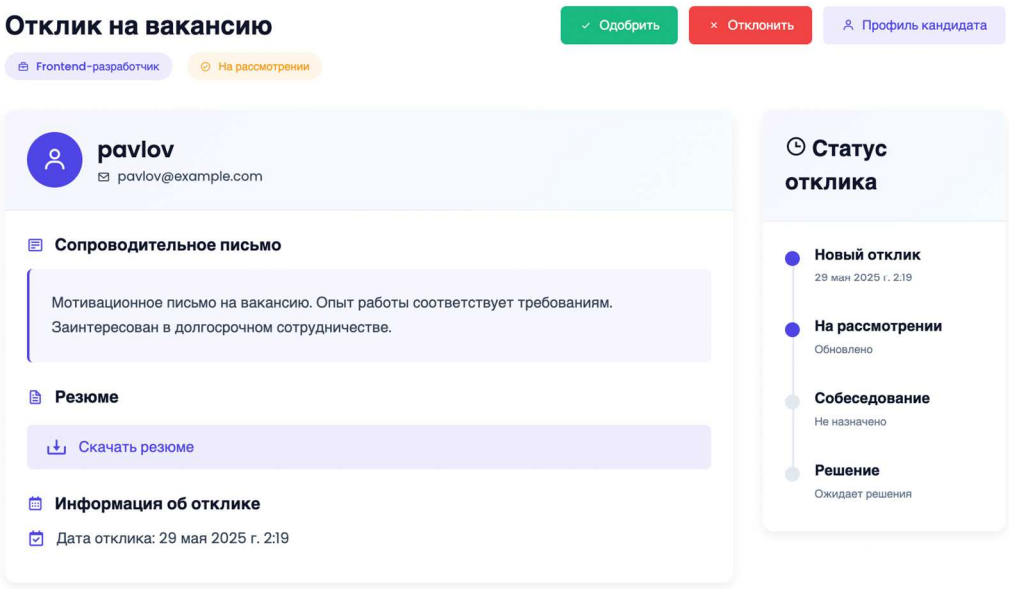



Рисунок 15 – Экранная форма управления откликами на вакансию

Представление для обработки откликов приведено в листинге А.5 приложения А.

## 3.2. Разработка пользовательского интерфейса и интерфейса панели администратора

При проектировании интерфейса HR-платформы были применены принципы удобства использования (usability) и адаптивного дизайна. Основные черты дизайна:

- простота и интуитивная понятность;
- минимизация количества действий для выполнения задач;
- адаптивность под различные устройства;
- визуальная согласованность всех элементов.

Для реализации этих черт использовался фреймворк Bootstrap, который обеспечивает единый стиль, адаптивную сетку и готовые компоненты для форм, таблиц и навигации. Полный программный код интерфейсов указан в листинге В.2 приложения А.

Разработанная экранная форма пользовательского профиля приведена на рисунке 16.

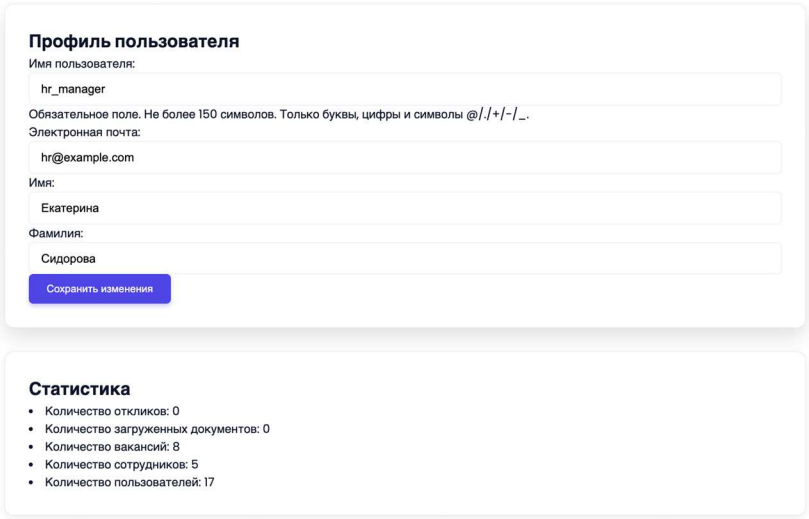



Рисунок 16 – Экранная форма пользовательского профиля с возможностью редактирования информации

Базовый шаблон содержит навигационное меню с ролевой фильтрацией. Он представлен в листинге А.6 приложения А.

Разработанная панель администратора приведена на рисунке 17.

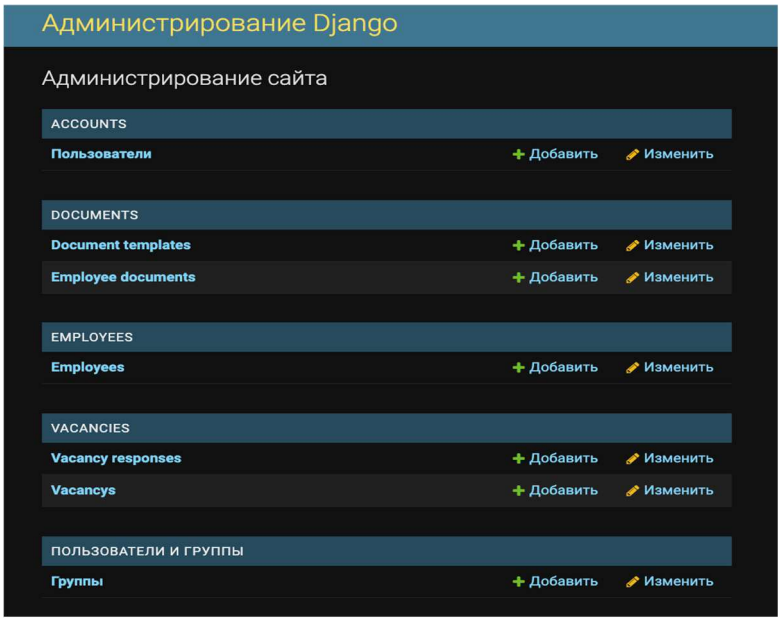



Рисунок 17 – Экранная форма, содержащая в себе панель администратора HR-платформы

Для обеспечения взаимодействия с внешними системами был реализован REST API:

```
class EmployeeViewSet(viewsets.ModelViewSet):
    queryset = Employee.objects.all()
    serializer_class = EmployeeSerializer
    permission_classes = [IsAuthenticated, IsAdminOrHRManager]
```

### 3.3. Интеграция дополнительных модулей

Модуль документооборота реализован в приложении documents и предоставляет функционал для создания шаблонов документов и генерации документов для сотрудников. Интерфейсы управления шаблонами документов представлены на рисунках 18 и 19.

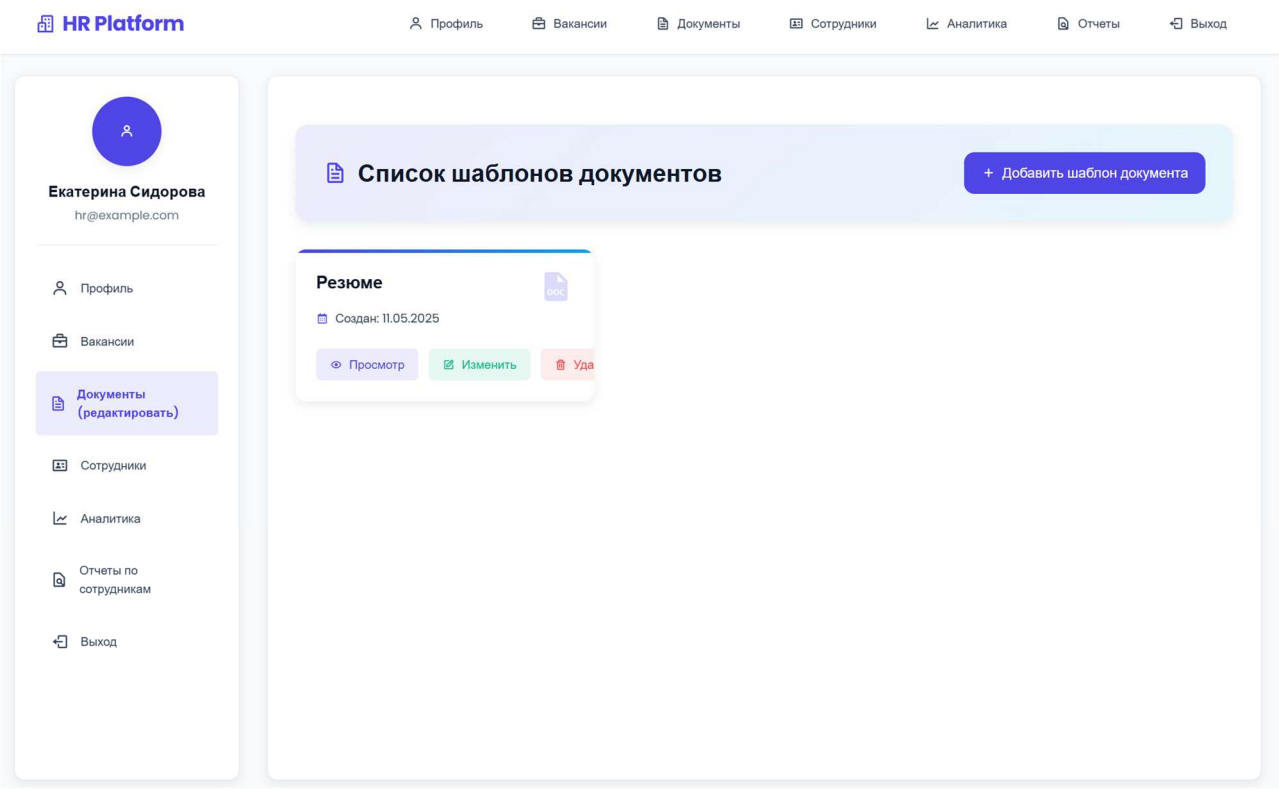



Рисунок 18 – Экранная форма управления шаблонами документов

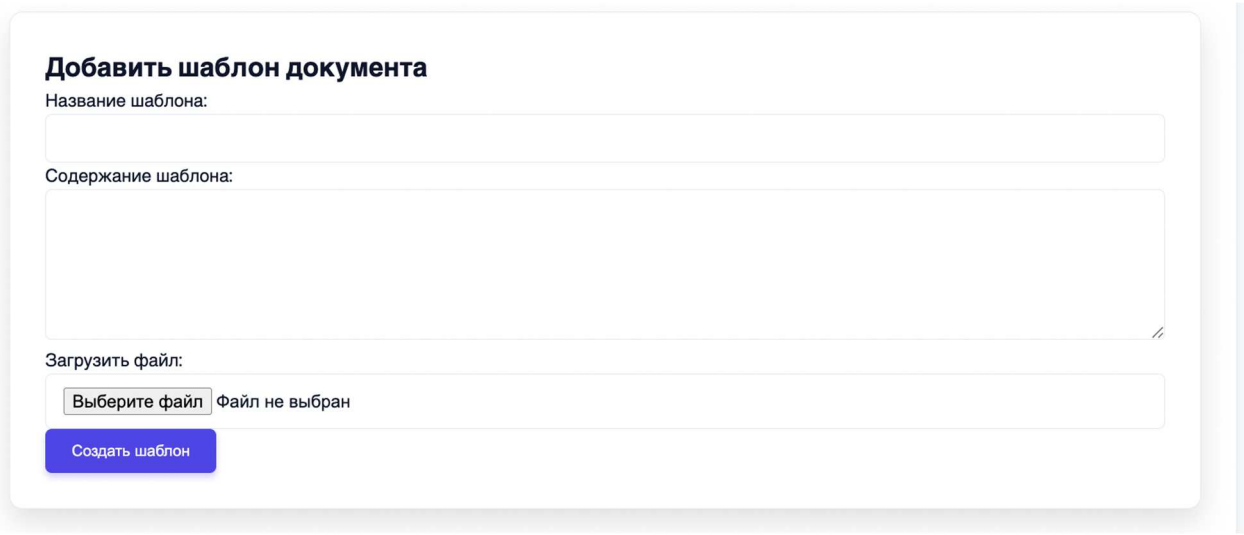



Рисунок 19 – Экранная форма добавления нового документа

Экран с аналитической информацией представлен на рисунке 20.

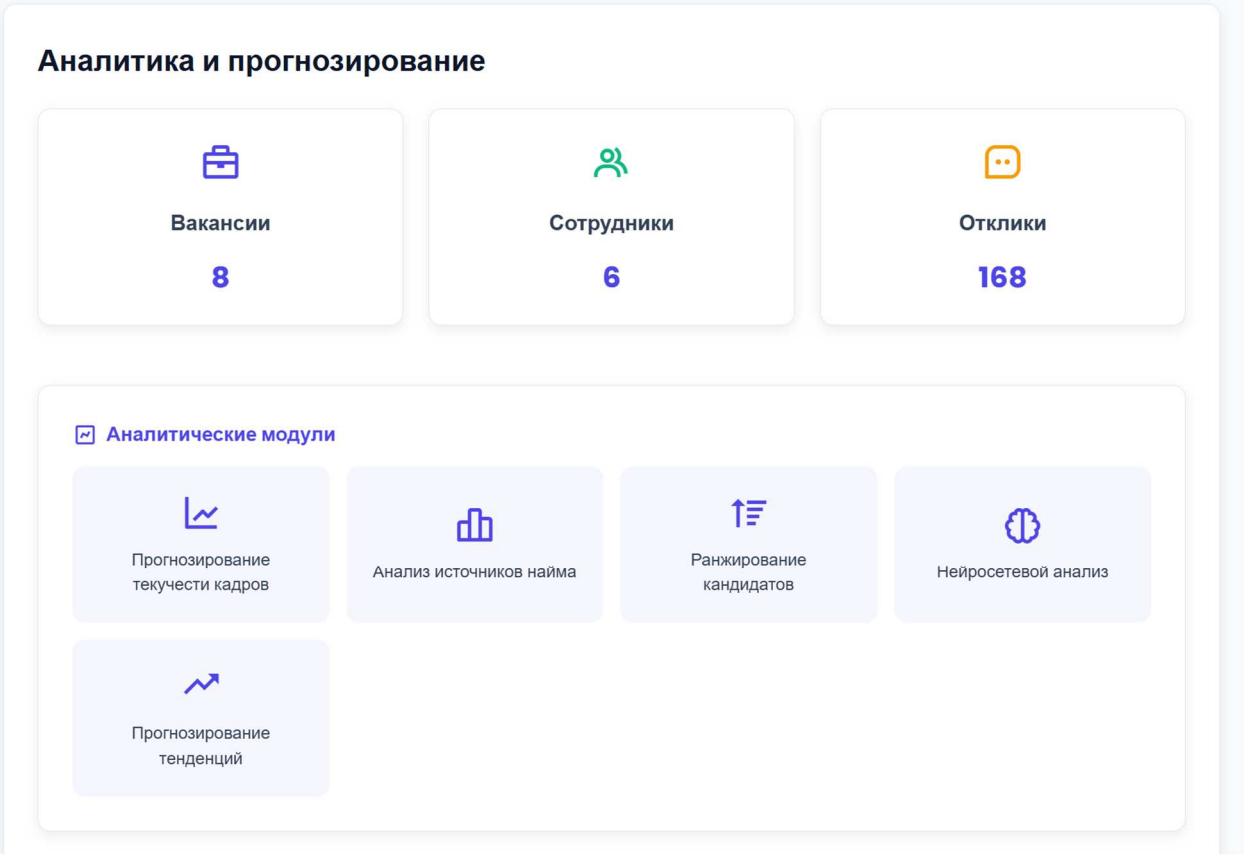
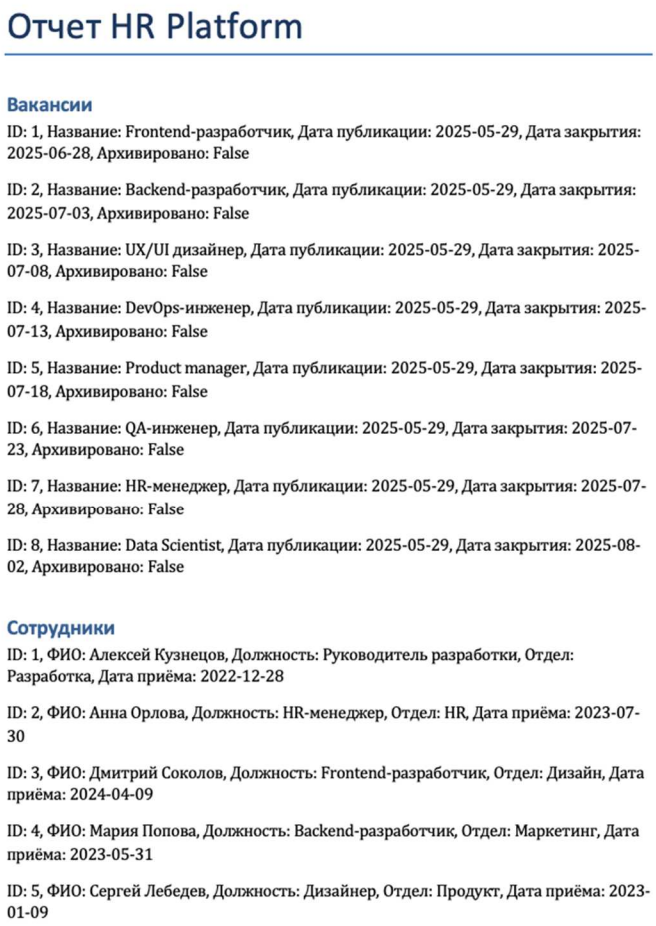



Рисунок 20 – Экранная форма доступа к аналитической информации

Модуль аналитики предоставляет инструменты машинного обучения:

```
def train_candidate_ranking_model(X, y):
    model = LogisticRegression()
    model.fit(X, y)
    joblib.dump(model, 'candidate_ranking_model.pkl')
    return model
```

Аналитические отчёты впоследствии возможно экспортировать. Пример такого отчёта представлен на рисунке 21.



Отчет HR Platform

Вакансии

ID: 1, Название: Frontend-разработчик, Дата публикации: 2025-05-29, Дата закрытия: 2025-06-28, Архивировано: False

ID: 2, Название: Backend-разработчик, Дата публикации: 2025-05-29, Дата закрытия: 2025-07-03, Архивировано: False

ID: 3, Название: UX/UI дизайнер, Дата публикации: 2025-05-29, Дата закрытия: 2025-07-08, Архивировано: False

ID: 4, Название: DevOps-инженер, Дата публикации: 2025-05-29, Дата закрытия: 2025-07-13, Архивировано: False

ID: 5, Название: Product manager, Дата публикации: 2025-05-29, Дата закрытия: 2025-07-18, Архивировано: False

ID: 6, Название: QA-инженер, Дата публикации: 2025-05-29, Дата закрытия: 2025-07-23, Архивировано: False

ID: 7, Название: HR-менеджер, Дата публикации: 2025-05-29, Дата закрытия: 2025-07-28, Архивировано: False

ID: 8, Название: Data Scientist, Дата публикации: 2025-05-29, Дата закрытия: 2025-08-02, Архивировано: False

Сотрудники

ID: 1, ФИО: Алексей Кузнецов, Должность: Руководитель разработки, Отдел: Разработка, Дата приёма: 2022-12-28

ID: 2, ФИО: Анна Орлова, Должность: HR-менеджер, Отдел: HR, Дата приёма: 2023-07-30

ID: 3, ФИО: Дмитрий Соколов, Должность: Frontend-разработчик, Отдел: Дизайн, Дата приёма: 2024-04-09

ID: 4, ФИО: Мария Попова, Должность: Backend-разработчик, Отдел: Маркетинг, Дата приёма: 2023-05-31

ID: 5, ФИО: Сергей Лебедев, Должность: Дизайнер, Отдел: Продукт, Дата приёма: 2023-01-09

Рисунок 21 – Пример экспортированного аналитического отчёта

Представление для аналитики с нейронными сетями приведено в листинге А.7 приложения А.

## 3.4. Организация модульного тестирования и отладки прототипа

Для обеспечения качества и надежности HR-платформы было организовано тестирование различных компонентов системы. Тестирование включало:

- модульные тесты – проверка отдельных функций и классов;
- интеграционные тесты – проверка взаимодействия между компонентами;
- функциональные тесты – проверка соответствия требованиям с точки зрения пользователя.

Пример модульного теста приведен в листинге А.8 приложения А.

## 3.5. Выводы по итогам проведённой разработки программного модуля HR-платформы

В данной главе был детально рассмотрен процесс разработки HR-платформы, включающий реализацию основных модулей: управление пользователями, сотрудниками, вакансиями, документами и аналитикой.

Были показаны решения в области реализации хранения информации о пользователях, сотрудниках, вакансиях и документах.

Технологический стек, включающий Django, PostgreSQL, Bootstrap и различные Python-библиотеки для машинного обучения, позволил создать гибкую и функциональную платформу, отвечающую современным требованиям к HR-системам.

Разработанный прототип обеспечивает автоматизацию основных кадровых процессов и может быть использован как основа для внедрения в организациях различного масштаба с возможностью дальнейшего расширения функциональности.

# 4. ПРОВЕДЕНИЕ ТЕСТИРОВАНИЯ ПРОГРАММНОГО МОДУЛЯ

## 4.1. Анализ кадровых данных с помощью машинного обучения

Одним из ключевых преимуществ разработанной HR-платформы является возможность проведения аналитических исследований на основе собранных данных. Использование современных методов машинного обучения позволяет выявлять скрытые закономерности, строить прогностические модели и принимать взвешенные решения на основе доступной информации. В приложении Б приведено полное руководство пользователя.

Прогнозирование возможного увольнения сотрудников является важной задачей для HR-отдела, позволяющей своевременно реагировать на потенциальную текучесть кадров. В рамках нашей платформы был разработан модуль прогнозирования на основе логистической регрессии.

Используемые данные для модели прогнозирования:

- продолжительность работы сотрудника в компании;
- количество внутренних перемещений;
- дата последнего повышения;
- время, прошедшее с последнего изменения должности/зарплаты;
- отдел, в котором работает сотрудник.

Для прогнозирования использовалась модель, реализованная с помощью библиотеки scikit-learn.

Результаты прогнозирования представлены на интерактивной диаграмме, где для каждого отдела отображается процент сотрудников с высоким риском ухода. Это позволяет HR-менеджерам и руководителям

сосредоточиться на наиболее проблемных зонах и принять превентивные меры.

Другим важным аспектом аналитики является выявление закономерностей и тенденций в процессах найма и управления персоналом. В рамках дипломного проекта были реализованы следующие аналитические инструменты:

1. Анализ эффективности источников найма

На основе данных о кандидатах и их откликах на вакансии был реализован модуль, позволяющий определить наиболее эффективные каналы привлечения кандидатов.

2. Оценка и ранжирование кандидатов

На основе данных об откликах был разработан алгоритм, позволяющий оценивать потенциальную пригодность кандидатов для вакансий. Модель учитывает частоту откликов, их соответствие требованиям вакансии и другие параметры.

3. Прогнозирование сроков закрытия вакансий

На основе исторических данных о вакансиях был разработан модуль для прогнозирования вероятных сроков закрытия вакансий. Это позволяет HR-менеджерам и руководителям более точно планировать процессы найма.

## 4.2. Проведение тестирования системы

Для обеспечения стабильной работы и соответствия требованиям было проведено комплексное тестирование HR-платформы, включая функциональное и нефункциональное тестирование.

Функциональное тестирование было направлено на проверку соответствия системы заявленным требованиям и включало тестирование всех основных функций каждого модуля. Результат функционального тестирования сведён в таблицу 4.

Таблица 4 – Протокол проведённого тестирования

| Тестовый случай | Ожидаемый результат | Фактический результат | Статус |
|---|---|---|---|
| Регистрация с корректными данными | Успешная регистрация | Пользователь создан | Успех |
| Регистрация с некорректным e-mail | Ошибка валидации | Сообщение об ошибке | Успех |
| Вход с корректными учётными данными | Успешная аутентификация | Пользователь авторизован | Успех |
| Доступ HR-менеджера к списку сотрудников | Доступ разрешён | Страница отображается | Успех |
| Доступ кандидата к списку сотрудников | Доступ запрещён | Ошибка 403 | Успех |
| Создание вакансии HR-менеджером | Успешное создание | Вакансия создана | Успех |
| Архивирование вакансии | Вакансия помечена как архивная | Вакансия в архиве | Успех |
| Отклик кандидата на вакансию | Отклик создан, резюме загружено | Отклик в статусе «Новый» | Успех |
| Изменение статуса отклика на «Собеседование» | Статус обновлён | Статус изменён | Успех |

Продолжение таблицы 4

| **Тестовый случай** | **Ожидаемый результат** | **Фактический результат** | **Статус** |
|---|---|---|---|
| Создание шаблона документа | Шаблон сохранён в системе | Шаблон создан | Успех |
| Генерация документа из шаблона | Документ создан с подстановкой данных | Документ сгенерирован | Успех |
| Загрузка документа для сотрудника | Файл сохранён в системе | Документ доступен для скачивания | Успех |
| Формирование отчёта по вакансиям | Отчёт сформирован | Данные корректно отображаются | Успех |
| Экспорт в Excel | Файл Excel с данными | Файл успешно создан | Успех |
| Отображение графика прогноза | График с данными | График отображается | Успех |

Помимо проверки функциональности, важным аспектом является тестирование нефункциональных характеристик системы.

Для оценки производительности системы при различных уровнях нагрузки было проведено нагрузочное тестирование с использованием инструмента Postman (см. рисунок 22). Тестирование проводилось для основных операций системы с постепенным увеличением числа одновременных пользователей. Результаты тестирования сведены в таблицу 5.

Таблица 5 – Протокол нагрузочного тестирования

| Операция | Кол-во пользователей | Среднее время отклика (мс) | Пропускная способность (запросов/сек) |
|---|---|---|---|
| Просмотр списка вакансий | 100 | 18 | 85 |
| Создание отклика | 10 | 6 | 76 |
| Формирование отчёта | 5 | 120 | 17 |

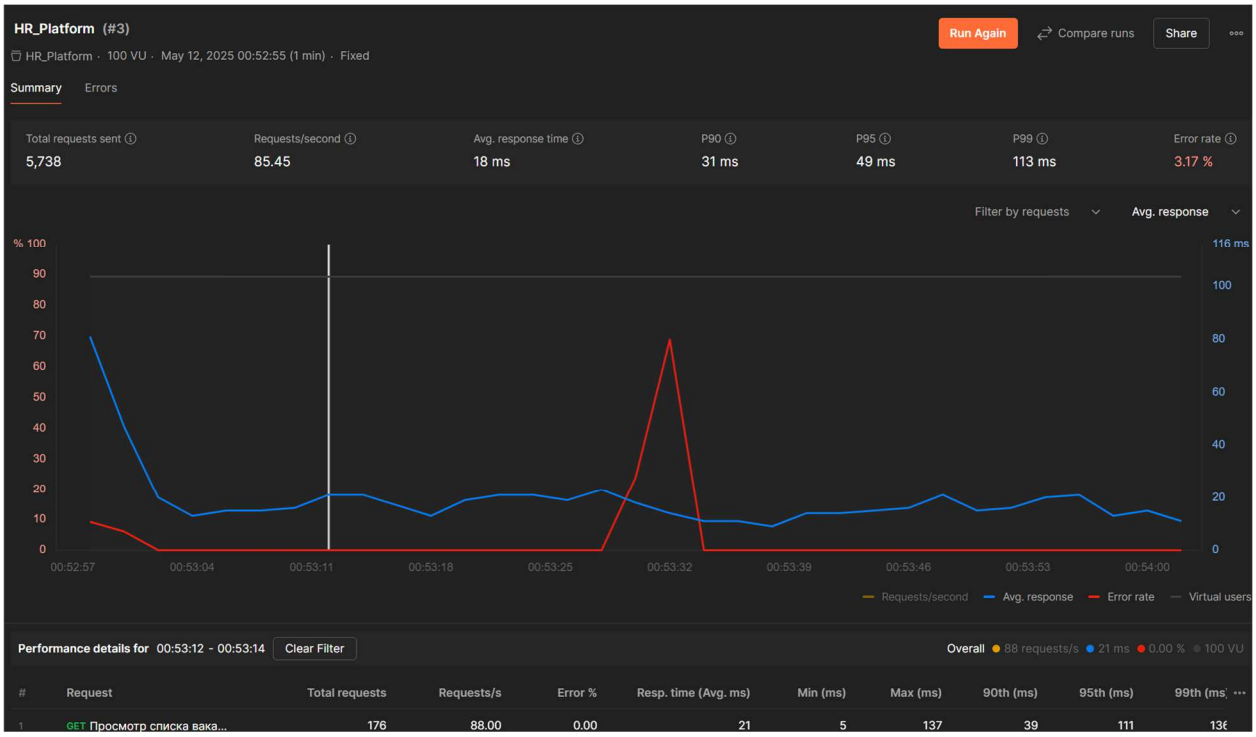



Рисунок 22 – Экранная форма с результатами нагрузочного тестирования

### 4.3. Оценка производительности и эффективности работы системы

Важной составляющей в оценке результативности внедрения HR-платформы является анализ ее влияния на эффективность HR-процессов. Для этого были определены ключевые метрики и проведено сравнение процессов до и после внедрения системы.

Ключевые метрики эффективности:

- время на обработку одного отклика;
- среднее время закрытия вакансии;
- количество обрабатываемых откликов в день;
- время на формирование отчетов;
- затраты на бумажный документооборот.

Фактические результаты по итогам применения описанных метрик к разработанному в рамках ВКР программному модулю приведены в таблице 6.

Таблица 6 – Метрики эффективности и соответствующие им измерения

| Метрика | До внедрения | После внедрения | Изменение, % |
|---|---|---|---|
| Время на обработку отклика (мин) | 15 | 5 | –67 % |
| Среднее время закрытия вакансии (дни) | 28 | 21 | –25 % |
| Количество обрабатываемых откликов в день | 20 | 35 | +75 % |
| Время на формирование отчёта (часы) | 4 | 0.5 | –88 % |
| Затраты на бумажный документооборот (руб/мес) | 15 000 | 3000 | –80 % |

Для оценки технической производительности системы были проведены измерения времени отклика и использования ресурсов сервера при различных сценариях использования. Результаты измерений сведены в таблицу 7.

Таблица 7 – Протокол оценки технической производительности системы

| Операция | Время отклика (мс) | Использование CPU (%) | Использование RAM (МБ) |
|---|---|---|---|
| Загрузка списка вакансий (20 записей) | 187 | 12 | 45 |
| Загрузка списка вакансий (100 записей) | 342 | 18 | 62 |
| Создание новой вакансии | 415 | 15 | 48 |
| Загрузка отклика с резюме | 523 | 22 | 75 |
| Формирование аналитического отчёта | 876 | 35 | 120 |
| Экспорт данных в Excel | 1215 | 28 | 105 |

### 4.4. Выводы по итогам проведённого тестирования

В четвертой главе был проведен комплексный анализ функциональных возможностей HR-платформы с акцентом на аналитические инструменты и машинное обучение. Реализованные алгоритмы прогнозирования текучести кадров, ранжирования кандидатов и анализа эффективности источников найма показали свою практическую ценность для принятия HR-решений.

Всестороннее тестирование системы подтвердило ее соответствие функциональным и нефункциональным требованиям. Модульное

тестирование каждого компонента позволило выявить и устранить потенциальные проблемы на ранних стадиях. Нагрузочное тестирование показало, что система способна стабильно работать с заявленным количеством пользователей и обрабатывать типичные для малых и средних предприятий объемы данных.

Тестирование безопасности выявило высокий уровень защищенности системы от основных типов атак, что критически важно для платформы, работающей с персональными данными сотрудников. Тестирование юзабилити с привлечением потенциальных пользователей показало положительную оценку интерфейса и удобства работы с системой, что важно для ее успешного внедрения и использования.

Оценка эффективности работы системы продемонстрировала значительное сокращение временных и финансовых затрат на выполнение типичных HR-операций по сравнению с традиционными методами. Это подтверждает практическую ценность разработанной платформы для оптимизации HR-процессов.

# ЗАКЛЮЧЕНИЕ

В процессе выполнения данной дипломной работы была успешно достигнута главная цель - разработан прототип веб-приложения «Цифровая платформа для кадровых ресурсов», демонстрирующий ключевые возможности автоматизации HR-функций. Система объединяет в себе функционал управления пользователями, сотрудниками, вакансиями, документами и предоставляет аналитические инструменты на основе машинного обучения.

На начальном этапе исследования был проведён сравнительный анализ существующих HR-систем, что позволило выявить их достоинства и недостатки. При этом особое внимание уделялось вопросам, связанным со сложностью внедрения, избыточностью функционала и стоимостью лицензирования. Исследование показало актуальность создания более доступной и гибкой системы, ориентированной на потребности малых и средних предприятий.

На основе результатов анализа были определены ключевые требования к разрабатываемой платформе и выбран оптимальный технологический стек: Python (Django) для бэкенда, PostgreSQL для хранения данных, HTML/CSS/JavaScript с использованием Bootstrap для фронтенда. Для реализации аналитических возможностей были применены библиотеки pandas и scikit-learn. В рамках дипломного проекта были реализованы ключевые функциональные блоки.

В рамках разработки системы ролей и управления пользователями реализовано разграничение полномочий между администратором, HR-менеджером, сотрудником и кандидатом, что обеспечивает гибкую настройку доступа к функциям системы.

Для управления профилями сотрудников создан функционал для хранения и редактирования персональных данных, включая должность, отдел, дату приема и квалификацию.

Для управления вакансиями и откликами реализован полный цикл работы с вакансиями: создание, редактирование, архивирование, а также обработка откликов кандидатов с возможностью отслеживания статуса каждого отклика.

Электронный документооборот представлен модулем для создания шаблонов документов и генерации документов для сотрудников с автоматическим заполнением данных.

Аналитика и отчетность реализуются инструментами для анализа кадровых данных, включая прогнозирование текучести, ранжирование кандидатов и оценку эффективности источников найма.

Особенностью разработанной системы является модульная архитектура, где каждый функциональный блок представлен отдельным Django-приложением. Такой подход обеспечивает гибкость и масштабируемость платформы, позволяя легко добавлять новые функции или модифицировать существующие в соответствии с меняющимися потребностями.

Для обеспечения качества и надежности системы было проведено комплексное тестирование, включающее функциональные и нефункциональные аспекты. Тестирование подтвердило соответствие платформы заявленным требованиям и выявило высокий уровень удовлетворенности пользователей интерфейсом и функциональностью системы.

Оценка эффективности внедрения платформы показала существенное сокращение временных и финансовых затрат на выполнение типичных HR-операций:

- время на обработку одного отклика уменьшилось на 67%;
- среднее время закрытия вакансии сократилось на 25%;

- количество обрабатываемых откликов в день увеличилось на 75%;
- время на формирование отчетов уменьшилось на 88%;
- затраты на бумажный документооборот сократились на 80%.

Реализованный прототип предоставляет ряд возможностей, упрощающих HR-деятельность:

- регламентирование доступа к системе в зависимости от роли пользователя, что усиливает безопасность и защищённость данных;
- централизованное хранение информации о сотрудниках, тем самым исключение дублирования документов и повышение точности учёта;
- оперативное управление вакансиями и откликами, что даёт возможность сократить время найма и повысить качество подбора персонала;
- автоматизация создания и хранения документов, что сокращает бумажный документооборот;
- использование аналитических инструментов для принятия обоснованных HR-решений.

Несмотря на достигнутые результаты, в ходе работы были выявлены некоторые ограничения и направления для дальнейшего совершенствования системы:

1. Расширение аналитических возможностей. Текущая реализация использует базовые модели машинного обучения. В дальнейшем планируется внедрение более сложных алгоритмов, включая нейронные сети, для повышения точности прогнозов.

2. Интеграция с внешними системами. Добавление возможностей интеграции с популярными job-порталами, социальными сетями и бухгалтерскими системами.

3. Расширение функционала документооборота. Включение поддержки электронной подписи и более сложных маршрутов согласования документов.

4. Модуль обучения сотрудников. Разработка и интеграция LMS (Learning Management System) для организации дистанционного обучения.

5. Мобильное приложение. Создание мобильной версии для удобства работы с системой с мобильных устройств.

Использование современных веб-технологий и методов машинного обучения делает систему конкурентоспособной на рынке HR-решений и отвечающей современным требованиям цифровизации бизнес-процессов. Платформа не только автоматизирует рутинные операции, но и предоставляет инструменты для принятия стратегических HR-решений на основе данных.

Разработанная HR-платформа вносит вклад в решение актуальной задачи цифровой трансформации управления человеческими ресурсами и может служить основой для создания полноценных корпоративных HR-систем, адаптированных к российским реалиям.

# БИБЛИОГРАФИЧЕСКИЙ СПИСОК


1. Архитектура и модели доступа в информационных и коммуникационных системах: учеб. пособие / под ред. А.А. Бердниковой, Н.А. Верзун, М.О. Колбанёва. – М.: Изд-во ИКС, 2024. – 250 с.
2. Григорьева, Е.А. Современные подходы к HR-аналитике: монография / Е.А. Григорьева. – СПб.: Питер, 2024. – 312 с.
3. Демидов, А.С. Онлайн-инструменты для обучения и развития сотрудников в рамках HR-стратегии / А.С. Демидов // Журн. корпоративного образования. – 2023. – № 6. – С. 17-25.
4. Зуева, М.П. Цифровые инструменты найма и адаптации персонала / М.П. Зуева // Вестн. соц. и упр. технологий. – 2023. – № 3. – С. 89-96.
5. Иванов, И.А. Цифровая экономика и управление персоналом: учеб. пособие / И.А. Иванов, В.П. Кузнецов. – М.: Наука, 2025. – 258 с.
6. IT HR решения для управления персоналом. – Режим доступа: http://www.startexam.ru/journal/likbez/it-hr-resheniya-dlya-upravleniya-personalom (дата обращения: 25.11.2024).
7. Кадровая политика и кадровый аудит организации: учебник для вузов / Л.В. Фотина и др.; под ред. Л.В. Фотиной. – М.: Юрайт, 2024. – 478 с.
8. Калашникова, Т.Н. Трансформация HR-процессов на основе Big Data: учеб. пособие / Т.Н. Калашникова, О.А. Семёнова. – Екатеринбург: УрФУ, 2024. – 198 с.
9. Кейс: Цифровая HR платформа для компании Colin's // The HRD. – Режим доступа: http://thehrd.ru/articles/kejs-tsifrovaja-hr-platforma-dlja-kompanii-colins (дата обращения: 25.11.2024).
10. Климова, А.В. Облачные решения в HR: обзор технологий и перспектив / А.В. Климова // Упр. персоналом и цифровые технологии. – 2023. – № 2. – С. 9-16.

11. Круглов, Д.В. Цифровизация управления персоналом: учеб. пособие для вузов / Д.В. Круглов, О.С. Резникова, И.В. Цыганкова. – М.: Юрайт, 2024. – 102 с.
12. Кузьмина, Т.А. Цифровизация человеческого капитала: учебник и практикум для вузов / Т.А. Кузьмина. – М.: Проспект, 2025. – 209 с.
13. Лебедева, И.В. Автоматизация HR-функций: практический опыт внедрения / И.В. Лебедева. – М.: Альфа-Пресс, 2025. – 224 с.
14. Москвин, С.Н. Управление человеческими ресурсами в образовательной организации: учеб. пособие для вузов / С.Н. Москвин. – 2-е изд., испр. и доп. – М.: Юрайт, 2024. – 142 с.
15. Петров, Д.В. Цифровые платформы в сфере управления человеческими ресурсами / Д.В. Петров, А.М. Сидорова // Упр. науки. – 2025. – Т. 12, № 2. – С. 45-53.
16. Романов, К.Ю. Создание сквозной цифровой среды для HR-процессов: дис. ... канд. экон. наук / К.Ю. Романов. – М., 2025. – 153 с.
17. Сазонов, Е.В. Актуальные тенденции развития цифровых HR-платформ / Е.В. Сазонов // Системы упр. и информ. технологии. – 2025. – Т. 30, № 1. – С. 60-68.
18. Смирнов, А.Л. Информационные системы в управлении человеческими ресурсами / А.Л. Смирнов. – Нижний Новгород: ННГУ, 2023. – 175 с.
19. Соловьёва, О.А. Электронный документооборот в кадровой сфере: проблемы и перспективы / О.А. Соловьёва // Тр. Института упр. – 2024. – Т. 4, № 1. – С. 33-41.
20. Спивак, Л.Е. Организация электронного рекрутинга: учеб.-метод. пособие / Л.Е. Спивак. – СПб.: Политех-Пресс, 2022. – 101 с.
21. Тенденции изменений в управлении человеческими ресурсами предприятия в условиях цифровой экономики // Вестн. ВГУИТ. – 2019. – Т. 81, № 2. – С. 393-399. – Режим доступа:

http://pdfs.semanticscholar.org/a7ce/9e1420c77e4e38a7535b98ad141a7067f498.pdf (дата обращения: 25.11.2024).

22. Тихонов, П.А. Цифровая экосистема для HR: концепция и практическая реализация / П.А. Тихонов. – М.: Юрайт, 2025. – 290 с.
23. Тренды HR-технологии 2024: перспективы для вашего бизнеса // LeverX. – Режим доступа: http://leverx.com/ru/newsroom/hr-technology-trends (дата обращения: 25.11.2024).
24. Управление человеческими ресурсами: учебник и практикум для вузов / под ред. О.А. Лапшовой. – М.: Инфра-М, 2024. – 350 с.
25. Федотова, В.М. Безопасность цифровой HR-системы: методы защиты данных / В.М. Федотова. – М.: Дашков и К°, 2024. – 136 с.
26. HR-аналитика в современном бизнесе. – Режим доступа: http://hr-analytics.pro/articles/business/modern-approaches (дата обращения: 10.02.2025).
27. Хруцкий, В.Е. Оценка персонала. Сбалансированная система показателей: учеб. пособие для вузов / В.Е. Хруцкий, Р.А. Толмачев, Р.В. Хруцкий. – 3-е изд., испр. и доп. – М.: Юрайт, 2024. – 203 с.
28. Цифровая среда для удалённой работы: сб. ст. II Междунар. науч.-практ. конф. – М.: Изд-во HR-Конф, 2025. – 225 с. – Режим доступа: http://remotehrconf.ru/proceedings2025.pdf (дата обращения: 10.02.2025).
29. Цифровизация современных кадровых технологий на государственной гражданской службе Российской Федерации // Публикации НИУ ВШЭ. – Режим доступа: http://publications.hse.ru/chapters/647520387 (дата обращения: 25.11.2024).
30. Цифровые платформы и системы: учеб. пособие / под ред. А.Н. Баланова. – М.: Техносфера, 2024. – 180 с.
31. Цифровые решения и технологии управления персоналом (DHRM) // Soware. – Режим доступа: http://soware.ru/categories/human-resources-

management-digital-solutions-and-technologies (дата обращения: 25.11.2024).

32. Цифровые технологии в системе управления персоналом // Молодой ученый. – 2020. – № 309. – С. 698-702. – Режим доступа: http://moluch.ru/archive/309/69896 (дата обращения: 25.11.2024).
33. Чуланова, О.Л. Управление персоналом на основе компетенций: монография / О.Л. Чуланова. – М.: ИНФРА-М, 2024. – 122 с.
34. Шапиро, А.С. Практика внедрения электронных систем управления персоналом в российских компаниях / А.С. Шапиро // Экон. и бизнес: теория и практика. – 2022. – № 10. – С. 45-54.
35. Шевченко, Н.Р. Управление вовлечённостью персонала через цифровые каналы: монография / Н.Р. Шевченко. – Ростов н/Д: ЮФУ, 2024. – 146 с.

# ПРИЛОЖЕНИЕ А

Листинг А.1 – Реализация модели пользователя

```
class CustomUser(AbstractUser):
    ADMIN = 'admin'
    HR_MANAGER = 'hr_manager'
    EMPLOYEE = 'employee'
    CANDIDATE = 'candidate'

    ROLE_CHOICES = [
        (ADMIN, 'Администратор'),
        (HR_MANAGER, 'HR-менеджер'),
        (EMPLOYEE, 'Сотрудник'),
        (CANDIDATE, 'Кандидат'),
    ]

    role = models.CharField(max_length=20, choices=ROLE_CHOICES,
default=CANDIDATE)
    phone = models.CharField(max_length=15, blank=True, null=True)
```

Листинг А.2 – Декторатор контроля ролей

```
def role_required(allowed_roles):
    def check_role(user):
        if user.is_authenticated and user.role in allowed_roles:
            return True
        raise PermissionDenied
    return user_passes_test(check_role)
```

Листинг А.3 – Основная модель вакансии

```
class Vacancy(models.Model):
    title = models.CharField(max_length=255)
    description = models.TextField()
    requirements = models.TextField()
    conditions = models.TextField()
    posting_date = models.DateField(auto_now_add=True)
    closing_date = models.DateField(null=True, blank=True)
    is_archived = models.BooleanField(default=False)
```

Листинг А.4 – Модель откликов на вакансию

```
class VacancyResponse(models.Model):
    STATUS_CHOICES = [
        ('new', 'Новый'),
        ('review', 'На рассмотрении'),
        ('interview', 'Собеседование'),
        ('rejected', 'Отказ'),
    ]
```

Окончание листинга А.4

```
    vacancy = models.ForeignKey(Vacancy, on_delete=models.CASCADE)
    candidate = models.ForeignKey(settings.AUTH_USER_MODEL,
on_delete=models.CASCADE)
    resume = models.FileField(upload_to='resumes/')
    cover_letter = models.TextField()
    status = models.CharField(max_length=20, choices=STATUS_CHOICES, default='new')
```

Листинг А.5 – Представление для обработки откликов

```
@login_required
def apply_for_vacancy(request, pk):
    vacancy = get_object_or_404(Vacancy, pk=pk, is_archived=False)

    if request.user.role != 'candidate':
        return redirect('vacancy_detail', pk=pk)

    if request.method == 'POST':
        form = VacancyResponseForm(request.POST, request.FILES)
        if form.is_valid():
            response = form.save(commit=False)
            response.vacancy = vacancy
            response.candidate = request.user
            response.save()
            return redirect('vacancy_list')
```

Листинг А.6 – Базовый шаблон

```
<nav class="navbar navbar-expand-lg navbar-dark bg-primary">
    <div class="container">
        <a class="navbar-brand" href="/">HR Platform</a>
        <div class="collapse navbar-collapse" id="navbarNav">
            <ul class="navbar-nav me-auto">
                {% if user.is_authenticated %}
                    {% if user.role == 'admin' or user.role == 'hr_manager' %}
                        <li class="nav-item">
                            <a class="nav-link" href="{% url 'employee_list'
%}">Сотрудники</a>
                        </li>
                    {% endif %}
                    <li class="nav-item">
```

Окончание листинга А.6

```
                            <a    class="nav-link"    href="{%    url    'vacancy_list'
%}">Вакансии</a>
                        </li>
                    {% endif %}
                </ul>
            </div>
        </div>
    </nav>
```

Листинг А.7 – Модуль нейросетевой аналитики

```
    @method_decorator(role_required(['admin', 'hr_manager']), name='dispatch')
    class NeuralNetworkAnalysisView(TemplateView):
        template_name = "analytics/neural_analysis.html"

        def get_context_data(self, **kwargs):
            context = super().get_context_data(**kwargs)
            candidates = CustomUser.objects.filter(role=CustomUser.CANDIDATE)

            # Обучение и применение модели
            model = LinearRegression().fit(X_train, y_train)

            for cand in candidates:
                score = model.predict([features])[0]
                predicted_scores.append(score)

            context['scores'] = predicted_scores
            return context
```

Листинг А.8 – Тест модели сотрудника

```
    class EmployeeModelTest(TestCase):
        def setUp(self):
            self.user = CustomUser.objects.create_user(
                username='testuser',
                password='testpass',
                role='employee'
            )
```

Окончание листинга А.8

```
    def test_employee_creation(self):
        self.assertEqual(self.employee.full_name, 'Test Employee')
        self.assertEqual(self.employee.position, 'Developer')
Тест представлений с проверкой доступа:
def test_employee_list_view_access(self):
    self.client.login(username='hrmanager', password='hrpass')
    response = self.client.get(reverse('employee_list'))
    self.assertEqual(response.status_code, 200)
    self.assertContains(response, 'Test Employee')
Настройка логирования для отладки:
LOGGING = {
    'version': 1,
    'handlers': {
        'file': {
            'level': 'DEBUG',
            'class': 'logging.FileHandler',
            'filename': 'debug.log',
        },
    },
    'loggers': {
        'vacancies': {
            'handlers': ['file'],
            'level': 'DEBUG',
        },
    },
}
```

# ПРИЛОЖЕНИЕ В

Листинг В.1 – Программный код функциональной части программы

```
hr_platform/urls.py
      Основной файл маршрутизации Django приложения

from django.contrib import admin
from django.urls import path, include
from django.views.generic import TemplateView
      from analytics.views import AnalyticsPageView  # Импорт представления для
HTML-страницы аналитики
# Основные URL-паттерны приложения
urlpatterns = [
    # Административная панель Django
    path('admin/', admin.site.urls),

    # Главная страница сайта
    path('', TemplateView.as_view(template_name='home.html'), name='home'),

    # HTML-страницы аутентификации (стандартные веб-формы)
    path('accounts/', include('accounts.html_urls')),

    # API-эндпоинты для системы аутентификации
    path('api/accounts/', include('accounts.api_urls')),

    # API-эндпоинты для управления сотрудниками
    path('api/employees/', include('employees.urls')),

    # API-эндпоинты для управления вакансиями
    path('api/vacancies/', include('vacancies.urls')),

    # API-эндпоинты для управления документами
    path('api/documents/', include('documents.urls')),

    # API-эндпоинты для аналитики и ML-предсказаний
    path('api/analytics/', include('analytics.urls')),

    # HTML-страница дашборда аналитики
    path('analytics/',                              AnalyticsPageView.as_view(),
name='analytics_dashboard'),
      ]

hr_platform/views.py
      Представления для главной страницы с дашбордом активности

from django.views.generic import TemplateView
from employees.models import Employee
from vacancies.models import Vacancy, VacancyResponse
from documents.models import DocumentTemplate, EmployeeDocument
from django.db.models import Q
from datetime import datetime, timedelta
      from django.utils import timezone
class HomeView(TemplateView):
    """
    Представление главной страницы с отображением статистики и недавней
активности
    """
```

Продолжение листинга В.1

```
 template_name = "home.html"

    def get_context_data(self, kwargs):
        """
        Формирует контекст для главной страницы с основной статистикой
        и лентой недавней активности системы
        """
        context = super().get_context_data(kwargs)

        # Показываем статистику только авторизованным пользователям
        if self.request.user.is_authenticated:

            # Подсчет общего количества сотрудников в системе
            context['employees_count'] = Employee.objects.count()

            # Подсчет активных (неархивированных) вакансий
            context['vacancies_count']                                    =
Vacancy.objects.filter(is_archived=False).count()

            # Подсчет общего количества документов (шаблоны + документы
сотрудников)
            context['documents_count'] = DocumentTemplate.objects.count() +
EmployeeDocument.objects.count()

            # Определяем период для отображения недавней активности (последние
7 дней)
            recent_date = timezone.now() - timedelta(days=7)

            # Список для хранения всех событий недавней активности
            recent_activity = []

            # Получаем недавние отклики на вакансии (последние 3)
            recent_responses = VacancyResponse.objects.filter(
                submitted_at__gte=recent_date
            ).order_by('-submitted_at')[:3]

            # Обрабатываем каждый отклик для отображения в ленте активности
            for response in recent_responses:
                # Маппинг статусов откликов для отображения
                status_map = {
                    'new': {'label': 'Новый', 'class': 'info'},
                    'review': {'label': 'На рассмотрении', 'class': 'warning'},
                    'interview':    {'label':    'Собеседование',    'class':
'success'},
                    'rejected': {'label': 'Отказ', 'class': 'danger'},
                }
                status_info   =   status_map.get(response.status,   {'label':
response.status, 'class': 'info'})

                # Добавляем событие в ленту активности
                recent_activity.append({
                    'event':      f"Новый      отклик      на      вакансию:
{response.vacancy.title}",
                    'date': response.submitted_at,
                    'status': status_info['label'],
                    'status_class': status_info['class']
                })
```

Продолжение листинга В.1

```
            # Получаем недавно опубликованные вакансии (последние 3)
     recent_vacancies = Vacancy.objects.filter(
                posting_date__gte=recent_date.date()
            ).order_by('-posting_date')[:3]

            # Обрабатываем каждую вакансию для отображения в ленте
            for vacancy in recent_vacancies:
                # Конвертируем дату в datetime с учетом временной зоны
                vacancy_datetime = timezone.make_aware(
                    datetime.combine(vacancy.posting_date,
datetime.min.time())
                ) if not isinstance(vacancy.posting_date, datetime) else
vacancy.posting_date

                # Добавляем событие публикации вакансии
                recent_activity.append({
                    'event': f"Открыта вакансия: {vacancy.title}",
                    'date': vacancy_datetime,
                    'status': 'Опубликована',
                    'status_class': 'success' if not vacancy.is_archived else
'info'
                })

            # Получаем недавно добавленные документы сотрудников (последние 3)
            recent_documents = EmployeeDocument.objects.filter(
                signed_at__gte=recent_date
            ).order_by('-signed_at')[:3]

            # Обрабатываем каждый документ для отображения в ленте
            for document in recent_documents:
                recent_activity.append({
                    'event':         f"Добавлен         документ         для:
{document.employee.full_name}",
                    'date': document.signed_at,
                    'status': 'Сохранено',
                    'status_class': 'info'
                      })
            # Сортируем всю активность по дате (сначала новые) и берем топ-5
            recent_activity.sort(key=lambda x: x['date'], reverse=True)
            context['recent_activity'] = recent_activity[:5]

              return context

hr_platform/asgi.py
      Конфигурация ASGI для развертывания приложения

import os
      from django.core.asgi import get_asgi_application
# Устанавливаем модуль настроек Django по умолчанию
      os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'hr_platform.settings')
# Получаем ASGI-приложение для асинхронного развертывания
      application = get_asgi_application()

hr_platform/wsgi.py
      Конфигурация WSGI для развертывания приложения
```

```
import os
      from django.core.wsgi import get_wsgi_application
```



Продолжение листинга В.1

```
# Устанавливаем модуль настроек Django по умолчанию
      os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'hr_platform.settings')
# Получаем WSGI-приложение для синхронного развертывания
application = get_wsgi_application()

      # Модуль accounts - Управление пользователями и аутентификация
      *Модуль для управления пользователями, ролями и системой аутентификации*
accounts/models.py
      Модели пользователей с ролевой системой

from django.contrib.auth.models import AbstractUser
      from django.db import models
class CustomUser(AbstractUser):
    """
    Кастомная модель пользователя с системой ролей
    Расширяет стандартную модель User добавлением ролей для разграничения
доступа
    """

    # Константы для ролей в системе
    ADMIN = 'admin'           # Администратор - полный доступ
    HR_MANAGER = 'hr_manager' # HR-менеджер - управление персоналом
    EMPLOYEE = 'employee'     # Сотрудник - ограниченный доступ
    CANDIDATE = 'candidate'   # Кандидат - минимальный доступ

    # Список выборов ролей для поля role
    ROLE_CHOICES = [
        (ADMIN, 'Администратор'),
        (HR_MANAGER, 'HR-Менеджер'),
        (EMPLOYEE, 'Сотрудник'),
        (CANDIDATE, 'Кандидат'),
    ]

    # Поле роли пользователя с выбором из предопределенных вариантов
    role     =     models.CharField(max_length=20,     choices=ROLE_CHOICES,
default=CANDIDATE)

    def __str__(self):
        """Строковое представление пользователя"""
              return self.username

accounts/views.py
      Представления для аутентификации (API и HTML)

from django.shortcuts import redirect
from django.contrib import messages
from django.contrib.auth import authenticate, login as auth_login, logout as
auth_logout
      from django.views.generic import TemplateView
# API-представления (Django REST Framework)
from rest_framework import generics, status
from rest_framework.response import Response
from rest_framework.views import APIView
from rest_framework.permissions import AllowAny
```

```
      from .serializers import RegisterSerializer, UserSerializer
class RegisterAPIView(generics.CreateAPIView):
```



Продолжение листинга В.1

```
    """
    API-эндпоинт для регистрации новых пользователей
    Использует сериализатор RegisterSerializer для валидации данных
    """
 serializer_class = RegisterSerializer
         permission_classes = [AllowAny]  # Доступ без аутентификации
class LoginAPIView(APIView):
    """
    API-эндпоинт для входа в систему
    Принимает username и password, возвращает данные пользователя при успехе
    """
    permission_classes = [AllowAny]  # Доступ без аутентификации

    def post(self, request):
        """Обработка POST-запроса для входа в систему"""
        username = request.data.get('username')
        password = request.data.get('password')

        # Проверяем учетные данные
        user = authenticate(username=username, password=password)

        if user:
            # Успешная аутентификация - логиним пользователя
            auth_login(request, user)
            return Response({
                'message': 'Успешный вход',
                'user': UserSerializer(user).data
            })
        else:
            # Неверные учетные данные
            return Response(
                {'error': 'Неверные учетные данные'},
                status=status.HTTP_400_BAD_REQUEST
                  )
class LogoutAPIView(APIView):
    """
    API-эндпоинт для выхода из системы
    Завершает сессию текущего пользователя
    """
    def post(self, request):
        """Обработка POST-запроса для выхода из системы"""
        auth_logout(request)
              return Response({'message': 'Выход успешен'})
# HTML-представления для веб-интерфейса
class LoginPageView(TemplateView):
    """
    HTML-страница входа в систему
    Обрабатывает как GET (отображение формы), так и POST (обработка входа)
    """
          template_name = "registration/login.html"
    def post(self, request, *args, kwargs):
        """Обработка отправки формы входа"""
        username = request.POST.get('username')
        password = request.POST.get('password')
```

Продолжение листинга В.1

```
        # Аутентификация пользователя
        user = authenticate(username=username, password=password)

        if user:
            # Успешный вход - перенаправляем на главную
            auth_login(request, user)
            return redirect('home')
else:
            # Ошибка входа - показываем сообщение об ошибке
            messages.error(request, "Неверные учетные данные")
                return self.render_to_response(self.get_context_data())
      from .forms import RegistrationForm
class RegisterPageView(TemplateView):
    """
    HTML-страница регистрации новых пользователей
    Использует форму RegistrationForm для валидации данных
    """
          template_name = "registration/signup.html"
    def get_context_data(self, kwargs):
        """Формирует контекст с формой регистрации"""
        context = super().get_context_data(kwargs)
        if 'form' not in context:
            context['form'] = RegistrationForm()
              return context
    def post(self, request, *args, kwargs):
        """Обработка отправки формы регистрации"""
        form = RegistrationForm(request.POST)

        if form.is_valid():
            # Валидная форма - создаем пользователя и логиним
            user = form.save()
            auth_login(request, user)
            return redirect('home')
        else:
            # Ошибки валидации - показываем форму с ошибками
                return
self.render_to_response(self.get_context_data(form=form))
class ProfilePageView(TemplateView):
    """
    HTML-страница профиля пользователя
    Отображает информацию о текущем авторизованном пользователе
    """
          template_name = "registration/profile.html"

accounts/serializers.py
      Сериализаторы для API-эндпоинтов

from rest_framework import serializers
from .models import CustomUser
      from django.contrib.auth.password_validation import validate_password
class UserSerializer(serializers.ModelSerializer):
    """
    Сериализатор для отображения основной информации о пользователе
    Используется для ответов API без конфиденциальных данных
    """
```

```
    class Meta:
        model = CustomUser
```



Продолжение листинга В.1

```
            fields = ['id', 'username', 'email', 'role']  # Исключаем пароль
class RegisterSerializer(serializers.ModelSerializer):
    """
    Сериализатор для регистрации новых пользователей
    Включает валидацию паролей и создание пользователя
    """
    # Поле пароля с валидацией (только для записи)
    password = serializers.CharField(
  write_only=True,
        required=True,
        validators=[validate_password]  # Стандартная валидация Django
    )

    # Поле подтверждения пароля
    password2 = serializers.CharField(write_only=True, required=True)

    class Meta:
        model = CustomUser
        fields = ['username', 'password', 'password2', 'email', 'role']

    def validate(self, attrs):
        """
        Валидация на уровне объекта
        Проверяет совпадение паролей
        """
        if attrs['password'] != attrs['password2']:
            raise serializers.ValidationError({
                "password": "Пароли должны совпадать."
            })
        return attrs

    def create(self, validated_data):
        """
        Создание нового пользователя
        Удаляет password2 и правильно хеширует пароль
        """
        # Удаляем поле подтверждения пароля
        validated_data.pop('password2', None)

        # Создаем пользователя без сохранения пароля
        user = CustomUser.objects.create(
            username=validated_data['username'],
            email=validated_data['email'],
            role=validated_data.get('role', CustomUser.CANDIDATE)
        )

        # Правильно хешируем и сохраняем пароль
        user.set_password(validated_data['password'])
        user.save()
              return user

accounts/forms.py
      Django-формы для HTML-интерфейса
```

```
from django import forms
      from .models import CustomUser
```



Продолжение листинга В.1

```
class RegistrationForm(forms.ModelForm):
    """
    Форма регистрации для HTML-интерфейса
    Включает валидацию и создание пользователей
    """
    # Поля паролей с виджетом для скрытия ввода
    password = forms.CharField(widget=forms.PasswordInput, label="Пароль")
          password2       =       forms.CharField(widget=forms.PasswordInput,
label="Повторите пароль")
    class Meta:
  model = CustomUser
              fields = ['username', 'email', 'role', 'password', 'password2']
    def clean(self):
        """
        Валидация формы на уровне объекта
        Проверяет совпадение введенных паролей
        """
        cleaned_data = super().clean()
        password1 = cleaned_data.get("password")
        password2 = cleaned_data.get("password2")

        if password1 and password2 and password1 != password2:
            raise forms.ValidationError("Пароли не совпадают")

              return cleaned_data
    def save(self, commit=True):
        """
        Сохранение пользователя с правильным хешированием пароля
        """
        user = super().save(commit=False)
        # Используем set_password для хеширования
        user.set_password(self.cleaned_data["password"])

        if commit:
            user.save()
              return user

accounts/urls.py
      URL-маршруты для всех представлений модуля

from django.urls import path
from .views import (
    LoginPageView, RegisterPageView, ProfilePageView,
    AdminRegisterPageView, UserListView, InviteUserView,
    CandidateProfileDetailView, RegisterAPIView,
    LoginAPIView, LogoutAPIView, UserUpdateView,
      )
# Комбинированные маршруты для HTML и API
urlpatterns = [
    # HTML-страницы аутентификации
    path('login/', LoginPageView.as_view(), name='login'),
    path('register/', RegisterPageView.as_view(), name='signup'),
    path('admin_register/',                  AdminRegisterPageView.as_view(),
name='admin_signup'),
    path('users/', UserListView.as_view(), name='users_list'),
```

```
    path('invite/', InviteUserView.as_view(), name='invite_user'),
    path('profile/', ProfilePageView.as_view(), name='profile'),
```



Продолжение листинга В.1

```
    path('candidate/<int:pk>/',          CandidateProfileDetailView.as_view(),
name='candidate_profile'),

    # API-эндпоинты
    path('api/register/', RegisterAPIView.as_view(), name='signup_api'),
    path('api/login/', LoginAPIView.as_view(), name='login_api'),
    path('api/logout/', LogoutAPIView.as_view(), name='logout_api'),
    path('user/<int:pk>/update/',                    UserUpdateView.as_view(),
name='user_update'),
      ]

accounts/api_urls.py
      Отдельные API-маршруты для подключения в основном urls.py

from django.urls import path
      from .views import RegisterAPIView, LoginAPIView, LogoutAPIView
# Только API-эндпоинты для аутентификации
urlpatterns = [
    path('register/', RegisterAPIView.as_view(), name='signup_api'),
    path('login/', LoginAPIView.as_view(), name='login_api'),
    path('logout/', LogoutAPIView.as_view(), name='logout_api'),
      ]

accounts/html_urls.py
      Отдельные HTML-маршруты для подключения в основном urls.py

from django.urls import path
      from .views import LoginPageView, RegisterPageView, ProfilePageView
# Только HTML-страницы аутентификации
urlpatterns = [
    path('login/', LoginPageView.as_view(), name='login'),
    path('register/', RegisterPageView.as_view(), name='signup'),
    path('profile/', ProfilePageView.as_view(), name='profile'),
      ]

accounts/admin.py
      Конфигурация административной панели

from django.contrib import admin
from django.contrib.auth.admin import UserAdmin
      from .models import CustomUser
class CustomUserAdmin(UserAdmin):
    """
    Кастомизация административной панели для CustomUser
    Добавляет поле role в отображение и редактирование
    """
    # Отображаемые поля в списке пользователей
    list_display = ('username', 'email', 'role', 'is_staff')

    # Добавляем поле role в форму редактирования
    fieldsets = UserAdmin.fieldsets + (
        (None, {'fields': ('role',)}),
          )
# Регистрируем кастомную модель в админке
```

```
      admin.site.register(CustomUser, CustomUserAdmin)
```



Продолжение листинга В.1

```
accounts/decorators.py
      Декораторы для контроля доступа по ролям

from django.http import HttpResponseForbidden
      from functools import wraps
def role_required(allowed_roles):
    """
    Декоратор для ограничения доступа к представлениям по ролям

    Args:
        allowed_roles: список разрешенных ролей

    Usage:
        @role_required(['admin', 'hr_manager'])
        def my_view(request):
            return HttpResponse("Только для админов и HR-менеджеров")
    """
    def decorator(view_func):
        @wraps(view_func)
        def _wrapped_view(request, *args, kwargs):
            # Проверяем аутентификацию
            if not request.user.is_authenticated:
                return HttpResponseForbidden("Требуется авторизация.")

            # Проверяем роль пользователя
            if request.user.role not in allowed_roles:
                return HttpResponseForbidden(
                    "У вас нет прав для доступа к этой странице."
                )

            # Вызываем оригинальное представление
            return view_func(request, *args, kwargs)
        return _wrapped_view
          return decorator

accounts/apps.py
      Конфигурация Django-приложения

      from django.apps import AppConfig
class AccountsConfig(AppConfig):
    """
    Конфигурация приложения accounts
    Определяет настройки автополей и имя приложения
    """
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'accounts'

      # Модуль employees - Управление сотрудниками
      *Модуль для управления информацией о сотрудниках компании*
employees/models.py
      Модель сотрудника с полной информацией

from django.db import models
      from accounts.models import CustomUser
```

```
class Employee(models.Model):
    """
```



Продолжение листинга В.1

```
    Модель сотрудника компании
    Связана с пользователем через OneToOne для хранения профессиональной
информации
    """

    # Связь один-к-одному с пользователем системы
    # При удалении пользователя - удаляется и профиль сотрудника
    user = models.OneToOneField(
        CustomUser,
        on_delete=models.CASCADE,
        related_name='employee_profile'
    )

    # Основная информация о сотруднике
    full_name = models.CharField(max_length=255, verbose_name="ФИО")
  date_hired = models.DateField(verbose_name="Дата приема на работу")
    position = models.CharField(max_length=255, verbose_name="Должность")
    qualification = models.TextField(verbose_name="Квалификация и навыки")
    department = models.CharField(max_length=255, verbose_name="Отдел")

    # История изменений и дополнительная информация
    history = models.TextField(
        blank=True,
        null=True,
        verbose_name="История изменений"
          )
    class Meta:
        verbose_name = "Сотрудник"
        verbose_name_plural = "Сотрудники"
              ordering = ['full_name']  # Сортировка по умолчанию по ФИО
    def __str__(self):
        """Строковое представление сотрудника"""
              return self.full_name
    def get_years_of_service(self):
        """
        Вычисляет стаж работы сотрудника в годах
        """
        from datetime import date
        today = date.today()
        return today.year - self.date_hired.year - (
            (today.month,      today.day)      <      (self.date_hired.month,
self.date_hired.day)
              )

employees/views.py
      API-представления для работы с сотрудниками

from rest_framework import generics
from .models import Employee
      from .serializers import EmployeeSerializer
class EmployeeListCreateView(generics.ListCreateAPIView):
    """
    API-представление для списка сотрудников и создания новых
```

```
    GET: Возвращает список всех сотрудников
    POST: Создает нового сотрудника
```



Продолжение листинга В.1

```
    """
    queryset = Employee.objects.all()
    serializer_class = EmployeeSerializer

    def get_queryset(self):
        """
        Может фильтровать сотрудников по различным параметрам
        """
        queryset = Employee.objects.all()

        # Пример фильтрации по отделу
        department = self.request.query_params.get('department', None)
        if department is not None:
            queryset = queryset.filter(department__icontains=department)

        # Пример фильтрации по должности
        position = self.request.query_params.get('position', None)
    if position is not None:
            queryset = queryset.filter(position__icontains=position)

              return queryset
class EmployeeDetailView(generics.RetrieveUpdateDestroyAPIView):
    """
    API-представление для детальной работы с конкретным сотрудником

    GET: Получить информацию о сотруднике
    PUT/PATCH: Обновить информацию о сотруднике
    DELETE: Удалить сотрудника
    """
    queryset = Employee.objects.all()
    serializer_class = EmployeeSerializer

    def perform_update(self, serializer):
        """
        Дополнительная логика при обновлении сотрудника
        Например, логирование изменений
        """
        # Получаем старые данные для истории
        old_instance = Employee.objects.get(pk=self.get_object().pk)

        # Сохраняем обновленные данные
        serializer.save()

        # Можно добавить логику для отслеживания изменений
        # Например, записать в историю изменений
        new_instance = serializer.instance
        if old_instance.position != new_instance.position:
            history_entry = f"Изменена должность с '{old_instance.position}' на
'{new_instance.position}'"
            if new_instance.history:
                new_instance.history += f"\n{history_entry}"
            else:
                new_instance.history = history_entry
                  new_instance.save()
```

Продолжение листинга В.1

```
      Сериализаторы для API работы с сотрудниками

from rest_framework import serializers
from .models import Employee
      from accounts.serializers import UserSerializer
class EmployeeSerializer(serializers.ModelSerializer):
    """
    Сериализатор для модели Employee
    Включает все поля сотрудника и информацию о связанном пользователе
    """

    # Вложенный сериализатор для отображения информации о пользователе
    user_info = UserSerializer(source='user', read_only=True)

    # Поле для отображения стажа работы (только чтение)
    years_of_service = serializers.SerializerMethodField()

    class Meta:
        model = Employee
    fields = [
            'id', 'user', 'user_info', 'full_name',
            'date_hired', 'position', 'qualification',
            'department', 'history', 'years_of_service'
        ]

    def get_years_of_service(self, obj):
        """
        Вычисляет и возвращает стаж работы сотрудника
        """
        return obj.get_years_of_service()

    def validate_date_hired(self, value):
        """
        Валидация даты приема на работу
        Проверяет, что дата не в будущем
        """
        from datetime import date
        if value > date.today():
            raise serializers.ValidationError(
                "Дата приема на работу не может быть в будущем"
            )
        return value

    def validate_full_name(self, value):
        """
        Валидация ФИО
        Проверяет минимальную длину и отсутствие цифр
        """
        if len(value.strip()) < 2:
            raise serializers.ValidationError(
                "ФИО должно содержать минимум 2 символа"
            )

        if any(char.isdigit() for char in value):
```

```
            raise serializers.ValidationError(
                "ФИО не должно содержать цифры"
```



Продолжение листинга В.1

```
            )

              return value.strip()

employees/urls.py
      URL-маршруты для API сотрудников

from django.urls import path
      from .views import EmployeeListCreateView, EmployeeDetailView
# API-маршруты для работы с сотрудниками
urlpatterns = [
    # Список сотрудников и создание нового
    # GET /api/employees/ - список всех сотрудников
    # POST /api/employees/ - создание нового сотрудника
    path('', EmployeeListCreateView.as_view(), name='employees'),

    # Работа с конкретным сотрудником
    # GET /api/employees/{id}/ - информация о сотруднике
    # PUT /api/employees/{id}/ - полное обновление сотрудника
    # PATCH /api/employees/{id}/ - частичное обновление сотрудника
    # DELETE /api/employees/{id}/ - удаление сотрудника
path('<int:pk>/', EmployeeDetailView.as_view(), name='employee-detail'),
      ]

employees/forms.py
      Django-формы для HTML-интерфейса управления сотрудниками

from django import forms
from .models import Employee
      from accounts.models import CustomUser
class EmployeeForm(forms.ModelForm):
    """
    Форма для создания и редактирования сотрудников
    Используется в HTML-интерфейсе администрирования
    """

    class Meta:
        model = Employee
        fields = [
            'user', 'full_name', 'date_hired',
            'position', 'qualification', 'department'
        ]

        # Человекочитаемые названия полей
        labels = {
            'user': 'Пользователь',
            'full_name': 'ФИО',
            'date_hired': 'Дата приёма',
            'position': 'Должность',
            'qualification': 'Квалификация',
            'department': 'Отдел',
        }

        # Кастомные виджеты для улучшения UX
```

```
    widgets = {
        # HTML5 виджет для выбора даты
        'date_hired': forms.DateInput(attrs={
            'type': 'date',
            'class': 'form-control'
        }),

        # Текстовая область для квалификации
        'qualification': forms.Textarea(attrs={
            'rows': 3,
            'class': 'form-control',
            'placeholder': 'Опишите квалификацию и навыки сотрудника'
        }),

        # Стилизация остальных полей
        'full_name': forms.TextInput(attrs={
            'class': 'form-control',
            'placeholder': 'Фамилия Имя Отчество'
        }),
        'position': forms.TextInput(attrs={
            'class': 'form-control',
            'placeholder': 'Должность сотрудника'
        }),
        'department': forms.TextInput(attrs={
            'class': 'form-control',
      'placeholder': 'Название отдела'
        }),
    }

def __init__(self, *args, kwargs):
    """
    Инициализация формы с дополнительной настройкой
    """
    super().__init__(*args, kwargs)

    # Фильтруем пользователей только с ролью employee или hr_manager
    self.fields['user'].queryset = CustomUser.objects.filter(
        role__in=[CustomUser.EMPLOYEE, CustomUser.HR_MANAGER]
    )

    # Устанавливаем пустой выбор для пользователя
    self.fields['user'].empty_label = "Выберите пользователя"

def clean_full_name(self):
    """
    Дополнительная валидация ФИО
    """
    full_name = self.cleaned_data.get('full_name')

    if full_name:
        # Проверяем на минимальное количество слов (имя и фамилия)
        words = full_name.strip().split()
        if len(words) < 2:
            raise forms.ValidationError(
                "ФИО должно содержать минимум имя и фамилию"
            )

        # Проверяем на наличие цифр
        if any(char.isdigit() for char in full_name):
            raise forms.ValidationError(
                "ФИО не должно содержать цифры"
```

```
                )
```



Продолжение листинга В.1

```
        return full_name

    def clean_date_hired(self):
        """
        Валидация даты приема на работу
        """
        date_hired = self.cleaned_data.get('date_hired')

        if date_hired:
            from datetime import date
            if date_hired > date.today():
                raise forms.ValidationError(
                    "Дата приема на работу не может быть в будущем"
                )

              return date_hired
```

employees/apps.py

```
      Конфигурация Django-приложения
      from django.apps import AppConfig
class EmployeesConfig(AppConfig):
    """
    Конфигурация приложения employees
    Определяет настройки автополей и имя приложения
    """
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'employees'
    verbose_name = 'Управление сотрудниками'

    def ready(self):
        """
        Выполняется при готовности приложения
        Здесь можно подключить сигналы Django
        """
        # Импортируем сигналы, если они есть
        # import employees.signals
              pass
```

employees/admin.py

```
      Конфигурация административной панели

from django.contrib import admin
      from .models import Employee
@admin.register(Employee)
class EmployeeAdmin(admin.ModelAdmin):
    """
    Конфигурация административного интерфейса для сотрудников
    """

    # Поля, отображаемые в списке сотрудников
    list_display = [
        'full_name', 'position', 'department',
        'date_hired', 'get_user_email'
    ]
```

```
    # Поля для фильтрации в боковой панели
```



Продолжение листинга В.1

```
    list_filter = ['department', 'position', 'date_hired']

    # Поля для поиска
    search_fields = ['full_name', 'position', 'department', 'user__username']

    # Поля только для чтения
    readonly_fields = ['get_years_of_service']

    # Группировка полей в форме редактирования
    fieldsets = (
        ('Основная информация', {
            'fields': ('user', 'full_name', 'date_hired')
        }),
        ('Профессиональная информация', {
            'fields': ('position', 'department', 'qualification')
        }),
        ('Дополнительно', {
            'fields': ('history', 'get_years_of_service'),
            'classes': ('collapse',)  # Свернутая секция
        }),
    )

    def get_user_email(self, obj):
        """
        Отображает email связанного пользователя
        """
        return obj.user.email if obj.user else '-'
    get_user_email.short_description = 'Email'

    def get_years_of_service(self, obj):
        """
        Отображает стаж работы сотрудника
        """
        return f"{obj.get_years_of_service()} лет"
    get_years_of_service.short_description = 'Стаж работы'

      # Модуль vacancies - Управление вакансиями
      *Модуль для управления вакансиями и откликами кандидатов*
vacancies/models.py
     Модели вакансий и откликов

from django.db import models
     from accounts.models import CustomUser
class Vacancy(models.Model):
    """
    Модель вакансии компании
    Содержит всю информацию о доступных позициях
    """

    # Основная информация о вакансии
    title = models.CharField(max_length=255, verbose_name="Название вакансии")
    description = models.TextField(verbose_name="Описание вакансии")
    requirements = models.TextField(verbose_name="Требования к кандидату")
    conditions = models.TextField(verbose_name="Условия работы")
```

Продолжение листинга В.1

```
    # Даты управления вакансией
    posting_date = models.DateField(
        auto_now_add=True,
        verbose_name="Дата публикации"
    )
    closing_date = models.DateField(
        null=True,
        blank=True,
        verbose_name="Дата закрытия"
    )

    # Статус вакансии
    is_archived = models.BooleanField(
        default=False,
        verbose_name="Архивирована"
    )

    class Meta:
        verbose_name = "Вакансия"
        verbose_name_plural = "Вакансии"
        ordering = ['-posting_date']  # Сортировка по дате (новые сначала)

    def __str__(self):
        """Строковое представление вакансии"""
    return self.title

    @property
    def is_active(self):
        """
        Проверяет, активна ли вакансия
        Вакансия активна, если не архивирована и не истекла дата закрытия
        """
        from datetime import date

        if self.is_archived:
            return False

        if self.closing_date and self.closing_date < date.today():
            return False

        return True

    def get_responses_count(self):
        """
        Возвращает количество откликов на вакансию
        """
              return self.responses.count()
class VacancyResponse(models.Model):
    """
    Модель отклика кандидата на вакансию
    Связывает кандидата с вакансией и отслеживает статус рассмотрения
    """

    # Статусы рассмотрения отклика
    STATUS_CHOICES = [
```

```
        ('new', 'Новый'),
        ('review', 'На рассмотрении'),
```



Продолжение листинга В.1

```
        ('interview', 'Собеседование'),
        ('accepted', 'Принят'),
        ('rejected', 'Отклонен'),
    ]

    # Основные связи
    vacancy = models.ForeignKey(
        Vacancy,
        on_delete=models.CASCADE,
        related_name='responses',
        verbose_name="Вакансия"
    )
    candidate = models.ForeignKey(
        CustomUser,
        on_delete=models.CASCADE,
        related_name='vacancy_responses',
        verbose_name="Кандидат"
    )

    # Данные отклика
    resume = models.FileField(
        upload_to='resumes/',
        verbose_name="Резюме"
    )
    cover_letter = models.TextField(
  blank=True,
        verbose_name="Сопроводительное письмо"
    )

    # Управление процессом
    status = models.CharField(
        max_length=20,
        choices=STATUS_CHOICES,
        default='new',
        verbose_name="Статус"
    )
    submitted_at = models.DateTimeField(
        auto_now_add=True,
        verbose_name="Дата подачи"
    )

    # Комментарии HR-менеджера
    hr_comments = models.TextField(
        blank=True,
        verbose_name="Комментарии HR"
    )

    class Meta:
        verbose_name = "Отклик на вакансию"
        verbose_name_plural = "Отклики на вакансии"
        unique_together = ['vacancy', 'candidate']  # Один отклик на вакансию
от кандидата
        ordering = ['-submitted_at']
```

```
    def __str__(self):
        """Строковое представление отклика"""
```



Продолжение листинга В.1

```
        return f"{self.candidate.username} -> {self.vacancy.title}"

    def get_status_display_class(self):
        """
        Возвращает CSS-класс для отображения статуса
        """
        status_classes = {
            'new': 'badge-info',
            'review': 'badge-warning',
            'interview': 'badge-primary',
            'accepted': 'badge-success',
            'rejected': 'badge-danger',
        }
              return status_classes.get(self.status, 'badge-secondary')

vacancies/views.py
      API-представления для работы с вакансиями

from rest_framework import generics, status
from rest_framework.decorators import api_view, permission_classes
from rest_framework.response import Response
from rest_framework.permissions import IsAuthenticated
from django.shortcuts import get_object_or_404
from .models import Vacancy, VacancyResponse
      from .serializers import VacancySerializer, VacancyResponseSerializer
class VacancyListCreateView(generics.ListCreateAPIView):
    """
  API-представление для списка вакансий и создания новых

    GET: Возвращает список активных вакансий
    POST: Создает новую вакансию (только для HR-менеджеров и админов)
    """
    serializer_class = VacancySerializer

    def get_queryset(self):
        """
        Фильтрует вакансии в зависимости от параметров запроса
        """
        queryset = Vacancy.objects.all()

        # Показываем только активные вакансии для обычных пользователей
        show_archived = self.request.query_params.get('archived', 'false')
        if show_archived.lower() != 'true':
            queryset = queryset.filter(is_archived=False)

        # Фильтрация по ключевым словам в названии
        search = self.request.query_params.get('search', None)
        if search:
            queryset = queryset.filter(title__icontains=search)

        return queryset

    def perform_create(self, serializer):
        """
```

Продолжение листинга В.1

```
        Дополнительная логика при создании вакансии
        """
        # Можно добавить автора вакансии
        # serializer.save(created_by=self.request.user)
              serializer.save()
class VacancyDetailView(generics.RetrieveUpdateDestroyAPIView):
    """
    API-представление для работы с конкретной вакансией

    GET: Получить детали вакансии
    PUT/PATCH: Обновить вакансию
    DELETE: Удалить вакансию
    """
    queryset = Vacancy.objects.all()
          serializer_class = VacancySerializer
class VacancyResponseListCreateView(generics.ListCreateAPIView):
    """
    API-представление для откликов на вакансии

    GET: Список откликов (для HR) или откликов пользователя
    POST: Создать новый отклик на вакансию
    """
    serializer_class = VacancyResponseSerializer
    permission_classes = [IsAuthenticated]

    def get_queryset(self):
        """
        Возвращает отклики в зависимости от роли пользователя
        """
        user = self.request.user

  # HR-менеджеры и админы видят все отклики
        if user.role in ['admin', 'hr_manager']:
            return VacancyResponse.objects.all()

        # Обычные пользователи видят только свои отклики
        return VacancyResponse.objects.filter(candidate=user)

    def perform_create(self, serializer):
        """
        Автоматически устанавливает кандидата как текущего пользователя
        """
              serializer.save(candidate=self.request.user)
class VacancyResponseDetailView(generics.RetrieveUpdateDestroyAPIView):
    """
    API-представление для работы с конкретным откликом
    """
    queryset = VacancyResponse.objects.all()
    serializer_class = VacancyResponseSerializer
    permission_classes = [IsAuthenticated]

    def get_queryset(self):
        """
        Фильтрует отклики по правам доступа
        """
        user = self.request.user
```

```
        # HR-менеджеры и админы видят все отклики
        if user.role in ['admin', 'hr_manager']:
            return VacancyResponse.objects.all()

        # Пользователи видят только свои отклики
              return VacancyResponse.objects.filter(candidate=user)
@api_view(['POST'])
@permission_classes([IsAuthenticated])
def respond_to_vacancy(request, vacancy_id):
    """
    Специальный эндпоинт для отклика на вакансию
    Упрощенный интерфейс для подачи отклика
    """
    vacancy = get_object_or_404(Vacancy, id=vacancy_id)

    # Проверяем, что вакансия активна
    if not vacancy.is_active:
        return Response(
            {'error': 'Вакансия неактивна или закрыта'},
            status=status.HTTP_400_BAD_REQUEST
        )

    # Проверяем, не подавал ли пользователь уже отклик
    existing_response = VacancyResponse.objects.filter(
        vacancy=vacancy,
        candidate=request.user
    ).first()

    if existing_response:
        return Response(
            {'error': 'Вы уже откликнулись на эту вакансию'},
            status=status.HTTP_400_BAD_REQUEST
    )

    # Создаем отклик
    serializer = VacancyResponseSerializer(data=request.data)
    if serializer.is_valid():
        serializer.save(
            vacancy=vacancy,
            candidate=request.user
        )
        return Response(serializer.data, status=status.HTTP_201_CREATED)

          return                                    Response(serializer.errors,
status=status.HTTP_400_BAD_REQUEST)
@api_view(['PATCH'])
@permission_classes([IsAuthenticated])
def update_response_status(request, response_id):
    """
    Эндпоинт для обновления статуса отклика (только для HR)
    """
    if request.user.role not in ['admin', 'hr_manager']:
        return Response(
            {'error': 'Недостаточно прав'},
            status=status.HTTP_403_FORBIDDEN
        )

    response_obj = get_object_or_404(VacancyResponse, id=response_id)

    new_status = request.data.get('status')
```

```
    hr_comments = request.data.get('hr_comments', '')
```



Продолжение листинга В.1

```
    if new_status not in dict(VacancyResponse.STATUS_CHOICES):
        return Response(
            {'error': 'Неверный статус'},
            status=status.HTTP_400_BAD_REQUEST
        )

    response_obj.status = new_status
    response_obj.hr_comments = hr_comments
    response_obj.save()

    return Response({
        'message': 'Статус обновлен',
        'status': response_obj.get_status_display()
          })

vacancies/serializers.py
      Сериализаторы для API работы с вакансиями

from rest_framework import serializers
from .models import Vacancy, VacancyResponse
      from accounts.serializers import UserSerializer
class VacancySerializer(serializers.ModelSerializer):
    """
    Сериализатор для модели Vacancy
    Включает дополнительные вычисляемые поля
    """

    # Вычисляемые поля только для чтения
  is_active = serializers.ReadOnlyField()
    responses_count = serializers.SerializerMethodField()
    days_since_posting = serializers.SerializerMethodField()

    class Meta:
        model = Vacancy
        fields = [
            'id', 'title', 'description', 'requirements',
            'conditions', 'posting_date', 'closing_date',
            'is_archived', 'is_active', 'responses_count',
            'days_since_posting'
        ]

    def get_responses_count(self, obj):
        """Возвращает количество откликов на вакансию"""
        return obj.get_responses_count()

    def get_days_since_posting(self, obj):
        """Возвращает количество дней с момента публикации"""
        from datetime import date
        delta = date.today() - obj.posting_date
        return delta.days

    def validate_closing_date(self, value):
        """
        Валидация даты закрытия вакансии
```

```
        """
        if value:
```



Продолжение листинга В.1

```
            from datetime import date
            if value < date.today():
                raise serializers.ValidationError(
                    "Дата закрытия не может быть в прошлом"
                )
        return value

    def validate_title(self, value):
        """
        Валидация названия вакансии
        """
        if len(value.strip()) < 5:
            raise serializers.ValidationError(
                "Название вакансии должно содержать минимум 5 символов"
            )
              return value.strip()
class VacancyResponseSerializer(serializers.ModelSerializer):
    """
    Сериализатор для откликов на вакансии
    """

    # Вложенные сериализаторы для отображения связанных данных
    candidate_info = UserSerializer(source='candidate', read_only=True)
    vacancy_info = VacancySerializer(source='vacancy', read_only=True)

    # Человекочитаемое отображение статуса
    status_display = serializers.CharField(
        source='get_status_display',
        read_only=True
    )

    # CSS-класс для статуса
    status_class = serializers.CharField(
        source='get_status_display_class',
        read_only=True
    )

    class Meta:
        model = VacancyResponse
        fields = [
            'id', 'vacancy', 'vacancy_info', 'candidate',
            'candidate_info', 'resume', 'cover_letter',
            'status', 'status_display', 'status_class',
            'submitted_at', 'hr_comments'
        ]
        read_only_fields = ['candidate', 'submitted_at']

    def validate_resume(self, value):
        """
        Валидация загружаемого резюме
        """
        # Проверяем размер файла (максимум 5MB)
        if value.size > 5 * 1024 * 1024:
            raise serializers.ValidationError(
```

```
                "Размер файла резюме не должен превышать 5MB"
            )
```



Продолжение листинга В.1

```
        # Проверяем расширение файла
        allowed_extensions = ['.pdf', '.doc', '.docx']
        file_extension = value.name.lower().split('.')[-1]
        if f'.{file_extension}' not in allowed_extensions:
            raise serializers.ValidationError(
                "Допустимые форматы резюме: PDF, DOC, DOCX"
            )

        return value

    def validate_cover_letter(self, value):
        """
        Валидация сопроводительного письма
        """
        if value and len(value.strip()) < 50:
            raise serializers.ValidationError(
                "Сопроводительное письмо должно содержать минимум 50 символов"
            )
        return value

    def validate(self, attrs):
        """
        Валидация на уровне объекта
        """
        # Проверяем, что кандидат не подает повторный отклик
        if not self.instance:  # Только для создания, не для обновления
            vacancy = attrs.get('vacancy')
            candidate = self.context['request'].user

            existing_response = VacancyResponse.objects.filter(
                vacancy=vacancy,
              candidate=candidate
            ).exists()

            if existing_response:
                raise serializers.ValidationError(
                    "Вы уже откликнулись на эту вакансию"
                )

              return attrs
class VacancyResponseCreateSerializer(serializers.ModelSerializer):
    """
    Упрощенный сериализатор для создания откликов
    Используется в специальных эндпоинтах
    """
    class Meta:
        model = VacancyResponse
        fields = ['resume', 'cover_letter']

    def validate_resume(self, value):
        """Валидация резюме для упрощенного сериализатора"""
        if value.size > 5 * 1024 * 1024:
            raise serializers.ValidationError(
```

```
                "Размер файла резюме не должен превышать 5MB"
            )
```



Продолжение листинга В.1

```
            return value

vacancies/urls.py
      URL-маршруты для API вакансий

from django.urls import path
from .views import (
    VacancyListCreateView, VacancyDetailView,
    VacancyResponseListCreateView, VacancyResponseDetailView,
    respond_to_vacancy, update_response_status
      )
# API-маршруты для работы с вакансиями и откликами
urlpatterns = [
    # Основные CRUD операции для вакансий
    # GET /api/vacancies/ - список вакансий
    # POST /api/vacancies/ - создание новой вакансии
    path('', VacancyListCreateView.as_view(), name='vacancies'),

    # Работа с конкретной вакансией
    # GET /api/vacancies/{id}/ - детали вакансии
    # PUT/PATCH /api/vacancies/{id}/ - обновление вакансии
    # DELETE /api/vacancies/{id}/ - удаление вакансии
    path('<int:pk>/', VacancyDetailView.as_view(), name='vacancy-detail'),

    # Работа с откликами на вакансии
    # GET /api/vacancies/responses/ - список откликов
    # POST /api/vacancies/responses/ - создание отклика
    path('responses/', VacancyResponseListCreateView.as_view(), name='vacancy-
responses'),

    # Работа с конкретным откликом
    # GET /api/vacancies/responses/{id}/ - детали отклика
    # PUT/PATCH /api/vacancies/responses/{id}/ - обновление отклика
    # DELETE /api/vacancies/responses/{id}/ - удаление отклика
  path('responses/<int:pk>/',             VacancyResponseDetailView.as_view(),
name='vacancy-response-detail'),

    # Специальные эндпоинты
    # POST /api/vacancies/{id}/respond/ - упрощенный отклик на вакансию
    path('<int:vacancy_id>/respond/',  respond_to_vacancy,  name='respond-to-
vacancy'),

    # PATCH /api/vacancies/responses/{id}/status/ - обновление статуса отклика
(для HR)
    path('responses/<int:response_id>/status/',        update_response_status,
name='update-response-status'),
      ]

vacancies/forms.py
      Django-формы для HTML-интерфейса

from django import forms
      from .models import Vacancy, VacancyResponse
class VacancyForm(forms.ModelForm):
    """
```

```
    Форма для создания и редактирования вакансий
    """
```



Продолжение листинга В.1

```
    class Meta:
        model = Vacancy
        fields = [
            'title', 'description', 'requirements',
            'conditions', 'closing_date', 'is_archived'
        ]

        # Человекочитаемые названия полей
        labels = {
            'title': 'Название',
            'description': 'Описание',
            'requirements': 'Требования',
            'conditions': 'Условия',
            'closing_date': 'Дата закрытия',
            'is_archived': 'Архивировано'
        }

        # Кастомные виджеты для улучшения UX
        widgets = {
            'closing_date': forms.DateInput(attrs={
                'type': 'date',
                'class': 'form-control'
            }),
            'description': forms.Textarea(attrs={
                'rows': 4,
                'class': 'form-control',
                'placeholder': 'Подробное описание вакансии'
            }),
            'requirements': forms.Textarea(attrs={
                'rows': 4,
                'class': 'form-control',
                'placeholder': 'Требования к кандидату'
            }),
            'conditions': forms.Textarea(attrs={
                'rows': 4,
          'class': 'form-control',
                'placeholder': 'Условия работы и компенсация'
            }),
            'title': forms.TextInput(attrs={
                'class': 'form-control',
                'placeholder': 'Название должности'
            }),
        }

    def clean_title(self):
        """Валидация названия вакансии"""
        title = self.cleaned_data.get('title')
        if title and len(title.strip()) < 5:
            raise forms.ValidationError(
                "Название вакансии должно содержать минимум 5 символов"
            )
        return title.strip() if title else title

    def clean_closing_date(self):
```

```
        """Валидация даты закрытия"""
        closing_date = self.cleaned_data.get('closing_date')
```



Продолжение листинга В.1

```
        if closing_date:
            from datetime import date
            if closing_date < date.today():
                raise forms.ValidationError(
                    "Дата закрытия не может быть в прошлом"
                )
              return closing_date
class VacancyResponseForm(forms.ModelForm):
    """
    Форма для подачи отклика на вакансию
    """
    class Meta:
        model = VacancyResponse
        fields = ['vacancy', 'resume', 'cover_letter']

        labels = {
            'vacancy': 'Вакансия',
            'resume': 'Резюме',
            'cover_letter': 'Сопроводительное письмо'
        }

        widgets = {
            'cover_letter': forms.Textarea(attrs={
                'rows': 5,
                'class': 'form-control',
                'placeholder': 'Расскажите о себе и почему вы подходите для
этой позиции'
            }),
            'resume': forms.FileInput(attrs={
                'class': 'form-control',
                'accept': '.pdf,.doc,.docx'
            })
        }

    def __init__(self, *args, kwargs):
        """
        Инициализация формы с фильтрацией активных вакансий
    """
        super().__init__(*args, kwargs)

        # Показываем только активные вакансии
        self.fields['vacancy'].queryset = Vacancy.objects.filter(
            is_archived=False
        )

        # Устанавливаем пустой выбор
        self.fields['vacancy'].empty_label = "Выберите вакансию"

    def clean_resume(self):
        """Валидация загружаемого резюме"""
        resume = self.cleaned_data.get('resume')

        if resume:
            # Проверяем размер файла (максимум 5MB)
```

```
            if resume.size > 5 * 1024 * 1024:
                raise forms.ValidationError(
```



Продолжение листинга В.1

```
                    "Размер файла резюме не должен превышать 5MB"
                )

            # Проверяем расширение файла
            allowed_extensions = ['.pdf', '.doc', '.docx']
            file_extension = resume.name.lower().split('.')[-1]
            if f'.{file_extension}' not in allowed_extensions:
                raise forms.ValidationError(
                    "Допустимые форматы резюме: PDF, DOC, DOCX"
                )

        return resume

    def clean_cover_letter(self):
        """Валидация сопроводительного письма"""
        cover_letter = self.cleaned_data.get('cover_letter')

        if cover_letter and len(cover_letter.strip()) < 50:
            raise forms.ValidationError(
                "Сопроводительное письмо должно содержать минимум 50 символов"
            )

              return cover_letter.strip() if cover_letter else cover_letter
class VacancyResponseStatusForm(forms.ModelForm):
    """
    Форма для HR-менеджеров для обновления статуса отклика
    """
    class Meta:
        model = VacancyResponse
        fields = ['status', 'hr_comments']

        labels = {
            'status': 'Статус рассмотрения',
            'hr_comments': 'Комментарии HR'
        }

        widgets = {
            'status': forms.Select(attrs={
                'class': 'form-control'
    }),
            'hr_comments': forms.Textarea(attrs={
                'rows': 3,
                'class': 'form-control',
                'placeholder': 'Комментарии по результатам рассмотрения'
            })
              }

vacancies/apps.py
      Конфигурация Django-приложения

      from django.apps import AppConfig
class VacanciesConfig(AppConfig):
    """
    Конфигурация приложения vacancies
```

```
    Определяет настройки автополей и имя приложения
    """
```



Продолжение листинга В.1

```
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'vacancies'
    verbose_name = 'Управление вакансиями'

    def ready(self):
        """
        Выполняется при готовности приложения
        Здесь можно подключить сигналы Django
        """
        # Пример подключения сигналов для уведомлений
        # import vacancies.signals
              pass

vacancies/admin.py
      Конфигурация административной панели

from django.contrib import admin
      from .models import Vacancy, VacancyResponse
@admin.register(Vacancy)
class VacancyAdmin(admin.ModelAdmin):
    """
    Конфигурация административного интерфейса для вакансий
    """

    # Поля для отображения в списке
    list_display = [
        'title', 'posting_date', 'closing_date',
        'is_archived', 'get_responses_count'
    ]

    # Фильтры в боковой панели
    list_filter = ['is_archived', 'posting_date']

    # Поля для поиска
    search_fields = ['title', 'description']

    # Поля только для чтения
    readonly_fields = ['posting_date', 'get_responses_count']

    # Действия с группой объектов
    actions = ['archive_vacancies', 'unarchive_vacancies']
    def get_responses_count(self, obj):
        """Отображает количество откликов"""
        return obj.get_responses_count()
    get_responses_count.short_description = 'Откликов'

    def archive_vacancies(self, request, queryset):
        """Действие для архивирования вакансий"""
        updated = queryset.update(is_archived=True)
        self.message_user(request, f'Архивировано {updated} вакансий.')
    archive_vacancies.short_description = 'Архивировать выбранные вакансии'

    def unarchive_vacancies(self, request, queryset):
        """Действие для разархивирования вакансий"""
```

```
        updated = queryset.update(is_archived=False)
        self.message_user(request, f'Разархивировано {updated} вакансий.')
```



Продолжение листинга В.1

```
          unarchive_vacancies.short_description = 'Разархивировать выбранные
вакансии'
@admin.register(VacancyResponse)
class VacancyResponseAdmin(admin.ModelAdmin):
    """
    Конфигурация административного интерфейса для откликов
    """

    # Поля для отображения в списке
    list_display = [
        'get_candidate_name', 'get_vacancy_title',
        'status', 'submitted_at'
    ]

    # Фильтры в боковой панели
    list_filter = ['status', 'submitted_at', 'vacancy__title']

    # Поля для поиска
    search_fields = [
        'candidate__username', 'candidate__email',
        'vacancy__title'
    ]

    # Поля только для чтения
    readonly_fields = ['submitted_at']

    # Группировка полей в форме
    fieldsets = (
        ('Основная информация', {
            'fields': ('vacancy', 'candidate', 'submitted_at')
        }),
        ('Документы', {
            'fields': ('resume', 'cover_letter')
        }),
        ('Рассмотрение', {
            'fields': ('status', 'hr_comments')
        }),
    )

    def get_candidate_name(self, obj):
        """Отображает имя кандидата"""
        return obj.candidate.username
  get_candidate_name.short_description = 'Кандидат'

    def get_vacancy_title(self, obj):
        """Отображает название вакансии"""
        return obj.vacancy.title
    get_vacancy_title.short_description = 'Вакансия'

      # Модуль documents - Управление документами
      *Модуль для управления шаблонами документов и документами сотрудников*
documents/models.py
      Модели для документооборота
```

```
from django.db import models
      from employees.models import Employee
```



Продолжение листинга В.1

```
class DocumentTemplate(models.Model):
    """
    Модель шаблона документа
    Хранит типовые шаблоны документов для создания персонализированных
документов
    """

    # Основная информация о шаблоне
    name = models.CharField(
        max_length=255,
        verbose_name="Название шаблона"
    )
    content = models.TextField(
        verbose_name="Содержание шаблона"
    )
    created_at = models.DateTimeField(
        auto_now_add=True,
        verbose_name="Дата создания"
          )
    class Meta:
        verbose_name = "Шаблон документа"
        verbose_name_plural = "Шаблоны документов"
              ordering = ['-created_at']  # Сортировка по дате создания (новые
сначала)
    def __str__(self):
        """Строковое представление шаблона"""
        return self.name

    def get_formatted_content(self, employee=None):
        """
        Возвращает контент шаблона с подставленными данными сотрудника
        Использует простые плейсхолдеры вида {field_name}
        """
        if not employee:
            return self.content

        # Словарь для замены плейсхолдеров
        replacements = {
            '{full_name}': employee.full_name,
            '{position}': employee.position,
            '{department}': employee.department,
            '{date_hired}': str(employee.date_hired),
            '{qualification}': employee.qualification,
        }
        formatted_content = self.content
        for placeholder, value in replacements.items():
            formatted_content = formatted_content.replace(placeholder, value)

              return formatted_content
class EmployeeDocument(models.Model):
    """
    Модель документа сотрудника
    Связывает конкретный документ с сотрудником и может быть основан на шаблоне
    """
```

```
    # Связи с другими моделями
```



Продолжение листинга В.1

```
    employee = models.ForeignKey(
        Employee,
        on_delete=models.CASCADE,
        related_name='documents',
        verbose_name="Сотрудник"
    )
    template = models.ForeignKey(
        DocumentTemplate,
        on_delete=models.SET_NULL,
        null=True,
        blank=True,
        verbose_name="Шаблон документа"
    )

    # Файл документа
    file = models.FileField(
        upload_to='employee_documents/',
        verbose_name="Файл документа"
    )

    # Метаданные документа
    signed_at = models.DateTimeField(
        auto_now_add=True,
        verbose_name="Дата подписания/создания"
    )

    class Meta:
        verbose_name = "Документ сотрудника"
        verbose_name_plural = "Документы сотрудников"
        ordering = ['-signed_at']  # Сортировка по дате (новые сначала)

    def __str__(self):
        """Строковое представление документа"""
        return f"Документ {self.id} для {self.employee.full_name}"

    def get_file_extension(self):
        """
        Возвращает расширение файла документа
        """
        if self.file and self.file.name:
            return self.file.name.split('.')[-1].lower()
        return None

    def get_file_size_mb(self):
        """
  Возвращает размер файла в мегабайтах
        """
        if self.file and hasattr(self.file, 'size'):
            return round(self.file.size / (1024 * 1024), 2)
              return 0

documents/views.py
      API-представления для работы с документами
```

```
from rest_framework import generics, status
from rest_framework.decorators import api_view, permission_classes
```



Продолжение листинга В.1

```
from rest_framework.response import Response
from rest_framework.permissions import IsAuthenticated
from django.shortcuts import get_object_or_404
from django.http import HttpResponse, Http404
from .models import DocumentTemplate, EmployeeDocument
from .serializers import DocumentTemplateSerializer,
EmployeeDocumentSerializer
    from employees.models import Employee
class DocumentTemplateListCreateView(generics.ListCreateAPIView):
    """
    API-представление для работы со списком шаблонов документов

    GET: Возвращает список всех шаблонов документов
    POST: Создает новый шаблон документа (только для админов и HR)
    """
    queryset = DocumentTemplate.objects.all()
    serializer_class = DocumentTemplateSerializer

    def get_queryset(self):
        """
        Фильтрует шаблоны по поисковому запросу если он передан
        """
        queryset = DocumentTemplate.objects.all()

        # Поиск по названию шаблона
        search = self.request.query_params.get('search', None)
        if search:
            queryset = queryset.filter(name__icontains=search)

            return queryset
class DocumentTemplateDetailView(generics.RetrieveUpdateDestroyAPIView):
    """
    API-представление для работы с конкретным шаблоном документа

    GET: Получить детали шаблона
    PUT/PATCH: Обновить шаблон
    DELETE: Удалить шаблон
    """
    queryset = DocumentTemplate.objects.all()
        serializer_class = DocumentTemplateSerializer
class EmployeeDocumentListCreateView(generics.ListCreateAPIView):
    """
    API-представление для работы с документами сотрудников

    GET: Список документов (фильтруется по правам доступа)
    POST: Создать новый документ для сотрудника
    """
  serializer_class = EmployeeDocumentSerializer
    permission_classes = [IsAuthenticated]

    def get_queryset(self):
        """
        Возвращает документы в зависимости от роли пользователя
        """
```

```
        user = self.request.user
```



Продолжение листинга В.1

```
        # Админы и HR видят все документы
        if user.role in ['admin', 'hr_manager']:
            queryset = EmployeeDocument.objects.all()
        # Сотрудники видят только свои документы
        elif hasattr(user, 'employee_profile'):
            queryset = EmployeeDocument.objects.filter(
                employee=user.employee_profile
            )
        else:
            # Пользователи без профиля сотрудника не видят документы
            queryset = EmployeeDocument.objects.none()

        # Фильтрация по сотруднику (для HR и админов)
        employee_id = self.request.query_params.get('employee', None)
        if employee_id and user.role in ['admin', 'hr_manager']:
            queryset = queryset.filter(employee_id=employee_id)

              return queryset
class EmployeeDocumentDetailView(generics.RetrieveUpdateDestroyAPIView):
    """
    API-представление для работы с конкретным документом сотрудника

    GET: Получить детали документа
    PUT/PATCH: Обновить документ
    DELETE: Удалить документ
    """
    serializer_class = EmployeeDocumentSerializer
    permission_classes = [IsAuthenticated]

    def get_queryset(self):
        """
        Фильтрует документы по правам доступа
        """
        user = self.request.user

        # Админы и HR видят все документы
        if user.role in ['admin', 'hr_manager']:
            return EmployeeDocument.objects.all()
        # Сотрудники видят только свои документы
        elif hasattr(user, 'employee_profile'):
            return EmployeeDocument.objects.filter(
                employee=user.employee_profile
            )
        else:
                return EmployeeDocument.objects.none()
@api_view(['POST'])
@permission_classes([IsAuthenticated])
def generate_document_from_template(request, template_id, employee_id):
    """
Специальный эндпоинт для генерации документа на основе шаблона
    Создает персонализированный документ для конкретного сотрудника
    """
    # Проверяем права доступа
    if request.user.role not in ['admin', 'hr_manager']:
```

```
        return Response(
            {'error': 'Недостаточно прав для генерации документов'},
```



Продолжение листинга В.1

```
            status=status.HTTP_403_FORBIDDEN
        )

    # Получаем шаблон и сотрудника
    template = get_object_or_404(DocumentTemplate, id=template_id)
    employee = get_object_or_404(Employee, id=employee_id)

    # Генерируем персонализированный контент
    personalized_content = template.get_formatted_content(employee)

    # Возвращаем сгенерированный контент
    return Response({
        'template_name': template.name,
        'employee_name': employee.full_name,
        'generated_content': personalized_content,
        'employee_id': employee.id,
        'template_id': template.id
          })
@api_view(['GET'])
@permission_classes([IsAuthenticated])
def download_document(request, document_id):
    """
    Эндпоинт для скачивания файла документа
    Проверяет права доступа перед отдачей файла
    """
    user = request.user

    # Получаем документ
    try:
        if user.role in ['admin', 'hr_manager']:
            document = EmployeeDocument.objects.get(id=document_id)
        elif hasattr(user, 'employee_profile'):
            document = EmployeeDocument.objects.get(
                id=document_id,
                employee=user.employee_profile
            )
        else:
            raise Http404
    except EmployeeDocument.DoesNotExist:
        raise Http404

    # Проверяем наличие файла
    if not document.file:
        return Response(
            {'error': 'Файл не найден'},
            status=status.HTTP_404_NOT_FOUND
        )

    try:
        # Отдаем файл
        response = HttpResponse(
      document.file.read(),
            content_type='application/octet-stream'
        )
```

```
        response['Content-Disposition']            =            f'attachment;
filename="{document.file.name}"'
```



Продолжение листинга В.1

```
        return response
    except Exception as e:
        return Response(
            {'error': 'Ошибка при загрузке файла'},
            status=status.HTTP_500_INTERNAL_SERVER_ERROR
              )

documents/serializers.py
      Сериализаторы для API работы с документами

from rest_framework import serializers
from .models import DocumentTemplate, EmployeeDocument
      from employees.serializers import EmployeeSerializer
class DocumentTemplateSerializer(serializers.ModelSerializer):
    """
    Сериализатор для шаблонов документов
    Включает валидацию и дополнительные поля
    """

    # Дополнительные поля только для чтения
    content_length = serializers.SerializerMethodField()
    usage_count = serializers.SerializerMethodField()

    class Meta:
        model = DocumentTemplate
        fields = [
            'id', 'name', 'content', 'created_at',
            'content_length', 'usage_count'
        ]
        read_only_fields = ['created_at']

    def get_content_length(self, obj):
        """Возвращает длину контента шаблона в символах"""
        return len(obj.content) if obj.content else 0

    def get_usage_count(self, obj):
        """Возвращает количество документов, созданных на основе этого
шаблона"""
        return obj.employeedocument_set.count()

    def validate_name(self, value):
        """
        Валидация названия шаблона
        """
        if len(value.strip()) < 3:
            raise serializers.ValidationError(
                "Название шаблона должно содержать минимум 3 символа"
            )
        return value.strip()

    def validate_content(self, value):
        """
        Валидация содержимого шаблона
  """
```

```
        if len(value.strip()) < 10:
            raise serializers.ValidationError(
```



Продолжение листинга В.1

```
                "Содержимое шаблона должно содержать минимум 10 символов"
            )
              return value.strip()
class EmployeeDocumentSerializer(serializers.ModelSerializer):
    """
    Сериализатор для документов сотрудников
    Включает информацию о связанном сотруднике и шаблоне
    """

    # Вложенные сериализаторы для отображения связанных данных
    employee_info = EmployeeSerializer(source='employee', read_only=True)
    template_info       =       DocumentTemplateSerializer(source='template',
read_only=True)

    # Дополнительные поля
    file_extension = serializers.CharField(
        source='get_file_extension',
        read_only=True
    )
    file_size_mb = serializers.FloatField(
        source='get_file_size_mb',
        read_only=True
    )
    file_url = serializers.SerializerMethodField()

    class Meta:
        model = EmployeeDocument
        fields = [
            'id', 'employee', 'employee_info', 'template',
            'template_info', 'file', 'file_url', 'file_extension',
            'file_size_mb', 'signed_at'
        ]
        read_only_fields = ['signed_at']

    def get_file_url(self, obj):
        """
        Возвращает URL для скачивания файла документа
        """
        if obj.file:
            request = self.context.get('request')
            if request:
                return
request.build_absolute_uri(f'/api/documents/employee/{obj.id}/download/')
        return None

    def validate_file(self, value):
        """
        Валидация загружаемого файла документа
        """
        if value:
            # Проверяем размер файла (максимум 10MB)
            if value.size > 10 * 1024 * 1024:
                raise serializers.ValidationError(
                    "Размер файла не должен превышать 10MB"
                )
```

```
            # Проверяем расширение файла
```



Продолжение листинга В.1

```
            allowed_extensions = ['.pdf', '.doc', '.docx', '.txt', '.rtf']
            file_extension = value.name.lower().split('.')[-1]
            if f'.{file_extension}' not in allowed_extensions:
                raise serializers.ValidationError(
                    "Допустимые форматы: PDF, DOC, DOCX, TXT, RTF"
                )

              return value
class DocumentGenerationSerializer(serializers.Serializer):
    """
    Сериализатор для генерации документов на основе шаблонов
    Используется в специальных эндпоинтах
    """
    template_id = serializers.IntegerField()
    employee_id = serializers.IntegerField()
    save_as_file = serializers.BooleanField(default=False)

    def validate_template_id(self, value):
        """Проверяем существование шаблона"""
        try:
            DocumentTemplate.objects.get(id=value)
        except DocumentTemplate.DoesNotExist:
            raise serializers.ValidationError("Шаблон не найден")
        return value

    def validate_employee_id(self, value):
        """Проверяем существование сотрудника"""
        from employees.models import Employee
        try:
            Employee.objects.get(id=value)
        except Employee.DoesNotExist:
            raise serializers.ValidationError("Сотрудник не найден")
              return value

documents/urls.py
      URL-маршруты для API документов

from django.urls import path
from .views import (
    DocumentTemplateListCreateView, DocumentTemplateDetailView,
    EmployeeDocumentListCreateView, EmployeeDocumentDetailView,
    generate_document_from_template, download_document
      )
# API-маршруты для работы с документами и шаблонами
urlpatterns = [
    # Работа с шаблонами документов
    # GET /api/documents/templates/ - список шаблонов
    # POST /api/documents/templates/ - создание нового шаблона
    path('templates/',               DocumentTemplateListCreateView.as_view(),
name='document-templates'),

    # Работа с конкретным шаблоном
    # GET /api/documents/templates/{id}/ - детали шаблона
    # PUT/PATCH /api/documents/templates/{id}/ - обновление шаблона
```

```
    # DELETE /api/documents/templates/{id}/ - удаление шаблона
```



Продолжение листинга В.1

```
path('templates/<int:pk>/',              DocumentTemplateDetailView.as_view(),
name='document-template-detail'),

    # Работа с документами сотрудников
    # GET /api/documents/employee/ - список документов
    # POST /api/documents/employee/ - создание нового документа
    path('employee/',                EmployeeDocumentListCreateView.as_view(),
name='employee-documents'),

    # Работа с конкретным документом сотрудника
    # GET /api/documents/employee/{id}/ - детали документа
    # PUT/PATCH /api/documents/employee/{id}/ - обновление документа
    # DELETE /api/documents/employee/{id}/ - удаление документа
    path('employee/<int:pk>/',           EmployeeDocumentDetailView.as_view(),
name='employee-document-detail'),

    # Специальные эндпоинты
    #  POST  /api/documents/generate/{template_id}/{employee_id}/  -  генерация
документа из шаблона
    path('generate/<int:template_id>/<int:employee_id>/',
         generate_document_from_template,
         name='generate-document'),

    # GET /api/documents/employee/{id}/download/ - скачивание файла документа
    path('employee/<int:document_id>/download/',
         download_document,
         name='download-document'),
      ]

documents/forms.py
      Django-формы для HTML-интерфейса управления документами

from django import forms
from .models import DocumentTemplate, EmployeeDocument
      from employees.models import Employee
class DocumentTemplateForm(forms.ModelForm):
    """
    Форма для создания и редактирования шаблонов документов
    Используется в HTML-интерфейсе администрирования
    """
    class Meta:
        model = DocumentTemplate
        fields = ['name', 'content']

        # Человекочитаемые названия полей
        labels = {
            'name': 'Название шаблона',
            'content': 'Содержание шаблона',
        }

        # Кастомные виджеты для улучшения UX
        widgets = {
            'name': forms.TextInput(attrs={
                'class': 'form-control',
                'placeholder': 'Например: Трудовой договор'
```

```
            }),
            'content': forms.Textarea(attrs={
```



Продолжение листинга В.1

```
                'rows': 10,
      'class': 'form-control',
                'placeholder': 'Введите текст шаблона. Используйте {full_name},
{position}, {department} для подстановки данных сотрудника'
            }),
        }

    def clean_name(self):
        """
        Валидация названия шаблона
        """
        name = self.cleaned_data.get('name')
        if name and len(name.strip()) < 3:
            raise forms.ValidationError(
                "Название шаблона должно содержать минимум 3 символа"
            )
        return name.strip() if name else name

    def clean_content(self):
        """
        Валидация содержимого шаблона
        """
        content = self.cleaned_data.get('content')
        if content and len(content.strip()) < 10:
            raise forms.ValidationError(
                "Содержимое шаблона должно содержать минимум 10 символов"
            )
              return content.strip() if content else content
class EmployeeDocumentForm(forms.ModelForm):
    """
    Форма для загрузки документов сотрудников
    Позволяет связать документ с сотрудником и опционально с шаблоном
    """
    class Meta:
        model = EmployeeDocument
        fields = ['employee', 'template', 'file']

        # Человекочитаемые названия полей
        labels = {
            'employee': 'Сотрудник',
            'template': 'Шаблон документа',
            'file': 'Файл документа'
        }

        # Кастомные виджеты
        widgets = {
            'employee': forms.Select(attrs={
                'class': 'form-control'
            }),
            'template': forms.Select(attrs={
                'class': 'form-control'
            }),
            'file': forms.FileInput(attrs={
                'class': 'form-control',
                'accept': '.pdf,.doc,.docx,.txt,.rtf'
```

```
            })
        }
```



Продолжение листинга В.1

```
    def __init__(self, *args, kwargs):
"""
        Инициализация формы с дополнительными настройками
        """
        super().__init__(*args, kwargs)

        # Устанавливаем пустые выборы для выпадающих списков
        self.fields['employee'].empty_label = "Выберите сотрудника"
        self.fields['template'].empty_label      =      "Выберите      шаблон
(необязательно)"

        # Делаем поле шаблона необязательным
        self.fields['template'].required = False

    def clean_file(self):
        """
        Валидация загружаемого файла
        """
        file = self.cleaned_data.get('file')

        if file:
            # Проверяем размер файла (максимум 10MB)
            if file.size > 10 * 1024 * 1024:
                raise forms.ValidationError(
                    "Размер файла не должен превышать 10MB"
                )

            # Проверяем расширение файла
            allowed_extensions = ['.pdf', '.doc', '.docx', '.txt', '.rtf']
            file_extension = file.name.lower().split('.')[-1]
            if f'.{file_extension}' not in allowed_extensions:
                raise forms.ValidationError(
                    "Допустимые форматы файлов: PDF, DOC, DOCX, TXT, RTF"
                )

              return file
class DocumentGenerationForm(forms.Form):
    """
    Форма для генерации документов на основе шаблонов
    Позволяет выбрать шаблон и сотрудника для персонализации
    """
    template = forms.ModelChoiceField(
        queryset=DocumentTemplate.objects.all(),
        label="Шаблон документа",
        empty_label="Выберите шаблон",
        widget=forms.Select(attrs={'class': 'form-control'})
    )

    employee = forms.ModelChoiceField(
        queryset=Employee.objects.all(),
        label="Сотрудник",
        empty_label="Выберите сотрудника",
        widget=forms.Select(attrs={'class': 'form-control'})
```

```
    )

    def clean(self):
```



Продолжение листинга В.1

```
        """
        Валидация формы на уровне объекта
        """
        cleaned_data = super().clean()
    template = cleaned_data.get('template')
        employee = cleaned_data.get('employee')

        if template and employee:
            # Можно добавить дополнительные проверки
            # Например, проверить совместимость шаблона и сотрудника
            pass

              return cleaned_data

documents/apps.py
      Конфигурация Django-приложения

      from django.apps import AppConfig
class DocumentsConfig(AppConfig):
    """
    Конфигурация приложения documents
    Определяет настройки автополей и имя приложения
    """
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'documents'
          verbose_name = 'Управление документами'

documents/admin.py
      Регистрация моделей в административной панели

from django.contrib import admin
      from .models import DocumentTemplate, EmployeeDocument
@admin.register(DocumentTemplate)
class DocumentTemplateAdmin(admin.ModelAdmin):
    """
    Конфигурация административного интерфейса для шаблонов документов
    """
    list_display = ['name', 'created_at', 'get_usage_count']
    list_filter = ['created_at']
    search_fields = ['name', 'content']
    readonly_fields = ['created_at']

    def get_usage_count(self, obj):
        """Отображает количество использований шаблона"""
        return obj.employeedocument_set.count()
          get_usage_count.short_description = 'Использований'
@admin.register(EmployeeDocument)
class EmployeeDocumentAdmin(admin.ModelAdmin):
    """
    Конфигурация административного интерфейса для документов сотрудников
    """
    list_display  =  ['get_employee_name',  'get_template_name',  'signed_at',
'file']
    list_filter = ['signed_at', 'template']
```

```
    search_fields = ['employee__full_name', 'template__name']
    readonly_fields = ['signed_at']
```



Продолжение листинга В.1

```
    def get_employee_name(self, obj):
        """Отображает имя сотрудника"""
        return obj.employee.full_name
    get_employee_name.short_description = 'Сотрудник'

  def get_template_name(self, obj):
        """Отображает название шаблона"""
        return obj.template.name if obj.template else '-'
    get_template_name.short_description = 'Шаблон'

      # Модуль analytics - HR-аналитика с машинным обучением
      *Модуль для HR-аналитики с использованием машинного обучения для предсказаний*
analytics/models.py
      Модели для аналитических данных

      from django.db import models
# В данном модуле модели не требуются, так как аналитика работает
# с данными из других модулей (employees, vacancies, documents)
      # и использует внешние ML-модели для предсказаний
# Если потребуется хранить результаты аналитики, можно добавить модели:
# class AnalyticsReport(models.Model):
#     """Модель для хранения сгенерированных аналитических отчетов"""
#     pass
#
# class PredictionResult(models.Model):
#     """Модель для хранения результатов ML-предсказаний"""
      #     pass

analytics/views.py
      API-представления для аналитики и ML-предсказаний

from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework import status
from rest_framework.permissions import IsAuthenticated
from django.views.generic import TemplateView
from django.contrib.auth.mixins import LoginRequiredMixin
import joblib
import numpy as np
from .ml_models import predict_candidate_ranking, predict_turnover_risk
from employees.models import Employee
from vacancies.models import Vacancy, VacancyResponse
      from documents.models import DocumentTemplate, EmployeeDocument
class CandidateRankingView(APIView):
    """
    API-представление для ранжирования кандидатов с помощью ML
    Использует обученную модель для оценки подходящности кандидата
    """
    permission_classes = [IsAuthenticated]

    def post(self, request):
        """
        Обрабатывает POST-запрос с характеристиками кандидата
```

```
        Возвращает оценку ранжирования от 0 до 1
        """
        # Получаем признаки кандидата из запроса
```



Продолжение листинга В.1

```
        features = request.data.get('features')

        if not features:
            return Response(
                {'error': 'Features not provided'},
                status=status.HTTP_400_BAD_REQUEST
    )

        # Проверяем права доступа (только HR и админы)
        if request.user.role not in ['admin', 'hr_manager']:
            return Response(
                {'error': 'Недостаточно прав для использования ML-функций'},
                status=status.HTTP_403_FORBIDDEN
            )

        try:
            # Загружаем обученную модель из файла
            model = joblib.load('candidate_ranking_model.pkl')
        except FileNotFoundError:
            return Response(
                {'error': 'Model not found. Модель не обучена.'},
                status=status.HTTP_500_INTERNAL_SERVER_ERROR
            )
        except Exception as e:
            return Response(
                {'error': f'Error loading model: {str(e)}'},
                status=status.HTTP_500_INTERNAL_SERVER_ERROR
            )

        try:
            # Делаем предсказание с помощью ML-модели
            ranking = predict_candidate_ranking(model, features)

            # Возвращаем результат ранжирования
            return Response({
                'ranking_score': float(ranking),
                'recommendation': self._get_ranking_recommendation(ranking),
                'features_used': features
            })

        except Exception as e:
            return Response(
                {'error': f'Prediction error: {str(e)}'},
                status=status.HTTP_500_INTERNAL_SERVER_ERROR
            )

    def _get_ranking_recommendation(self, score):
        """
        Возвращает текстовую рекомендацию на основе оценки
        """
        if score >= 0.8:
            return "Отличный кандидат - рекомендуется к собеседованию"
        elif score >= 0.6:
```

```
            return "Хороший кандидат - стоит рассмотреть"
        elif score >= 0.4:
            return "Средний кандидат - требует дополнительной оценки"
```



Продолжение листинга В.1

```
        else:
                return "Кандидат не соответствует требованиям"
class TurnoverPredictionView(APIView):
    """
    API-представление для предсказания риска увольнения сотрудников
    Использует ML-модель для анализа вероятности ухода сотрудника
    """
    permission_classes = [IsAuthenticated]
    def post(self, request):
        """
        Обрабатывает POST-запрос с характеристиками сотрудника
        Возвращает риск увольнения от 0 до 1
        """
        # Получаем признаки сотрудника из запроса
        features = request.data.get('features')
        employee_id = request.data.get('employee_id')

        if not features:
            return Response(
                {'error': 'Features not provided'},
                status=status.HTTP_400_BAD_REQUEST
            )

        # Проверяем права доступа (только HR и админы)
        if request.user.role not in ['admin', 'hr_manager']:
            return Response(
                {'error': 'Недостаточно прав для использования ML-функций'},
                status=status.HTTP_403_FORBIDDEN
            )

        try:
            # Загружаем обученную модель для предсказания увольнений
            model = joblib.load('turnover_model.pkl')
        except FileNotFoundError:
            return Response(
                {'error': 'Turnover model not found. Модель не обучена.'},
                status=status.HTTP_500_INTERNAL_SERVER_ERROR
            )
        except Exception as e:
            return Response(
                {'error': f'Error loading model: {str(e)}'},
                status=status.HTTP_500_INTERNAL_SERVER_ERROR
            )

        try:
            # Делаем предсказание риска увольнения
            risk = predict_turnover_risk(model, features)

            # Получаем дополнительную информацию о сотруднике, если указан ID
            employee_info = None
            if employee_id:
                try:
                    employee = Employee.objects.get(id=employee_id)
                    employee_info = {
```

```
                        'name': employee.full_name,
                        'position': employee.position,
                        'department': employee.department
```



Продолжение листинга В.1

```
                    }
                except Employee.DoesNotExist:
                    pass

            # Возвращаем результат предсказания
            return Response({
                'turnover_risk': float(risk),
                'risk_level': self._get_risk_level(risk),
    'recommendations': self._get_risk_recommendations(risk),
                'employee_info': employee_info,
                'features_used': features
            })

        except Exception as e:
            return Response(
                {'error': f'Prediction error: {str(e)}'},
                status=status.HTTP_500_INTERNAL_SERVER_ERROR
            )

    def _get_risk_level(self, risk):
        """
        Возвращает уровень риска в текстовом виде
        """
        if risk >= 0.7:
            return "Высокий риск"
        elif risk >= 0.4:
            return "Средний риск"
        else:
            return "Низкий риск"

    def _get_risk_recommendations(self, risk):
        """
        Возвращает рекомендации на основе уровня риска
        """
        if risk >= 0.7:
            return [
                "Провести индивидуальную беседу с сотрудником",
                "Рассмотреть возможность повышения или изменения условий",
                "Проанализировать причины неудовлетворенности"
            ]
        elif risk >= 0.4:
            return [
                "Обратить внимание на мотивацию сотрудника",
                "Провести оценку удовлетворенности работой",
                "Рассмотреть возможности карьерного роста"
            ]
        else:
            return [
                "Сотрудник находится в стабильном состоянии",
                "Поддерживать текущий уровень мотивации"
                  ]
class AnalyticsPageView(LoginRequiredMixin, TemplateView):
    """
    HTML-страница дашборда аналитики
```

```
    Отображает основные метрики и интерфейс для ML-функций
    """
    template_name = "analytics.html"
```



Продолжение листинга В.1

```
    login_url = '/accounts/login/'

    def get_context_data(self, kwargs):
        """
        Формирует контекст с аналитическими данными для дашборда
        """
        context = super().get_context_data(kwargs)

        # Проверяем права доступа к аналитике
  if self.request.user.role not in ['admin', 'hr_manager']:
            context['access_denied'] = True
            return context

        # Основная статистика
        context.update({
            # Общие метрики
            'total_employees': Employee.objects.count(),
            'total_vacancies': Vacancy.objects.count(),
            'active_vacancies':
Vacancy.objects.filter(is_archived=False).count(),
            'total_responses': VacancyResponse.objects.count(),
            'total_documents': EmployeeDocument.objects.count(),

            # Статистика по откликам
            'new_responses':
VacancyResponse.objects.filter(status='new').count(),
            'interview_responses':
VacancyResponse.objects.filter(status='interview').count(),
            'accepted_responses':
VacancyResponse.objects.filter(status='accepted').count(),
            'rejected_responses':
VacancyResponse.objects.filter(status='rejected').count(),

            # Департаменты и их размеры
            'departments': self._get_department_stats(),

            # Позиции и их популярность
            'positions': self._get_position_stats(),

            # Статистика по шаблонам документов
            'template_usage': self._get_template_usage_stats(),

            # Метрики для ML-моделей
            'ml_features_info': self._get_ml_features_info(),
        })

        return context

    def _get_department_stats(self):
        """
        Возвращает статистику по отделам
        """
        from django.db.models import Count
        return list(Employee.objects.values('department')
```

```
                .annotate(count=Count('id'))
                .order_by('-count')[:10])
```



Продолжение листинга В.1

```
    def _get_position_stats(self):
        """
        Возвращает статистику по должностям
        """
        from django.db.models import Count
        return list(Employee.objects.values('position')
                    .annotate(count=Count('id'))
                    .order_by('-count')[:10])

    def _get_template_usage_stats(self):
  """
        Возвращает статистику использования шаблонов документов
        """
        from django.db.models import Count
        return list(DocumentTemplate.objects.annotate(
            usage_count=Count('employeedocument')
        ).order_by('-usage_count')[:5])

    def _get_ml_features_info(self):
        """
        Возвращает информацию о признаках для ML-моделей
        """
        return {
            'candidate_features': [
                'education_level',  # Уровень образования (1-5)
                'experience_years', # Опыт работы в годах
                'skills_match',     # Соответствие навыков (0-1)
                'previous_positions', # Количество предыдущих позиций
                'age',              # Возраст кандидата
            ],
            'turnover_features': [
                'job_satisfaction', # Удовлетворенность работой (1-5)
                'salary_level',     # Уровень зарплаты (1-5)
                'years_in_company', # Лет в компании
                'promotion_frequency', # Частота повышений
                'work_life_balance', # Баланс работы и жизни (1-5)
            ]
              }
class AnalyticsReportView(APIView):
    """
    API-представление для генерации аналитических отчетов
    """
    permission_classes = [IsAuthenticated]

    def get(self, request):
        """
        Генерирует общий аналитический отчет
        """
        if request.user.role not in ['admin', 'hr_manager']:
            return Response(
                {'error': 'Недостаточно прав'},
                status=status.HTTP_403_FORBIDDEN
            )
```

```
        # Собираем данные для отчета
        report_data = {
            'generated_at': timezone.now().isoformat(),
```



Продолжение листинга В.1

```
            'generated_by': request.user.username,

            # Основные метрики
            'metrics': {
                'employees': {
                    'total': Employee.objects.count(),
                    'by_department':
list(Employee.objects.values('department')
                                        .annotate(count=Count('id'))),
                    'by_position': list(Employee.objects.values('position')
                                       .annotate(count=Count('id'))),
                },
      'vacancies': {
                    'total': Vacancy.objects.count(),
                    'active':
Vacancy.objects.filter(is_archived=False).count(),
                    'archived':
Vacancy.objects.filter(is_archived=True).count(),
                },
                'responses': {
                    'total': VacancyResponse.objects.count(),
                    'by_status': list(VacancyResponse.objects.values('status')
                                      .annotate(count=Count('id'))),
                },
            },

            # Тренды (можно расширить)
            'trends': {
                'hiring_trend': 'stable',  # Можно рассчитать на основе данных
                'turnover_trend': 'low',   # Можно рассчитать на основе данных
            }
        }

              return Response(report_data)

analytics/urls.py
      URL-маршруты для API аналитики

from django.urls import path
      from   .views   import   CandidateRankingView,   TurnoverPredictionView,
AnalyticsReportView
# API-маршруты для аналитики и ML-функций
urlpatterns = [
    # ML-предсказание ранжирования кандидатов
    # POST /api/analytics/candidate-ranking/
    #  Body:  {"features":  [education_level,  experience_years,  skills_match,
previous_positions, age]}
    path('candidate-ranking/',                 CandidateRankingView.as_view(),
name='candidate-ranking'),

    # ML-предсказание риска увольнения
    # POST /api/analytics/turnover/
```

```
    # Body: {"features": [job_satisfaction, salary_level, years_in_company,
promotion_frequency, work_life_balance], "employee_id": 123}
    path('turnover/',    TurnoverPredictionView.as_view(),    name='turnover-
prediction'),

    # Генерация аналитического отчета
```

Продолжение приложения В

Продолжение листинга В.1

```
    # GET /api/analytics/report/
    path('report/', AnalyticsReportView.as_view(), name='analytics-report'),
      ]

analytics/ml_models.py
      Модуль для работы с ML-моделями

import numpy as np
import joblib
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
      import warnings
# Отключаем предупреждения для чистоты вывода
      warnings.filterwarnings('ignore')
def                    train_candidate_ranking_model(X,                    y,
model_path='candidate_ranking_model.pkl'):
    """
    Обучает модель для ранжирования кандидатов

    Args:
        X: Матрица признаков кандидатов (n_samples, n_features)
        y: Целевая переменная - успешность кандидата (0/1)
        model_path: Путь для сохранения обученной модели

    Returns:
        Обученная модель

    Пример использования:
        features = [[4, 5, 0.8, 3, 28], [2, 2, 0.4, 1, 22], ...]  # [образование,
опыт, соответствие навыков, кол-во позиций, возраст]
        target = [1, 0, 1, 1, 0, ...]  # 1 - успешный кандидат, 0 - неуспешный
        model = train_candidate_ranking_model(features, target)
    """

    # Преобразуем в numpy массивы
    X = np.array(X)
    y = np.array(y)

    # Создаем и обучаем модель логистической регрессии
    model = LogisticRegression(
        random_state=42,
        max_iter=1000,
        solver='liblinear'  # Хороший солвер для небольших датасетов
    )

    # Нормализация признаков для лучшей работы модели
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)
```

```
    # Обучаем модель
    model.fit(X_scaled, y)

    # Сохраняем модель и скейлер
    model_data = {
        'model': model,
```



Продолжение листинга В.1

```
        'scaler': scaler,
        'feature_names':        ['education_level',        'experience_years',
'skills_match', 'previous_positions', 'age']
    }

    joblib.dump(model_data, model_path)
    print(f"Модель ранжирования кандидатов сохранена в {model_path}")

          return model
def predict_candidate_ranking(model_data, candidate_features):
    """
    Предсказывает ранжирование кандидата

    Args:
  model_data: Загруженные данные модели (модель + скейлер)
        candidate_features: Список признаков кандидата [education, experience,
skills, positions, age]

    Returns:
        Вероятность успешности кандидата (от 0 до 1)

    Пример:
        features = [4, 5, 0.8, 3, 28]  # Образование=4, опыт=5лет, навыки=80%,
позиций=3, возраст=28
        score = predict_candidate_ranking(model_data, features)
    """

    # Если model_data это просто модель (для обратной совместимости)
    if hasattr(model_data, 'predict_proba'):
        model = model_data
        scaler = None
    else:
        model = model_data['model']
        scaler = model_data.get('scaler')

    # Преобразуем признаки в numpy массив
    features_array = np.array([candidate_features])

    # Применяем нормализацию если есть скейлер
    if scaler:
        features_array = scaler.transform(features_array)

    # Получаем вероятность положительного класса (успешный кандидат)
    probability = model.predict_proba(features_array)[0][1]

          return probability
def train_turnover_model(X, y, model_path='turnover_model.pkl'):
    """
    Обучает модель для предсказания риска увольнения сотрудников

    Args:
```

```
        X: Матрица признаков сотрудников (n_samples, n_features)
        y: Целевая переменная - факт увольнения (0/1)
        model_path: Путь для сохранения обученной модели

    Returns:
        Обученная модель
```



Продолжение листинга В.1

```
    Пример использования:
        features = [[3, 4, 2, 0.5, 3], [5, 5, 5, 2, 4], ...]  # 
[удовлетворенность, зарплата, годы, повышения, баланс]
        target = [1, 0, 1, 0, ...]  # 1 - уволился, 0 - остался
        model = train_turnover_model(features, target)
    """

    # Преобразуем в numpy массивы
    X = np.array(X)
    y = np.array(y)

    # Используем Random Forest для лучшего качества предсказаний
    model = RandomForestClassifier(
        n_estimators=100,
      random_state=42,
        max_depth=10,
        min_samples_split=5
    )

    # Нормализация признаков
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    # Обучаем модель
    model.fit(X_scaled, y)

    # Сохраняем модель и скейлер
    model_data = {
        'model': model,
        'scaler': scaler,
        'feature_names':          ['job_satisfaction',          'salary_level', 
'years_in_company', 'promotion_frequency', 'work_life_balance']
    }

    joblib.dump(model_data, model_path)
    print(f"Модель предсказания увольнений сохранена в {model_path}")

          return model
def predict_turnover_risk(model_data, employee_features):
    """
    Предсказывает риск увольнения сотрудника

    Args:
        model_data: Загруженные данные модели (модель + скейлер)
        employee_features: Список признаков сотрудника [satisfaction, salary, 
years, promotions, balance]

    Returns:
        Вероятность увольнения (от 0 до 1)
```

```
    Пример:
        features = [3, 4, 2, 0.5, 3]  # Удовлетворенность=3, зарплата=4, годы=2,
повышения=0.5, баланс=3
        risk = predict_turnover_risk(model_data, features)
    """
```

Продолжение приложения В

Продолжение листинга В.1

```
    # Если model_data это просто модель (для обратной совместимости)
    if hasattr(model_data, 'predict_proba'):
        model = model_data
        scaler = None
    else:
        model = model_data['model']
        scaler = model_data.get('scaler')

    # Преобразуем признаки в numpy массив
    features_array = np.array([employee_features])

    # Применяем нормализацию если есть скейлер
    if scaler:
        features_array = scaler.transform(features_array)

    # Получаем вероятность увольнения (положительный класс)
  risk = model.predict_proba(features_array)[0][1]

          return risk
# Функции для генерации тестовых данных (для демонстрации)
def generate_sample_candidate_data(n_samples=1000):
    """
    Генерирует случайные данные кандидатов для обучения модели
    Используется только для демонстрации и тестирования
    """
    np.random.seed(42)

    # Генерируем признаки
    education = np.random.randint(1, 6, n_samples)  # Уровень образования 1-5
    experience = np.random.randint(0, 15, n_samples)  # Опыт 0-15 лет
    skills_match = np.random.uniform(0, 1, n_samples)  # Соответствие навыков
0-1
    positions = np.random.randint(0, 8, n_samples)  # Количество позиций 0-8
    age = np.random.randint(20, 60, n_samples)  # Возраст 20-60

    # Создаем целевую переменную на основе логики
    # Более образованные и опытные кандидаты с хорошим соответствием навыков
более успешны
    target = (
        (education >= 3) &
        (experience >= 2) &
        (skills_match >= 0.6) &
        (age <= 45)
    ).astype(int)

    # Добавляем немного шума
    noise = np.random.choice([0, 1], n_samples, p=[0.8, 0.2])
    target = np.logical_xor(target, noise).astype(int)

    X = np.column_stack([education, experience, skills_match, positions, age])
```

```
        return X, target
def generate_sample_turnover_data(n_samples=1000):
    """
    Генерирует случайные данные сотрудников для обучения модели увольнений
    Используется только для демонстрации и тестирования
    """
```



Продолжение листинга В.1

```
    np.random.seed(42)

    # Генерируем признаки
    satisfaction = np.random.randint(1, 6, n_samples)  # Удовлетворенность 1-5
    salary = np.random.randint(1, 6, n_samples)  # Уровень зарплаты 1-5
    years = np.random.randint(0, 20, n_samples)  # Лет в компании 0-20
    promotions = np.random.uniform(0, 3, n_samples)  # Частота повышений
    balance = np.random.randint(1, 6, n_samples)  # Баланс работы/жизни 1-5

    # Создаем целевую переменную
    # Неудовлетворенные сотрудники с низкой зарплатой и плохим балансом склонны
увольняться
    target = (
        (satisfaction <= 2) |
        (salary <= 2) |
        (balance <= 2) |
        ((years >= 10) & (promotions < 0.5))  # Долго работают без повышений
).astype(int)

    # Добавляем шум
    noise = np.random.choice([0, 1], n_samples, p=[0.7, 0.3])
    target = np.logical_xor(target, noise).astype(int)

    X = np.column_stack([satisfaction, salary, years, promotions, balance])

        return X, target
# Пример скрипта для первоначального обучения моделей
if __name__ == "__main__":
    print("Генерация тестовых данных и обучение моделей...")

    # Обучение модели ранжирования кандидатов
    print("\n1. Обучение модели ранжирования кандидатов...")
    X_candidates, y_candidates = generate_sample_candidate_data(1000)
    candidate_model = train_candidate_ranking_model(X_candidates,
y_candidates)

    # Обучение модели предсказания увольнений
    print("\n2. Обучение модели предсказания увольнений...")
    X_turnover, y_turnover = generate_sample_turnover_data(1000)
    turnover_model = train_turnover_model(X_turnover, y_turnover)

    print("\nМодели успешно обучены и сохранены!")
        print("Теперь можно использовать API для предсказаний.")

analytics/templatetags/analytics_extras.py
    Дополнительные теги шаблонов для аналитики

from django import template
    import json
# Регистрируем библиотеку тегов шаблонов
    register = template.Library()
```

```
@register.filter
def index(value, arg):
    """
    Получает элемент по индексу из списка или словаря

    Использование в шаблоне:
    {{ my_list|index:0 }}  - получить первый элемент списка
```



Продолжение листинга В.1

```
    {{ my_dict|index:"key" }}  - получить значение по ключу
    """
    try:
        return value[arg]
    except (IndexError, TypeError, KeyError):
            return None
@register.filter
def multiply(value, arg):
    """
    Умножает значение на аргумент
    Полезно для вычислений в шаблонах

    Использование:
    {{ value|multiply:100 }}  - умножить на 100 (например, для процентов)
    {{ price|multiply:1.2 }}  - умножить на 1.2 (например, добавить НДС)
    """
    try:
        return float(value) * float(arg)
  except (ValueError, TypeError):
            return 0
@register.filter
def percentage(value, total):
    """
    Вычисляет процентное соотношение

    Использование:
    {{ part|percentage:total }}  - получить процент части от целого
    """
    try:
        if float(total) == 0:
            return 0
        return round((float(value) / float(total)) * 100, 1)
    except (ValueError, TypeError, ZeroDivisionError):
            return 0
@register.filter
def json_encode(value):
    """
    Кодирует значение в JSON для использования в JavaScript

    Использование:
    <script>
        var data = {{ my_data|json_encode }};
    </script>
    """
          return json.dumps(value)
@register.simple_tag
def risk_badge_class(risk_level):
    """
    Возвращает CSS-класс для отображения уровня риска
```

```
    Использование:
    {% load analytics_extras %}
    <span class="badge {% risk_badge_class 'high' %}">Высокий риск</span>
    """
    risk_classes = {
        'low': 'badge-success',
        'medium': 'badge-warning',
```



Продолжение листинга В.1

```
        'high': 'badge-danger',
        'Низкий риск': 'badge-success',
        'Средний риск': 'badge-warning',
        'Высокий риск': 'badge-danger',
    }
          return risk_classes.get(risk_level, 'badge-secondary')
@register.simple_tag
def format_ml_score(score):
    """
    Форматирует ML-оценку для отображения

    Использование:
    {% format_ml_score 0.847 %}  -> "84.7%"
    """
    try:
        return f"{float(score) * 100:.1f}%"
    except (ValueError, TypeError):
            return "N/A"
@register.inclusion_tag('analytics/chart_data.html')

def render_chart_data(chart_type, data, title=""):
    """
    Тег включения для рендеринга данных графика

    Использование:
    {% render_chart_data "pie" department_stats "Сотрудники по отделам" %}
    """
    return {
        'chart_type': chart_type,
        'data': data,
        'title': title,
        'chart_id': f"chart_{hash(str(data))}"
          }
@register.filter
def dict_get(dictionary, key):
    """
    Получает значение из словаря по ключу
    Безопасная альтернатива обращению по ключу

    Использование:
    {{ my_dict|dict_get:"key_name" }}
    """
    if isinstance(dictionary, dict):
        return dictionary.get(key)
          return None
@register.simple_tag
def analytics_status_icon(status):
    """
    Возвращает иконку для статуса в аналитике
```

```
    Использование:
    {% analytics_status_icon "success" %}  -> <i class="fas fa-check-circle
text-success"></i>
    """
    icons = {
        'success': '<i class="fas fa-check-circle text-success"></i>',
        'warning': '<i class="fas fa-exclamation-triangle text-warning"></i>',
```

Продолжение приложения В

Окончание листинга В.1

```
        'danger': '<i class="fas fa-times-circle text-danger"></i>',
        'info': '<i class="fas fa-info-circle text-info"></i>',
        'primary': '<i class="fas fa-star text-primary"></i>',
    }
    return icons.get(status, '<i class="fas fa-circle text-muted"></i>')
```

Листинг В.2 – Программный код пользовательского интерфейса

```
base.html
Базовый шаблон с навигацией и общей структурой

<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <!-- Динамический заголовок страницы -->
  <title>{% block title %}HR Platform{% endblock %}</title>

  <!-- Подключение основных стилей -->
  <link rel="stylesheet" href="/static/css/styles.css">

  <!-- Блок для дополнительных стилей на конкретных страницах -->
  {% block extra_head %}{% endblock %}
</head>
<body>
  <!-- Верхняя навигационная панель -->
  <header>
    <div class="container header-container">
      <!-- Логотип с ссылкой на главную -->
      <h1 class="logo"><a href="{% url 'home' %}">HR Platform</a></h1>

      <!-- Навигационное меню -->
      <nav>
        <ul class="nav-list">
          {% if user.is_authenticated %}
            <!-- Меню для авторизованных пользователей -->
            <li><a href="{% url 'profile' %}">{{ user.username }}</a></li>
            <li><a href="{% url 'logout_api' %}">Выход</a></li>
          {% else %}
            <!-- Меню для неавторизованных пользователей -->
            <li><a href="{% url 'login' %}">Вход</a></li>
            <li><a href="{% url 'signup' %}">Регистрация</a></li>
          {% endif %}
        </ul>
      </nav>
    </div>
  </header>
```

```
  <!-- Основная область контента -->
  <div class="main-wrapper">
    {% if user.is_authenticated %}
      <!-- Боковая навигация для авторизованных пользователей -->
      <aside class="sidebar">
        <ul>
          <li><a href="{% url 'profile' %}">Профиль</a></li>
```



Продолжение листинга В.2

```
          <li><a href="{% url 'employees' %}">Сотрудники</a></li>
          <li><a href="{% url 'vacancies' %}">Вакансии</a></li>
          <li><a href="{% url 'document-templates' %}">Документы</a></li>
          <li><a href="{% url 'analytics_dashboard' %}">Аналитика</a></li>
        </ul>
</aside>
    {% endif %}

    <!-- Основной контент страницы -->
    <main class="container">
      <!-- Блок для контента конкретной страницы -->
      {% block content %}
      {% endblock %}
    </main>
  </div>

  <!-- Подвал сайта -->
  <footer>
    <div class="container">
      <p>&copy; {% now "Y" %} HR Platform. Все права защищены.</p>
    </div>
  </footer>

  <!-- Подключение основных скриптов -->
  <script src="/static/js/main.js"></script>

  <!-- Блок для дополнительных скриптов на конкретных страницах -->
  {% block extra_js %}{% endblock %}
</body>
</html>

home.html
Главная страница с приветствием и статистикой

{% extends 'base.html' %}
<!-- Заголовок страницы -->
{% block title %}Главная - HR Platform{% endblock %}
{% block content %}
<!-- Приветственная секция -->
<section class="hero">
  <h2>Добро пожаловать в HR Platform</h2>
  <p>Цифровая платформа для управления кадровыми ресурсами.</p>

  {% if not user.is_authenticated %}
  <!-- Призыв к действию для неавторизованных пользователей -->
  <div class="cta">
    <a href="{% url 'signup' %}" class="btn">Зарегистрироваться</a>
    <a href="{% url 'login' %}" class="btn secondary">Войти в систему</a>
  </div>
  {% endif %}
```

```
</section>
{% if user.is_authenticated %}
<!-- Дашборд с основной статистикой (только для авторизованных) -->
<section class="dashboard">
  <h3>Статистика системы</h3>

  <!-- Карточки с метриками -->
```



Продолжение листинга В.2

```
  <div class="metrics-grid">
    <div class="metric-card">
      <h4>Сотрудники</h4>
      <span class="metric-value">{{ employees_count|default:0 }}</span>
      <p>Всего в системе</p>
    </div>
    <div class="metric-card">
      <h4>Активные вакансии</h4>
      <span class="metric-value">{{ vacancies_count|default:0 }}</span>
      <p>Открыто для подачи заявок</p>
    </div>

    <div class="metric-card">
      <h4>Документы</h4>
      <span class="metric-value">{{ documents_count|default:0 }}</span>
      <p>Шаблоны и файлы сотрудников</p>
    </div>
  </div>

  <!-- Лента недавней активности -->
  {% if recent_activity %}
  <div class="activity-feed">
    <h4>Недавняя активность</h4>
    <ul class="activity-list">
      {% for activity in recent_activity %}
      <li class="activity-item">
        <div class="activity-content">
          <span class="activity-event">{{ activity.event }}</span>
          <span class="activity-status badge {{ activity.status_class }}">
            {{ activity.status }}
          </span>
        </div>
        <small class="activity-date">{{ activity.date|date:"d.m.Y H:i"
}}</small>
      </li>
      {% endfor %}
    </ul>
  </div>
  {% endif %}
</section>
{% endif %}
{% endblock %}

registration/login.html
Страница входа в систему

{% extends 'base.html' %}
<!-- Заголовок страницы -->
{% block title %}Вход - HR Platform{% endblock %}
{% block content %}
```

```
<div class="card form-card">
    <h2>Вход в систему</h2>

    <!-- Отображение сообщений об ошибках -->
    {% if messages %}
        {% for message in messages %}
            <div class="alert alert-{{ message.tags }}">{{ message }}</div>
```



Продолжение листинга В.2

```
        {% endfor %}
    {% endif %}

    <!-- Форма входа -->
    <form method="post" action="">
        <!-- CSRF-токен для защиты от атак -->
 {% csrf_token %}

        <!-- Поле имени пользователя -->
        <div class="form-group">
            <label for="username">Имя пользователя</label>
            <input
                type="text"
                id="username"
                name="username"
                placeholder="Введите имя пользователя"
                required
                class="form-control"
            >
        </div>

        <!-- Поле пароля -->
        <div class="form-group">
            <label for="password">Пароль</label>
            <input
                type="password"
                id="password"
                name="password"
                placeholder="Введите пароль"
                required
                class="form-control"
            >
        </div>

        <!-- Кнопка отправки -->
        <button type="submit" class="btn btn-primary">Войти</button>
    </form>

    <!-- Ссылка на регистрацию -->
    <p class="small-text">
        Нет аккаунта? <a href="{% url 'signup' %}">Зарегистрируйтесь
сейчас</a>
    </p>
</div>
{% endblock %}

registration/signup.html
Страница регистрации нового пользователя

{% extends 'base.html' %}
```

```
<!-- Заголовок страницы -->
{% block title %}Регистрация - HR Platform{% endblock %}
{% block content %}
<div class="card form-card">
    <h2>Регистрация</h2>

    <!-- Форма регистрации с автоматически сгенерированными полями -->
```



Продолжение листинга В.2

```
    <form method="post" action="">
        <!-- CSRF-токен для защиты -->
        {% csrf_token %}

        <!-- Отображение формы через Django -->
        {{ form.as_p }}

   <!-- Кнопка отправки -->
        <button type="submit" class="btn btn-
primary">Зарегистрироваться</button>
    </form>

    <!-- Ссылка на страницу входа -->
    <p class="small-text">
        Уже есть аккаунт? <a href="{% url 'login' %}">Войдите в систему</a>
    </p>
</div>
{% endblock %}

registration/profile.html
Страница профиля пользователя

{% extends "base.html" %}
<!-- Заголовок страницы -->
{% block title %}Профиль пользователя - HR Platform{% endblock %}
{% block content %}
<div class="card form-card">
    <h2>Профиль пользователя</h2>

    <!-- Информация о пользователе -->
    <div class="profile-info">
        <div class="info-group">
            <label>Имя пользователя:</label>
            <span class="info-value">{{ user.username }}</span>
        </div>

        <div class="info-group">
            <label>Email:</label>
            <span class="info-value">{{ user.email|default:"Не указан"
}}</span>
        </div>

        <div class="info-group">
            <label>Роль в системе:</label>
            <span class="info-value badge">{{ user.get_role_display }}</span>
        </div>

        <div class="info-group">
            <label>Дата регистрации:</label>
```

```
            <span class="info-value">{{ user.date_joined|date:"d.m.Y"
}}</span>
        </div>
    </div>

    <!-- Дополнительная информация для сотрудников -->
    {% if user.employee_profile %}
    <div class="employee-info">
```

Продолжение приложения В

Продолжение листинга В.2

```
        <h3>Информация о сотруднике</h3>

        <div class="info-group">
            <label>ФИО:</label>
            <span class="info-value">{{ user.employee_profile.full_name
}}</span>
        </div>

    <div class="info-group">
            <label>Должность:</label>
            <span class="info-value">{{ user.employee_profile.position
}}</span>
        </div>

        <div class="info-group">
            <label>Отдел:</label>
            <span class="info-value">{{ user.employee_profile.department
}}</span>
        </div>

        <div class="info-group">
            <label>Дата приема на работу:</label>
            <span class="info-value">{{
user.employee_profile.date_hired|date:"d.m.Y" }}</span>
        </div>
    </div>
    {% endif %}

    <!-- Кнопки действий -->
    <div class="profile-actions">
        <a href="{% url 'home' %}" class="btn btn-secondary">Вернуться на
главную</a>

        <!-- Дополнительные действия для HR и админов -->
        {% if user.role == "admin" or user.role == "hr_manager" %}
            <a href="{% url 'analytics_dashboard' %}" class="btn btn-
primary">Аналитика</a>
        {% endif %}
    </div>
</div>
{% endblock %}

employees/list.html
Страница списка сотрудников

{% extends 'base.html' %}
<!-- Заголовок страницы -->
{% block title %}Список сотрудников - HR Platform{% endblock %}
{% block content %}
```

```
<div class="page-header">
    <h2>Список сотрудников</h2>

    <!-- Кнопка добавления нового сотрудника (только для HR и админов) -->
    {% if user.is_authenticated %}
      {% if user.role == "admin" or user.role == "hr_manager" %}
        <a href="{% url 'employee_create' %}" class="btn btn-primary">
          Добавить сотрудника
```



Продолжение листинга В.2

```
        </a>
      {% endif %}
    {% endif %}
</div>
<!-- Таблица сотрудников -->
<div class="table-responsive">
    <table class="data-table">
      <thead>
        <tr>
  <th>ID</th>
          <th>ФИО</th>
          <th>Дата приёма</th>
          <th>Должность</th>
          <th>Квалификация</th>
          <th>Отдел</th>

          <!-- Колонка действий только для HR и админов -->
          {% if user.is_authenticated %}
            {% if user.role == "admin" or user.role == "hr_manager" %}
              <th>Действия</th>
            {% endif %}
          {% endif %}
        </tr>
      </thead>

      <tbody>
        {% for employee in employees %}
        <tr>
          <td>{{ employee.id }}</td>

          <!-- Ссылка на детальную страницу сотрудника -->
          <td>
            <a href="{% url 'employee_detail' employee.id %}">
              {{ employee.full_name }}
            </a>
          </td>

          <td>{{ employee.date_hired|date:"d.m.Y" }}</td>
          <td>{{ employee.position }}</td>
          <td>{{ employee.qualification|truncatechars:50 }}</td>
          <td>{{ employee.department }}</td>

          <!-- Действия с сотрудником (только для HR и админов) -->
          {% if user.is_authenticated %}
            {% if user.role == "admin" or user.role == "hr_manager" %}
              <td class="actions">
                <a href="{% url 'employee_update' employee.id %}" class="btn
btn-sm btn-outline">
                  Редактировать
```

```
                </a>
                <a href="{% url 'employee_delete' employee.id %}" class="btn
btn-sm btn-danger">
                  Удалить
                </a>
              </td>
            {% endif %}
          {% endif %}
```



Продолжение листинга В.2

```
        </tr>

        {% empty %}
        <!-- Сообщение при отсутствии сотрудников -->
        <tr>
          <td colspan="7" class="text-center">
            Сотрудники не найдены.
          </td>
        </tr>
        {% endfor %}
    </tbody>
    </table>
</div>
<!-- Пагинация (если необходимо) -->
{% if employees.has_other_pages %}
<nav class="pagination-nav">
    <ul class="pagination">
        {% if employees.has_previous %}
            <li><a href="?page={{ employees.previous_page_number }}">&laquo;
Предыдущая</a></li>
        {% endif %}

        <li class="current">
            Страница {{ employees.number }} из {{
employees.paginator.num_pages }}
        </li>

        {% if employees.has_next %}
            <li><a href="?page={{ employees.next_page_number }}">Следующая
&raquo;</a></li>
        {% endif %}
    </ul>
</nav>
{% endif %}
{% endblock %}

employees/detail.html
Детальная страница сотрудника

{% extends 'base.html' %}
<!-- Заголовок страницы -->
{% block title %}Профиль сотрудника - HR Platform{% endblock %}
{% block content %}
<div class="employee-detail">
    <!-- Заголовок с именем сотрудника -->
    <div class="page-header">
        <h2>Профиль сотрудника: {{ employee.full_name }}</h2>

        <!-- Кнопки действий (только для HR и админов) -->
```

```
        {% if user.is_authenticated %}
          {% if user.role == "admin" or user.role == "hr_manager" %}
            <div class="actions">
                <a href="{% url 'employee_update' employee.id %}" class="btn
btn-primary">
                    Редактировать
                </a>
```

Продолжение приложения В

Продолжение листинга В.2

```
                <a href="{% url 'employee_delete' employee.id %}" class="btn
btn-danger">
                    Удалить
                </a>
            </div>
          {% endif %}
        {% endif %}
    </div>

    <!-- Карточка с информацией о сотруднике -->
    <div class="card employee-info-card">
        <h3>Основная информация</h3>
        <div class="info-grid">
            <div class="info-item">
                <label>ФИО:</label>
                <span>{{ employee.full_name }}</span>
            </div>

            <div class="info-item">
                <label>Дата приёма на работу:</label>
                <span>{{ employee.date_hired|date:"d.m.Y" }}</span>
            </div>

            <div class="info-item">
                <label>Стаж работы:</label>
                <span>{{ employee.get_years_of_service }} лет</span>
            </div>

            <div class="info-item">
                <label>Должность:</label>
                <span>{{ employee.position }}</span>
            </div>

            <div class="info-item">
                <label>Отдел:</label>
                <span>{{ employee.department }}</span>
            </div>

            <div class="info-item full-width">
                <label>Квалификация:</label>
                <div class="qualification-text">
                    {{ employee.qualification|linebreaks }}
                </div>
            </div>

            {% if employee.history %}
            <div class="info-item full-width">
                <label>История изменений:</label>
                <div class="history-text">
```

```
                    {{ employee.history|linebreaks }}
                </div>
            </div>
            {% endif %}
        </div>
    </div>

    <!-- Связанная информация -->
```

Продолжение приложения В

Продолжение листинга В.2

```
    {% if user.role == "admin" or user.role == "hr_manager" %}
    <div class="related-info">
        <!-- Документы сотрудника -->
        {% if employee.documents.all %}
        <div class="card">
            <h3>Документы сотрудника</h3>
            <ul class="document-list">
                {% for document in employee.documents.all %}
                <li class="document-item">
                    <a href="{{ document.file.url }}" target="_blank">
                        {{ document.template.name|default:"Документ" }}
                    </a>
            <small>({{ document.signed_at|date:"d.m.Y" }})</small>
                </li>
                {% endfor %}
            </ul>
        </div>
        {% endif %}

        <!-- Связанный пользователь -->
        {% if employee.user %}
        <div class="card">
            <h3>Учетная запись</h3>
            <div class="user-info">
                <p><strong>Пользователь:</strong> {{ employee.user.username
}}</p>
                <p><strong>Email:</strong> {{ employee.user.email|default:"Не
указан" }}</p>
                <p><strong>Роль:</strong> {{ employee.user.get_role_display
}}</p>
                <p><strong>Дата регистрации:</strong> {{
employee.user.date_joined|date:"d.m.Y" }}</p>
            </div>
        </div>
        {% endif %}
    </div>
    {% endif %}

    <!-- Навигация -->
    <div class="navigation">
        <a href="{% url 'employees' %}" class="btn btn-secondary">
            &larr; Назад к списку сотрудников
        </a>
    </div>
</div>
{% endblock %}
<!-- Дополнительные стили для страницы -->
{% block extra_head %}
<style>
```

```
.employee-detail {
    max-width: 1000px;
    margin: 0 auto;
}
.info-grid {
    display: grid;
    grid-template-columns: 1fr 1fr;
    gap: 1rem;
```



Продолжение листинга В.2

```
    margin-top: 1rem;
}
.info-item {
    display: flex;
    flex-direction: column;
    gap: 0.25rem;
}
.info-item.full-width {
    grid-column: 1 / -1;
}
.info-item label {
    font-weight: bold;
    color: #666;
    font-size: 0.9rem;
}
.qualification-text,
.history-text {
    background: #f8f9fa;
    padding: 1rem;
    border-radius: 4px;
    border-left: 4px solid #007bff;
}
.document-list {
    list-style: none;
    padding: 0;
}
.document-item {
    padding: 0.5rem 0;
    border-bottom: 1px solid #eee;
}
.document-item:last-child {
    border-bottom: none;
}
.related-info {
    margin-top: 2rem;
    display: grid;
    grid-template-columns: 1fr 1fr;
    gap: 2rem;
}
@media (max-width: 768px) {
    .info-grid {
        grid-template-columns: 1fr;
    }

    .related-info {
        grid-template-columns: 1fr;
    }
}
</style>
```

```
{% endblock %}

employees/employee_form.html
Форма создания/редактирования сотрудника

{% extends 'base.html' %}
<!-- Динамический заголовок в зависимости от режима -->
{% block title %}
```



Продолжение листинга В.2

```
{% if object %}Редактировать сотрудника{% else %}Добавить сотрудника{% endif
%} - HR Platform
{% endblock %}
{% block content %}
<div class="card form-card">
  <!-- Заголовок формы -->
  <h2>
    {% if object %}
        Редактировать сотрудника: {{ object.full_name }}
    {% else %}
        Добавить нового сотрудника
    {% endif %}
  </h2>


<!-- Форма сотрудника -->
  <form method="post" class="employee-form">
    <!-- CSRF-токен для защиты -->
    {% csrf_token %}

    <!-- Отображение ошибок формы -->
    {% if form.errors %}
        <div class="alert alert-danger">
            <h4>Пожалуйста, исправьте следующие ошибки:</h4>
            {{ form.errors }}
        </div>
    {% endif %}

    <!-- Поля формы -->
    <div class="form-grid">
        <!-- Связанный пользователь -->
        <div class="form-group">
            <label for="{{ form.user.id_for_label }}">{{ form.user.label
}}</label>
            {{ form.user }}
            {% if form.user.help_text %}
                <small class="help-text">{{ form.user.help_text }}</small>
            {% endif %}
        </div>

        <!-- ФИО -->
        <div class="form-group">
            <label for="{{ form.full_name.id_for_label }}">{{
form.full_name.label }}</label>
            {{ form.full_name }}
            {% if form.full_name.errors %}
                <div class="field-errors">{{ form.full_name.errors }}</div>
            {% endif %}
        </div>
```

```
        <!-- Дата приема -->
        <div class="form-group">
            <label for="{{ form.date_hired.id_for_label }}">{{
form.date_hired.label }}</label>
            {{ form.date_hired }}
            {% if form.date_hired.errors %}
                <div class="field-errors">{{ form.date_hired.errors }}</div>
```



Продолжение листинга В.2

```
            {% endif %}
        </div>

        <!-- Должность -->
        <div class="form-group">
            <label for="{{ form.position.id_for_label }}">{{
form.position.label }}</label>
            {{ form.position }}
            {% if form.position.errors %}
                <div class="field-errors">{{ form.position.errors }}</div>
            {% endif %}
        </div>

        <!-- Отдел -->
        <div class="form-group">
          <label for="{{ form.department.id_for_label }}">{{
form.department.label }}</label>
            {{ form.department }}
            {% if form.department.errors %}
                <div class="field-errors">{{ form.department.errors }}</div>
            {% endif %}
        </div>

        <!-- Квалификация (полная ширина) -->
        <div class="form-group full-width">
            <label for="{{ form.qualification.id_for_label }}">{{
form.qualification.label }}</label>
            {{ form.qualification }}
            {% if form.qualification.errors %}
                <div class="field-errors">{{ form.qualification.errors
}}</div>
            {% endif %}
            <small class="help-text">
                Опишите образование, навыки, сертификаты и опыт работы
сотрудника
            </small>
        </div>
    </div>

    <!-- Кнопки действий -->
    <div class="form-actions">
        <button type="submit" class="btn btn-primary">
            {% if object %}Сохранить изменения{% else %}Создать сотрудника{%
endif %}
        </button>
        <a href="{% url 'employees' %}" class="btn btn-secondary">
            Отмена
        </a>
    </div>
```

```
  </form>
</div>
{% endblock %}
<!-- Дополнительные стили для формы -->
{% block extra_head %}
<style>
.employee-form {
    max-width: 800px;
```



Продолжение листинга В.2

```
    margin: 0 auto;
}
.form-grid {
    display: grid;
    grid-template-columns: 1fr 1fr;
    gap: 1.5rem;
    margin-bottom: 2rem;
}
.form-group.full-width {
    grid-column: 1 / -1;
}
.form-group label {
    display: block;
    margin-bottom: 0.5rem;
    font-weight: bold;
    color: #333;
}
.form-group input,
.form-group select,
.form-group textarea {
    width: 100%;
    padding: 0.75rem;
    border: 1px solid #ddd;
    border-radius: 4px;
    font-size: 1rem;
}
.form-group textarea {
    resize: vertical;
    min-height: 100px;
}
.help-text {
    display: block;
    margin-top: 0.25rem;
    color: #666;
    font-size: 0.875rem;
}
.field-errors {
    color: #dc3545;
    font-size: 0.875rem;
    margin-top: 0.25rem;
}
.form-actions {
    display: flex;
    gap: 1rem;
    justify-content: flex-start;
    padding-top: 1rem;
    border-top: 1px solid #eee;
}
```

```
@media (max-width: 768px) {
    .form-grid {
        grid-template-columns: 1fr;
    }

    .form-actions {
        flex-direction: column;
    }
```



Продолжение листинга В.2

```
}
</style>
{% endblock %}

employees/employee_confirm_delete.html
Страница подтверждения удаления сотрудника

{% extends 'base.html' %}
<!-- Заголовок страницы -->
{% block title %}Удалить сотрудника - HR Platform{% endblock %}
{% block content %}
<div class="card form-card delete-confirmation">
  <!-- Заголовок с предупреждением -->
  <div class="warning-header">
    <h2>⚠️ Удалить сотрудника</h2>
  </div>

  <!-- Информация об удаляемом сотруднике -->
  <div class="employee-summary">
    <h3>{{ object.full_name }}</h3>
    <div class="employee-details">
        <p><strong>Должность:</strong> {{ object.position }}</p>
        <p><strong>Отдел:</strong> {{ object.department }}</p>
        <p><strong>Дата приема:</strong> {{ object.date_hired|date:"d.m.Y"
}}</p>
    </div>
  </div>

  <!-- Предупреждение об удалении -->
  <div class="warning-message">
    <p>
      <strong>Внимание!</strong> Вы уверены, что хотите удалить сотрудника
      <em>"{{ object.full_name }}"</em>?
    </p>

    <div class="consequences">
        <p>Это действие приведет к:</p>
        <ul>
            <li>Удалению всех данных о сотруднике</li>
            <li>Удалению связанных документов</li>
            <li>Невозможности восстановления информации</li>
        </ul>
    </div>
  </div>

  <!-- Форма подтверждения удаления -->
  <form method="post" class="delete-form">
    <!-- CSRF-токен -->
    {% csrf_token %}
```

```
    <!-- Кнопки действий -->
    <div class="form-actions">
        <button type="submit" class="btn btn-danger">
            ✓ Подтвердить удаление
        </button>
        <a href="{% url 'employees' %}" class="btn btn-secondary">
            ✕ Отмена
```



Продолжение листинга В.2

```
        </a>
    </div>
  </form>
</div>
{% endblock %}
<!-- Дополнительные стили для страницы удаления -->
{% block extra_head %}
<style>
.delete-confirmation {
    max-width: 600px;
    margin: 2rem auto;
    border: 2px solid #dc3545;
}
.warning-header {
    background-color: #f8d7da;
    margin: -1rem -1rem 1rem -1rem;
    padding: 1rem;
    border-bottom: 1px solid #dc3545;
}
.warning-header h2 {
    margin: 0;
    color: #721c24;
}
.employee-summary {
    background-color: #f8f9fa;
    padding: 1rem;
    border-radius: 4px;
    margin-bottom: 1rem;
}
.employee-summary h3 {
    margin: 0 0 1rem 0;
    color: #495057;
}
.employee-details p {
    margin: 0.25rem 0;
    color: #6c757d;
}
.warning-message {
    margin: 1.5rem 0;
}
.consequences {
    background-color: #fff3cd;
    border: 1px solid #ffeaa7;
    border-radius: 4px;
    padding: 1rem;
    margin-top: 1rem;
}
.consequences ul {
```

```
    margin: 0.5rem 0 0 1rem;
    color: #856404;
}
.consequences li {
    margin: 0.25rem 0;
}
.delete-form {
    border-top: 1px solid #dee2e6;
```

Окончание приложения В

Окончание листинга В.2

```
    padding-top: 1rem;
}
.form-actions {
    display: flex;
    gap: 1rem;
    justify-content: center;
}
.btn-danger {
    background-color: #dc3545;
    border-color: #dc3545;
    color: white;
}
.btn-danger:hover {
    background-color: #c82333;
    border-color: #bd2130;
}
@media (max-width: 768px) {
    .form-actions {
        flex-direction: column;
    }
}
```

# ПРИЛОЖЕНИЕ С

РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ HR-ПЛАТФОРМЫ

## 1. РЕГИСТРАЦИЯ И ВХОД В СИСТЕМУ

### 1.1 Регистрация нового пользователя

Для регистрации в системе необходимо выполнить следующие шаги:

1. На главной странице сайта нажмите кнопку «Регистрация» в правом верхнем углу.

2. Заполните форму регистрации:

- Логин (имя пользователя)
- Электронная почта
- Пароль (не менее 8 символов)
- Подтверждение пароля

3. Нажмите кнопку «Зарегистрироваться».

4. На указанную электронную почту будет отправлено письмо для подтверждения регистрации.

5. Перейдите по ссылке из письма для активации аккаунта.

### 1.2 Вход в систему

Для входа в систему выполните следующие действия:

1. На главной странице нажмите кнопку «Вход» в правом верхнем углу.

2. Введите имя пользователя и пароль.

3. Нажмите кнопку «Войти».

## 2. ФУНКЦИИ ДЛЯ РАЗНЫХ РОЛЕЙ ПОЛЬЗОВАТЕЛЕЙ

### 2.1 Администратор

Администратор имеет доступ ко всем функциям системы:

- Управление пользователями (создание, редактирование, блокировка)

- Управление ролями и правами доступа
- Создание и редактирование отделов компании
- Настройка системных параметров
- Доступ ко всем аналитическим отчетам
- Управление шаблонами документов
- Полный доступ к управлению вакансиями

Для доступа к панели администратора:

1. Войдите в систему с учетной записью администратора
2. В главном меню выберите пункт «Администрирование»

**2.2 HR-менеджер**

HR-менеджер имеет доступ к следующим функциям:

- Управление профилями сотрудников
- Управление вакансиями (создание, редактирование, закрытие)
- Обработка откликов кандидатов
- Доступ к HR-аналитике
- Работа с документами (создание, редактирование, генерация)

Для работы с сотрудниками:

1. В главном меню выберите пункт «Сотрудники»
2. Используйте фильтры для поиска нужного сотрудника
3. Нажмите на имя сотрудника для просмотра или редактирования профиля

**2.3 Сотрудник**

Сотрудник имеет доступ к следующим функциям:

- Просмотр и редактирование собственного профиля
- Просмотр доступных вакансий компании

- Просмотр и скачивание своих документов
- Просмотр базовой информации о компании и коллегах

Для просмотра своего профиля:

1. В главном меню выберите пункт «Мой профиль»
2. Для редактирования информации нажмите кнопку «Редактировать»

2.4 Кандидат

Кандидат имеет доступ к следующим функциям:

- Просмотр доступных вакансий
- Подача откликов на вакансии
- Загрузка резюме и сопроводительных писем
- Просмотр статуса своих откликов

Для поиска вакансий:

1. В главном меню выберите пункт «Вакансии»
2. Используйте фильтры для поиска подходящих вакансий
3. Нажмите на название вакансии для просмотра подробной информации

## 3. РАБОТА С ВАКАНСИЯМИ

### 3.1 Создание новой вакансии (для HR-менеджера)

1. В главном меню выберите пункт «Вакансии».
2. Нажмите кнопку «Создать вакансию».
3. Заполните форму создания вакансии:
   - Название должности
   - Описание вакансии
   - Требования к кандидату
   - Условия работы

- Дата закрытия вакансии (опционально)

4. Нажмите кнопку «Сохранить».

**3.2 Редактирование вакансии**

1. В списке вакансий найдите нужную вакансию.

2. Нажмите кнопку «Редактировать» рядом с вакансией.

3. Внесите необходимые изменения.

4. Нажмите кнопку «Сохранить».

**3.3 Архивирование вакансии**

1. В списке вакансий найдите нужную вакансию.

2. Нажмите кнопку «Архивировать».

3. Подтвердите действие в диалоговом окне.

**3.4 Подача отклика на вакансию (для кандидата)**

1. В списке вакансий найдите интересующую вакансию.

2. Нажмите на название вакансии для просмотра подробной информации.

3. Нажмите кнопку «Откликнуться».

4. Заполните форму отклика:

- Загрузите резюме (в формате PDF или DOC)

- Напишите сопроводительное письмо

5. Нажмите кнопку «Отправить отклик».

**3.5 Управление откликами (для HR-менеджера)**

1. В главном меню выберите пункт «Отклики».

2. Используйте фильтры для поиска откликов по вакансиям или статусам.

3. Нажмите на отклик для просмотра подробной информации.

4. Для изменения статуса отклика:

- Выберите новый статус из выпадающего списка

- Добавьте комментарий (опционально)

- Нажмите кнопку «Сохранить»

## 4. УПРАВЛЕНИЕ ДОКУМЕНТАМИ

### 4.1 Создание шаблона документа (для HR-менеджера)

1. В главном меню выберите пункт «Документы».

2. Перейдите на вкладку «Шаблоны».

3. Нажмите кнопку «Создать шаблон».

4. Заполните форму создания шаблона:

- Название шаблона

- Содержание шаблона (используйте переменные в формате {переменная} для автоматической подстановки данных)

- Или загрузите файл шаблона (DOCX)

5. Нажмите кнопку «Сохранить».

### 4.2 Поиск и просмотр документов

1. В главном меню выберите пункт «Документы».

2. Нажмите на название документа для просмотра.

3. Для скачивания документа нажмите кнопку «Скачать».

## 5. АНАЛИТИЧЕСКИЕ ОТЧЕТЫ

### 5.1 Доступные типы отчетов

В системе доступны следующие типы отчетов:

- Отчет по вакансиям

- Статистика по открытым/закрытым вакансиям
- Среднее время закрытия вакансий
- Отчет по сотрудникам
- Статистика по численности персонала
- Распределение сотрудников по отделам
- Анализ текучести кадров
- Отчет по эффективности источников найма
- Количество откликов по каналам привлечения
- Процент успешных наймов по каналам
- Стоимость привлечения кандидатов
- Прогнозный отчет
- Прогноз текучести кадров
- Анализ рисков увольнения
- Прогноз сроков закрытия вакансий

**5.2 Формирование и экспорт отчетов**

1. В главном меню выберите пункт «Аналитика».

2. Выберите тип отчета из списка.

3. Для экспорта отчета нажмите кнопку «Экспорт» и выберите формат (Excel или PDF).

**5.3 Интерпретация результатов**

- Графики и диаграммы
- Столбчатые диаграммы показывают сравнение значений по категориям
- Линейные графики отображают изменение значений во времени
- Круговые диаграммы демонстрируют процентное соотношение

- Прогнозные модели

- Цветовая индикация (зеленый/желтый/красный) показывает уровень риска

- Числовые значения указывают на вероятность события в процентах

- Временные прогнозы указывают ожидаемые сроки в днях

## 6. НАСТРОЙКИ ПРОФИЛЯ И ВЫХОД ИЗ СИСТЕМЫ

### 6.1 Изменение настроек профиля

1. В правом верхнем углу нажмите на профиль.
2. Внесите необходимые изменения в данные профиля.
3. Для изменения пароля введите текущий пароль и новый пароль.
4. Нажмите кнопку «Сохранить».

### 6.2 Выход из системы

1. В правом верхнем углу выберите пункт «Выход».
2. Система завершит сеанс работы и перенаправит вас на страницу входа.