**2 Семестр**

**Раздел 3 Проектирование**

**3.1 Лабораторная работа (ЛР) - 5 Неделя**

# Лабораторная работа №3

**«Технологии промышленной разработки программного обеспечения. Требования»   
(2 семестр)**

**Варианты заданий**

**Постановка задачи**

Провести архитектурное проектирование с получением проектной модели уровня платформы (операционной системы и среды проектирования) по теме из списка тем, использованного в Лабораторной работе №1 текущего курса.

Предполагается, что студент продолжает проектирование по теме, выбранной им в Лабораторной работе №1. По согласовании с преподавателем студент может сменить тему при условии, что по новой теме он проведёт разработку технического залания и концептуальное проектирование.

**Минимальные требования к выполнению заданий**.

1. Должна быть разработана проектная модель программного продукта по выбранной теме, содержащая диаграммы классов и системных взаимодействий не менее двух уровней конкретизации, доходящих до представления платформы (операционной системы и среды проектирования)
2. Проектная модель должна включаться в пояснительную записку или сопровождать её.

**Методические указания**.

Назначение проектируемого продукта – автоматизировать поддержку персоналом объекта функций объекта, указанного как тема проектирования.

В рамках предложенных тем подразумевается обеспечение продуктом не менее двух рабочих мест. Студенту рекомендуется предусмотреть в проекте реализацию этих рабочих мест как клиентов клиент-серверной системы, а продукта – как распределённого приложения с выбором сетевых технологий. Принятие этой рекомендации стимулируется повышением итоговой оценки.

Предложенные темы допускают добавление или усложнение функций, выполнение которых может включать элементы принятия решений. Студенту рекомендуется предусмотреть в проекте расширение темы такого рода. Принятие этой рекомендации стимулируется повышением итоговой оценки.

Студенту рекомендуется согласовать концептуальную модель разрабатываемого варианта продукта (таблицу прецедентов, диаграммы прецедентов, концептуальных классов, системных взаимодействий) прежде, чем приступать к архитектурному проектированию, чтобы убедиться в корректном понимании темы.

**Выбор темы проектирования**

**Темы проектирования (реинжиниринга)**

|  | Тема (вариант) | Характеристика варианта ) | Балл |
| --- | --- | --- | --- |
| 1 | Регистратура поликлиники | Минимальный репертуар функций: расписание приёма, запись на приём, перенос приёма, журнал приёма | 3 |
| 2 | Отделение стационара больницы | фонд палат и койко-мест, состав мед. персонала и его дежурство, регистрация пациентов, диагноз и предписания, история лечения | 6 |
| 3 | Кол-центр скорой помощи | Минимальный репертуар функций: регистрация обращений, состав и расписание бригад, диспетчирование выездов, расписание операторов, переадресация вызовов | 3 |
| 4 | Станция МЧС | Минимальный репертуар функций: регистрация обращений, состав и расписание бригад, диспетчирование выездов с учётом сложности вызова, расписание операторов, переадресация вызовов | 6 |
| 5 | Библиотека | Минимальный репертуар функций: каталог фонда (с глубиной классификации не менее 6 по отдельным аспектам), Минимальный репертуар функций: реестр читателей, оборот литератур, обновление фондов | 6 |
| 6 | Театральная касса | Минимальный репертуар функций: расписание спектаклей, заказ и возврат билетов, входной контроль зрителей | 6 |
| 7 | Детский сад | Минимальный репертуар функций: состав групп детей, фонд помещений, размещение групп, состав и назначение воспитателей, учёт посещения | 3 |
| 8 | Склад | Минимальный репертуар функций: фонд складских помещений, реестр объектов, принятых на хранение (с привязкой к срокам хранения, владельцам и размещению),бронирование | 6 |
| 9 | Кухня | Минимальный репертуар функций: каталог рецептуры, реестр оборудования, запас продуктов, текущий график приготовления, служба приёма и выдачи заказов, служба поставки продуктов | 9 |
| 10 | Магазин online | Минимальный репертуар функций: сайт с окнами рекламы товаров и окном заказов, финансовая служба, служба доставки, актуализация каталога | 3 |
| 11 | Отделение банка | Минимальный репертуар функций: счета клиентов, приход и расход денежных средств, начисление процентов, собственные средства и их движение | 6 |
| 12 | Умный дом | Минимальный репертуар функций: контроль дверей, температуры и освещения. | 6 |
| 13 | Жилищный кооператив | Минимальный репертуар функций: учёт недвижимости, реестр собственников, квартплата, текущий ремонт, поддержка деятельности правления | 3 |
| 14 | Продажа авиабилетов | Минимальный репертуар функций: расписание полётов, парк самолётов, продажа и прочие операции с билетами, оповещение и регистрация пассажиров | 6 |
| 15 | Гостиница | Минимальный репертуар функций: гостиничный фонд, бронирование, заселение | 3 |
| 16 | Метеостанция | Минимальный репертуар функций: сбор местных метеоданных, составление метеосводки, составление метеопрогноза на основе получения и обработки спутниковых данных и соседских сводок | 6 |
| 17 | Ресторан | Минимальный репертуар функций: бронирование мест (столов), получение заказа и передача его на кухню, подача заказа и расчёт с клиентом | 9 |
| 18 | Транспортная контора | Минимальный репертуар функций: автопарк, приём заказов, формирование бригад и расписания перевозок | 3 |
| 19 | Туристическое агентство | Минимальный репертуар функций: каталог маршрутов, реестр агентов (мест проживания, агентств-посредников) | 6 |
| 20 | Расписание в средней школе | Минимальный репертуар функций: расписание, задания, явка учеников, успеваемость | 6 |

**Критерии оценки**

| 1. | Уровень сложности темы | - балл в таблице тем проектирования (реинжиниринга)  - с применением сетевых технологий  - с элементами принятия решений | 3 – 9  +2  +2 | |
| --- | --- | --- | --- | --- |
| 2. | Качество оформления проекта | * стиль (в т.ч.: имена, отступы и проч.) * корректность диаграмм | +/- 1  балл | |
| 3. | Качество оформления проекта | Грамотность комментариев, корректность диаграмм, полнота выражения замысла, соблюдение шаблонов GRASP и других принципов ООП | +/- 1 балл | |
|  | Итого | | | 0-15 | |

# Оглавление

[**Лабораторная работа №3 1**](#_m4ias627c5bk)

[**Оглавление 4**](#_g1n2y3n1cuo3)

[**Определения 5**](#_o7plbxc3iqxn)

[**Архитектурное проектирование веб-сервиса для подбора кандидатов 6**](#_og6n60i4lqma)

[**Цель 6**](#_nh30x86yql6v)

[**1. Уточнение архитектурного уровня 6**](#_xt46oocyd29z)

[1.1 Сервис загрузки и анализа резюме 6](#_ihhqeod4euds)

[1.2 Сервис управления вакансиями 7](#_2j9szzkwola4)

[1.3 Сервис сопоставления кандидатов 7](#_g859eb1i9qf0)

[1.4 Сервис управления пользователями 7](#_lpdkn1xnp67v)

[1.5 Фронтенд 7](#_v4ix3x30nkom)

[1.6 Обоснование выбора технологического стека и средств реализации 8](#_dzmecxvujvdp)

[**2. Операционная система и среда проектирования 10**](#_6r6yl48vdgym)

[2.1 Операционная система 10](#_6gkx5vv2d8s1)

[2.2 Среда разработки и вспомогательные инструменты 11](#_d82lya72q1mv)

[**3. Диаграммы 13**](#_bb50hajcjg7v)

[3.1 Диаграмма классов (пример на уровне Django-моделей) 13](#_gzmv49qy8z13)

[3.2 Диаграмма системных взаимодействий (UML sequence diagrams) 15](#_79yrggv39hoz)

[3.3 Диаграмма развертывания (UML Deployment Diagram) 16](#_l95qe7d27dqk)

[**4. Уровни конкретизации 18**](#_14qabfmt6vq2)

[4.1 Взаимодействие микросервисов на уровне API (REST/HTTP) 18](#_35sxn7g7bhkv)

[4.2 Внутренние компоненты микросервисов: модели, контроллеры, хранилище и ML-интеграция 20](#_wutxk8d5rpx1)

# 

# Определения

Проектирование – этап разработки программного обеспечения, включающий создание структурной и архитектурной модели будущей системы с учётом функциональных и технических требований.

Проектная модель уровня платформы – Модель программной системы, учитывающая особенности операционной системы и среды проектирования. Содержит детализацию взаимодействий компонентов на уровне используемой аппаратно-программной платформы.

Техническое задание (ТЗ) – документ, описывающий цели, требования и условия реализации программного продукта. Может быть разработан при изменении темы лабораторной работы.

Концептуальное проектирование – начальный этап проектирования, включающий формирование общей идеи системы, её целей, функций и основных компонентов без детализации реализации.

Диаграмма классов – графическая модель в UML, отображающая структуру системы через описание классов, их атрибутов, методов и связей между ними.

Диаграмма системных взаимодействий

UML-диаграмма, демонстрирующая обмен сообщениями между элементами системы. Может включать последовательности вызовов и взаимодействия компонентов.

Уровень конкретизации – степень детализации проектной модели. Подразумевается, что диаграммы должны быть представлены как на общем уровне, так и в более подробном виде.

Рабочее место (в контексте ПО) – локальная пользовательская среда, поддерживающая выполнение определённых функций в рамках информационной системы.

Клиент-серверная система – архитектура распределённых приложений, в которой клиенты (рабочие места) запрашивают данные у сервера, который управляет ресурсами и предоставляет услуги.

Распределённое приложение – программный продукт, состоящий из частей, работающих на разных устройствах в сети, взаимодействующих друг с другом для достижения общей цели.

Сетевые технологии – средства и протоколы передачи данных по сети, используемые при разработке клиент-серверных и распределённых приложений.

Элементы принятия решений – функциональные части программы, позволяющие системе анализировать данные и выбирать действия на основе логических условий или алгоритмов.

Таблица прецедентов – документ UML-анализа, описывающий сценарии использования системы (функции, выполняемые актёрами).

Диаграмма прецедентов – UML-диаграмма, отображающая отношения между пользователями (актёрами) и возможностями (прецедентами) системы.

Диаграмма концептуальных классов – UML-диаграмма, показывающая абстрактные представления объектов предметной области и связи между ними на ранних этапах проектирования.

GRASP (General Responsibility Assignment Software Patterns) – набор принципов назначения ответственностей классам в ООП, включая паттерны: Creator, Information Expert, Low Coupling, High Cohesion и другие.

Объектно-ориентированные принципы (ООП) – основополагающие подходы к разработке ПО, такие как инкапсуляция, наследование, полиморфизм, абстракция.

Критерии оценки – параметры, по которым оценивается качество выполненной лабораторной работы. Включают сложность темы, стиль оформления, корректность диаграмм, комментарии и использование принципов ООП.

# Архитектурное проектирование веб-сервиса для подбора кандидатов

# Цель

Создание проектной модели уровня платформы: операционной системы, среды проектирования и взаимодействий между компонентами веб-сервиса с применением принципов DDD, микросервисной архитектуры, и технологии контейнеризации.

# 1. Уточнение архитектурного уровня

В рамках архитектурного проектирования веб-сервиса для автоматизации подбора кандидатов было принято обоснованное решение о реализации программной системы в виде микросервисной архитектуры. Данный подход обеспечивает модульность, масштабируемость и гибкость, что критически важно при работе с разнородными компонентами, использующими машинное обучение, обработку естественного языка и взаимодействие с внешними API.

В отличие от монолитной архитектуры, где все компоненты связаны в единое целое, микросервисная архитектура позволяет декомпозировать систему на независимые сервисы, каждый из которых реализует конкретную бизнес-функциональность. Это упрощает разработку, упрощает внедрение новых функций, ускоряет деплой и масштабирование, а также повышает устойчивость системы к частичным отказам.

Далее рассмотрим ключевые компоненты которые будут созданы для структуры веб-сервиса.

## 1.1 Сервис загрузки и анализа резюме

Данный компонент обеспечивает прием пользовательских данных (резюме в формате PDF, DOCX и др.) и выполняет их последующую обработку с использованием алгоритмов обработки естественного языка (NLP). Сервис реализует предобработку текста (tokenization, stop-word removal, лемматизация), извлечение сущностей (Named Entity Recognition), а также семантическую векторизацию с применением моделей BERT или аналогичных трансформеров.

Сервис обладает высокой вычислительной нагрузкой и, вследствие этого, развертывается в отдельной изолированной среде с возможностью горизонтального масштабирования через Kubernetes Horizontal Pod Autoscaler. Для оптимизации обработки реализуется очередность задач и поддержка асинхронной архитектуры (например, с использованием Celery и Redis).

## 1.2 Сервис управления вакансиями

Этот сервис отвечает за создание, редактирование, публикацию и деактивацию вакансий. Он предоставляет REST API для работодателей и HR-специалистов, а также хранит связанное описание вакансии, требования, зарплатные ожидания и предпочтительные навыки.

Модель вакансии снабжена валидацией: например, обязательным наличием ключевых компетенций и описания обязанностей. Сервис взаимодействует с базой данных через ORM, реализованную средствами Django ORM или SQLAlchemy. Помимо CRUD-операций, предусмотрена возможность интеграции с внешними источниками вакансий, такими как HeadHunter API, для импорта объявлений.

## 1.3 Сервис сопоставления кандидатов

Является ядром интеллектуальной логики платформы. Он отвечает за вычисление степени соответствия между резюме и вакансиями на основе результатов, полученных от NLP-анализа. В основе сопоставления — алгоритмы семантического сравнения, использование cosine similarity, TF-IDF, либо кастомных нейросетевых моделей.

Сервис реализует поддержку алгоритмов машинного обучения, обученных на размеченных данных (например, успешные наймы), и может быть переобучен на основе пользовательских решений (feedback loop). Архитектурно выделяется как stateless-сервис, вызываемый по запросу, с ML-инференсом, развернутым через TensorFlow Serving, TorchServe или KFServing.

## 1.4 Сервис управления пользователями

Данный компонент обеспечивает функции регистрации, аутентификации и авторизации пользователей платформы: кандидатов, HR-специалистов и администраторов. Сервис реализует контроль доступа по ролям (RBAC), управление сессиями, восстановление пароля, подтверждение через email/SMS и OAuth 2.0-авторизацию (например, через Google или LinkedIn).

Для обеспечения безопасности реализуется двухфакторная аутентификация и шифрование персональных данных. Компонент изолирован в отдельный сервис для минимизации рисков при обработке чувствительной информации. Также сервис предоставляет REST API для взаимодействия с другими компонентами системы.

## 1.5 Фронтенд

Клиентская часть веб-приложения реализуется на базе современного JavaScript-фреймворка (React или Vue.js) и служит интерфейсом для всех ролей пользователей. Фронтенд получает данные через REST API от вышеперечисленных сервисов и отображает их в виде адаптивных страниц.

Особое внимание уделяется UX-дизайну, так как удобство и интуитивная навигация — ключевые требования для повышения конверсии. Фронтенд изолирован от логики бэкенда и может быть развернут на отдельных серверах или CDN с использованием Nginx в качестве реверс-прокси.

## 1.6 Обоснование выбора технологического стека и средств реализации

Для обеспечения стабильной, масштабируемой и отказоустойчивой работы распределённого веб-сервиса была выбрана технологическая связка, включающая Docker и Kubernetes для контейнеризации и оркестрации, PostgreSQL в качестве основной системы управления базами данных, Django + Django REST Framework для реализации серверной логики, а также React (или Vue.js) — в качестве клиентской технологии для построения пользовательского интерфейса. Выбор каждого из компонентов обусловлен их зрелостью, активным сообществом, высоким уровнем совместимости и соответствием современным требованиям к промышленной разработке программного обеспечения.

Docker и Kubernetes: контейнеризация и оркестрация

Docker представляет собой современную платформу контейнеризации, позволяющую изолировать каждую компоненту веб-сервиса (будь то модуль загрузки резюме или сервис сопоставления кандидатов) в собственную среду выполнения, содержащую все зависимости. Это обеспечивает предсказуемость окружения и облегчает процессы разработки, тестирования и развертывания.

Каждый микросервис упаковывается в отдельный Docker-контейнер, что обеспечивает:

1. минимальные накладные расходы на ресурсы по сравнению с виртуальными машинами,
2. мгновенный запуск новых экземпляров сервисов,
3. полную изоляцию зависимостей.

Для управления жизненным циклом контейнеров и обеспечения оркестрации используется Kubernetes (K8s). Это индустриальный стандарт, обеспечивающий:

1. автоматическое масштабирование сервисов в зависимости от нагрузки,
2. отказоустойчивость (например, автоматический перезапуск упавших контейнеров),
3. декларируемое управление конфигурациями через YAML-манифесты,
4. балансировку нагрузки и безопасное обновление версий (rolling updates, blue-green deployment),
5. интеграцию с современными DevOps-практиками (например, GitOps через ArgoCD или FluxCD).

Совместное использование Docker и Kubernetes позволяет разработке и эксплуатации системы быть независимыми друг от друга, следуя принципам CI/CD и инфраструктуры как кода (IaC).

PostgreSQL: объектно-реляционная СУБД

Для хранения резюме, вакансий, профилей пользователей, логов и других структурированных данных в проекте используется PostgreSQL — объектно-реляционная система управления базами данных, зарекомендовавшая себя как стабильное, мощное и гибкое решение для корпоративного уровня.

Ключевые причины выбора PostgreSQL:

1. расширенная поддержка типов данных, включая JSONB (удобно для хранения полуструктурированных данных и логов),
2. транзакционная согласованность (ACID) — критично для обеспечения целостности пользовательских данных и заказов,
3. встроенные средства полнотекстового поиска, применимые для предварительного анализа резюме,
4. поддержка индексов (GIN, GiST) и материализованных представлений, что ускоряет подбор по фильтрам и поисковым критериям,
5. совместимость с Django ORM, что обеспечивает гибкость в разработке.

PostgreSQL также демонстрирует отличную совместимость с распределёнными системами и легко масштабируется как вертикально, так и горизонтально (с использованием репликации, партиционирования и шардирования).

Django и Django REST Framework: реализация backend-логики

Для реализации серверной части платформы выбран фреймворк Django на языке Python, в связке с Django REST Framework (DRF) для построения RESTful API. Django обеспечивает высокий уровень абстракции и ускоряет разработку благодаря встроенным модулям: система аутентификации, панель администратора, маршрутизация, ORM и другие.

Аргументы в пользу Django и DRF:

1. большое количество готовых решений — сокращает время на разработку и тестирование,
2. интеграция с PostgreSQL на уровне ORM без потери производительности,
3. высокий уровень безопасности по умолчанию (CSRF, XSS, SQL Injection и др.),
4. структурированная архитектура MVC, упрощающая командную разработку,
5. возможность расширения через микросервисы, реализованные как отдельные Django-приложения,
6. совместимость с ML/AI-библиотеками Python — важное преимущество для задач анализа резюме и классификации кандидатов.

DRF, в свою очередь, позволяет быстро создавать API-эндпойнты с поддержкой сериализации, пагинации, фильтрации и аутентификации, а также формирует удобный browsable интерфейс для тестирования API.

React / Vue.js: клиентский интерфейс

Для построения интерфейса пользователя применяются современные JavaScript-фреймворки — React или Vue.js, каждый из которых обладает своими преимуществами. Оба фреймворка позволяют реализовать одностраничное приложение (SPA), обеспечивающее интерактивность, быстрый отклик на действия пользователя и снижение количества HTTP-запросов.

React отличается:

1. компонентным подходом и использованием JSX,
2. высокой производительностью за счёт виртуального DOM,
3. широкой экосистемой и поддержкой TypeScript,
4. возможностью интеграции с Redux для управления состоянием.

Vue.js является более легковесным и простым для начального освоения, сохраняя при этом мощный функционал:

1. декларативный шаблонный синтаксис,
2. двухсторонняя привязка данных (two-way binding),
3. низкий порог вхождения и высокая скорость разработки.

Фронтенд взаимодействует с бэкендом через REST API, поддерживает адаптивную вёрстку и может быть дополнен PWA-функциональностью. Для улучшения производительности предполагается предварительная сборка и минимизация ресурсов (Webpack/Vite), а для управления маршрутами и авторизацией — использование библиотек Vue Router или React Router.

# 2. Операционная система и среда проектирования

## 2.1 Операционная система

Выбор операционной системы и среды проектирования является ключевым этапом архитектурного проектирования, поскольку напрямую влияет на устойчивость, масштабируемость, воспроизводимость и безопасность создаваемого программного продукта. В условиях промышленной разработки распределённых веб-сервисов, использующих микросервисную архитектуру и облачную инфраструктуру, особое значение приобретают средства контейнеризации и автоматизированного развертывания. Ниже приведено обоснование выбранного стека системных и инструментальных средств.

Ubuntu Server 22.04: базовая операционная система

В качестве основной операционной системы для хостов, обслуживающих продакшн- и staging-контейнеры, выбрана Ubuntu Server 22.04 LTS. Это одно из самых популярных и широко поддерживаемых серверных дистрибутивов Linux, отличающееся стабильностью, высокой производительностью и широким сообществом.

Преимущества использования Ubuntu Server:

1. Статус долгосрочной поддержки (LTS): гарантированная поддержка обновлений безопасности на протяжении пяти лет, совместимость с Docker и Kubernetes без необходимости в дополнительной настройке ядра, активное сообщество и обилие документации, включая официальную поддержку в облаках AWS, GCP, Azure, легкость автоматизации благодаря поддержке cloud-init, Ansible и других инструментов конфигурации.
2. Ubuntu Server является де-факто стандартом в DevOps-среде, что гарантирует высокую совместимость со всеми инструментами контейнеризации и CI/CD.

Docker и Docker Compose: локальная среда разработки

Для локальной разработки, тестирования и отладки микросервисов применяется платформа Docker совместно с Docker Compose. Docker предоставляет контейнеризированную среду, позволяющую разрабатывать каждый сервис независимо, без зависимости от системных библиотек хост-машины.

Docker Compose используется для управления многоконтейнерной конфигурацией, обеспечивая разработчику возможность поднятия полной среды (включая backend, frontend, базу данных, ML-сервис) одной командой. Это позволяет:

1. обеспечить воспроизводимость окружения на различных этапах CI/CD-пайплайна,
2. упростить отладку межсервисного взаимодействия (например, между сервисом анализа резюме и базой данных),
3. быстро переключаться между различными конфигурациями (production, staging, development).

Таким образом, Docker Compose является эффективным инструментом для изоляции окружения разработчика, обеспечивая быструю итеративную разработку без зависимости от глобальных настроек операционной системы.

Kubernetes: продукционное окружение и оркестрация

Для промышленной эксплуатации веб-сервиса в облачной или гибридной инфраструктуре используется Kubernetes — индустриальный стандарт для оркестрации контейнеров. Kubernetes обеспечивает: автоматическое масштабирование микросервисов в зависимости от загрузки (Horizontal Pod Autoscaler), распределение нагрузки и управление отказоустойчивостью через ReplicaSets и StatefulSets, декларируемое управление конфигурацией и секретами (ConfigMaps, Secrets), стратегии обновлений без простоя (rolling update, blue-green deployment).

В зависимости от среды развёртывания, возможно использование:

1. Minikube — локальная реализация Kubernetes для тестирования и обучения,
2. Google Kubernetes Engine (GKE), Amazon EKS, Yandex Managed Service for Kubernetes — управляемые облачные кластеры, снижающие операционные издержки на администрирование и повышающие безопасность.
3. Kubernetes позволяет добиться максимальной гибкости и масштабируемости за счёт автоматизации жизненного цикла сервисов, что критично для высоконагруженных и быстроразвивающихся систем, основанных на микросервисной архитектуре.
4. Nginx или Traefik: Ingress-контроллер
5. В качестве шлюза входного трафика в кластер Kubernetes выбран Ingress-контроллер, реализованный на базе Nginx или Traefik. Оба решения обеспечивают маршрутизацию HTTP(S)-трафика к внутренним сервисам, защиту от некорректных запросов и интеграцию с TLS-сертификатами.
6. Nginx — зрелый, высокопроизводительный HTTP-сервер и обратный прокси, обеспечивающий тонкую настройку правил маршрутизации, SSL-терминацию и фильтрацию по IP-адресам. Поддерживается Kubernetes-плагином ingress-nginx и легко масштабируется.
7. Traefik — современный Ingress-контроллер, ориентированный на микросервисные системы. Имеет встроенную поддержку автоматического получения сертификатов Let's Encrypt, динамическую конфигурацию маршрутов через Kubernetes API и визуальную панель мониторинга.

Окончательный выбор между Nginx и Traefik осуществляется на основе приоритетов: если критична производительность и стабильность — предпочтение отдается Nginx; если приоритет — гибкость и автоматизация, особенно в DevOps-среде, — то выбирается Traefik.

## 2.2 Среда разработки и вспомогательные инструменты

Для реализации и сопровождения промышленного веб-сервиса необходимо использование профессиональной среды разработки, поддерживающей высокий уровень автоматизации, отладки, тестирования и управления зависимостями. В разработке проекта были задействованы ведущие инструменты в области веб-разработки, анализа данных и DevOps, отобранные на основе критериев производительности, совместимости с выбранным стеком технологий и соответствия требованиям командной разработки. Ниже приведено обоснование выбора каждого компонента среды.

Backend: PyCharm + Python 3.10 + Django + PostgreSQL

Разработка серверной логики осуществляется в среде PyCharm Professional, которая предоставляет широкие возможности для работы с Python и Django-проектами. Поддержка автогенерации кода, встроенный инспектор качества, интеграция с Git, дебаггер и встроенный терминал делают эту среду максимально удобной для разработчиков и соответствующей требованиям индустриального уровня.

Использование Python 3.10 продиктовано необходимостью поддержки современных языковых конструкций (например, сопоставления с образцом — structural pattern matching), улучшенной производительности интерпретатора и совместимости с последними версиями Django, библиотек машинного обучения и системных инструментов.

Фреймворк Django, как уже было обосновано ранее, обеспечивает модульность, безопасность и встроенную поддержку административного интерфейса, что значительно сокращает время разработки. В связке с PostgreSQL обеспечивается эффективная работа с транзакционными данными, поддержка полнотекстового поиска и возможность выполнения сложных аналитических запросов. Используемая связка полностью совместима с Django ORM, что позволяет обеспечить переносимость и масштабируемость серверной логики.

Frontend: Visual Studio Code + React + TypeScript

Для клиентской части используется легковесный и расширяемый редактор Visual Studio Code, разработанный Microsoft и обладающий широкой экосистемой плагинов, включая поддержку React, TypeScript, ESLint, Prettier и интеграцию с Git. VS Code обеспечивает высокую скорость работы, удобство настройки окружения и отличную поддержку отладчиков и терминалов, что делает его идеальным выбором как для индивидуальных разработчиков, так и для командной разработки.

Фреймворк React выбран как основа интерфейса благодаря своей компонентной модели, поддержке виртуального DOM и высокой производительности при создании одностраничных приложений (SPA). Использование TypeScript — надмножества JavaScript с поддержкой статической типизации — позволяет существенно повысить надёжность кода, уменьшить количество ошибок на этапе выполнения и упростить сопровождение проекта в условиях масштабирования команды.

Такое сочетание инструментов гарантирует совместимость с RESTful API backend-сервиса, адаптивность интерфейса под различные устройства и расширяемость пользовательского интерфейса за счёт повторно используемых компонент.

ML-модели: Jupyter Notebook + HuggingFace Transformers

Для разработки, тестирования и валидации моделей обработки естественного языка (NLP) используется связка Jupyter Notebook и библиотека HuggingFace Transformers.

Jupyter предоставляет интерактивную среду, идеально подходящую для исследования данных, визуализации результатов и построения прототипов моделей. Использование Jupyter позволяет вести разработку моделей в формате документируемых вычислений (notebook-based research), что критично при проведении экспериментов с различными архитектурами и параметрами моделей.

Библиотека Transformers от HuggingFace предоставляет доступ к state-of-the-art трансформерным моделям (BERT, RoBERTa, DistilBERT и др.), которые используются для векторизации резюме и описаний вакансий. Она поддерживает как предобученные модели, так и дообучение на кастомных датасетах, что позволяет обеспечить высокую точность в задаче семантического сопоставления профилей кандидатов и требований вакансий. Дополнительно, библиотека интегрируется с TensorFlow и PyTorch, обеспечивая гибкость в выборе фреймворка машинного обучения.

DevOps: GitHub Actions или GitLab CI/CD, ArgoCD (опционально)

Проектная инфраструктура поддерживает автоматизированные процессы сборки, тестирования и развертывания с использованием CI/CD пайплайнов. В зависимости от выбранного репозитория используются:

GitHub Actions — для проектов, размещённых на GitHub, с возможностью создания пайплайнов на YAML, интеграции с Docker, публикации образов и деплоя в Kubernetes;

GitLab CI/CD — аналогичная система для GitLab, с полной интеграцией процессов сборки, тестирования и сканирования уязвимостей.

Пайплайны CI/CD включают: автоматическую сборку и тестирование каждого коммита, создание Docker-образов и загрузку их в реестр (Docker Hub или GitLab Container Registry), деплой образов в кластер Kubernetes с последующим обновлением Ingress.

Для организации GitOps-подхода (инфраструктура как код + управление конфигурацией через Git) может быть дополнительно использован ArgoCD — инструмент синхронизации Kubernetes-кластера с репозиторием конфигураций. ArgoCD обеспечивает: мониторинг изменений в Git и автоматический деплой YAML-манифестов, поддержку стратегий отката и истории изменений, визуальный интерфейс для отслеживания состояния кластера.

Это позволяет достигнуть полной прозрачности процессов развертывания, сократить число ошибок при ручной настройке и ускорить доставку обновлений до продуктивного окружения.

# 3. Диаграммы

## 3.1 Диаграмма классов (пример на уровне Django-моделей)

На диаграмме классов представлены ключевые сущности, лежащие в основе логики веб-сервиса для подбора кандидатов: User, Resume, Vacancy, Employer, Application. Они соответствуют основным агрегатам, выделенным в рамках паттерна предметно-ориентированного проектирования (Domain-Driven Design), и отражают доменную модель системы. Диаграмма отображается в соответствующие Python классы

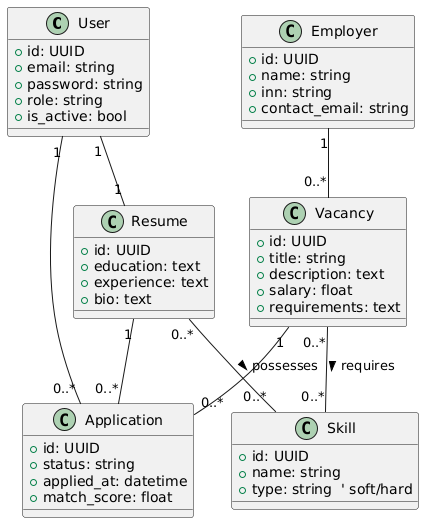
Обоснование элементов и связей:

1. User — универсальная сущность, представляющая зарегистрированного пользователя платформы. Обладает базовыми атрибутами (email, role, is\_active) и может быть как соискателем, так и представителем работодателя. Связь: один пользователь может быть связан с одним Resume и несколькими Application.
2. Resume — сущность, описывающая квалификационный профиль кандидата. Содержит поля: education, experience, skills, bio. Один User может иметь одно Resume (связь один-к-одному).
3. Vacancy — рабочая позиция, опубликованная работодателем. Содержит такие атрибуты, как title, description, salary, requirements, а также списки hard\_skills и soft\_skills. Одна компания (Employer) может публиковать множество вакансий.
4. Employer — представляет организацию, размещающую вакансии. Хранит информацию о названии, ИНН, и контактных данных. Связь: один Employer может иметь множество Vacancy.
5. Application — отклик кандидата на конкретную вакансию. Является связующей сущностью между Resume и Vacancy. Содержит поля status, applied\_at, match\_score (результат машинного сопоставления).

Связи ManyToMany:

1. Resume ↔ Skill (один кандидат может обладать множеством навыков, один навык может встречаться у разных кандидатов).
2. Vacancy ↔ Skill — для сопоставления требований к должности и компетенций кандидатов.

Дополнительно сущность Skill разделяется по типу: hard (технические, измеримые) и soft (межличностные и когнитивные). Это разделение важно для точного подбора по профилю и потенциального применения моделей машинного обучения.



## 3.2 Диаграмма системных взаимодействий (UML sequence diagrams)

Для иллюстрации взаимодействия между ключевыми компонентами системы в рамках пользовательских сценариев были разработаны диаграммы последовательностей. Они позволяют наглядно представить динамику вызовов, поток управления и передачи данных между участниками (акторами и компонентами системы) в процессе выполнения определённого действия.

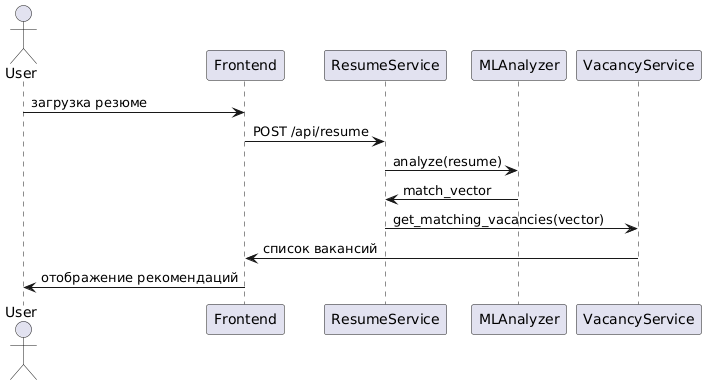
1. Подача резюме кандидатом

Сценарий начинается с действия пользователя — загрузки резюме через веб-интерфейс. После этого фронтенд-часть (обычно реализованная на React или Vue.js) формирует HTTP-запрос методом POST и направляет его в ResumeService. Использование именно POST обусловлено тем, что в данном случае происходит создание новой сущности — загружается резюме, содержащее потенциально чувствительные пользовательские данные. Метод POST в REST-интерфейсе семантически означает «создать новый ресурс» и позволяет передавать вложенные данные в теле запроса (например, файл, JSON-структуру и т.п.).

На диаграмме POST отображается как действие, инициированное пользователем, передающее данные через frontend в backend-сервис, и оно отмечает границу между клиентской и серверной логикой. Его наличие подчёркивает факт внешнего взаимодействия с API, а не локального вызова внутри одного компонента.

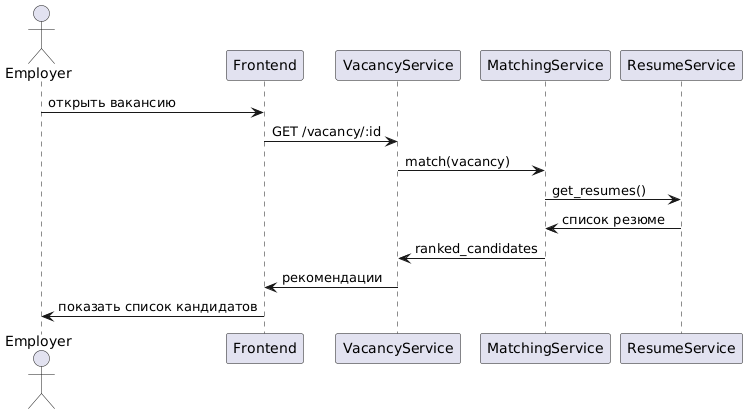
После приёма запроса ResumeService выполняет первичную валидацию и парсинг файла, преобразуя содержимое в структурированный формат. Затем данные передаются в MLAnalyzer, который реализует семантическую обработку текста с применением трансформерных моделей. Полученные векторы и выделенные сущности сохраняются, и по их результатам инициируется вызов в VacancyService — с целью вычисления наиболее релевантных вакансий на основе извлечённых данных.

Все перечисленные компоненты (Frontend, ResumeService, MLAnalyzer, VacancyService) представлены на диаграмме как отдельные элементы распределённой системы, взаимодействующие между собой по протоколу HTTP/REST. Это означает, что каждый из сервисов работает как изолированный процесс (или контейнер) и обменивается данными через стандартизованные web-интерфейсы. Такая архитектура обеспечивает слабую связанность и масштабируемость, позволяя развивать и разворачивать компоненты независимо друг от друга.



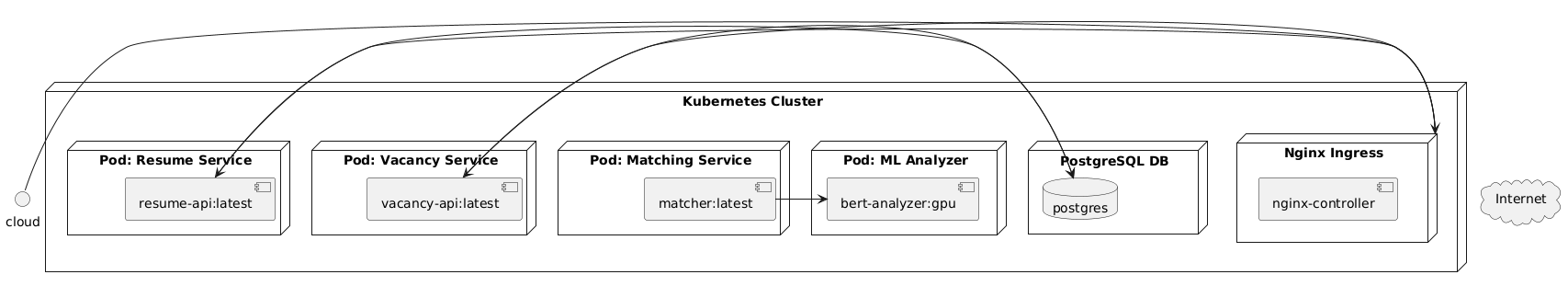
2. Запрос рекомендаций по вакансии

Работодатель через интерфейс инициирует подбор кандидатов под опубликованную вакансию. Запрос отсылается в VacancyService, который, в свою очередь, вызывает MatchingService для анализа требований. Затем происходит семантическое сравнение с существующими резюме, полученными из ResumeService.



## 3.3 Диаграмма развертывания (UML Deployment Diagram)

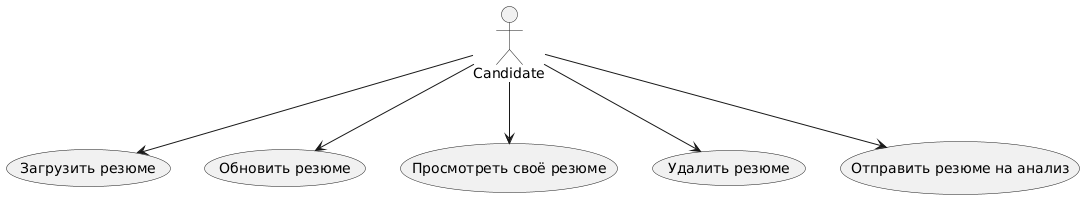
Диаграмма развертывания отражает физическую инфраструктуру и показывает, как компоненты системы развёрнуты на серверах, контейнерах и хостах. Это особенно важно в микросервисной архитектуре, где каждый сервис разворачивается независимо.



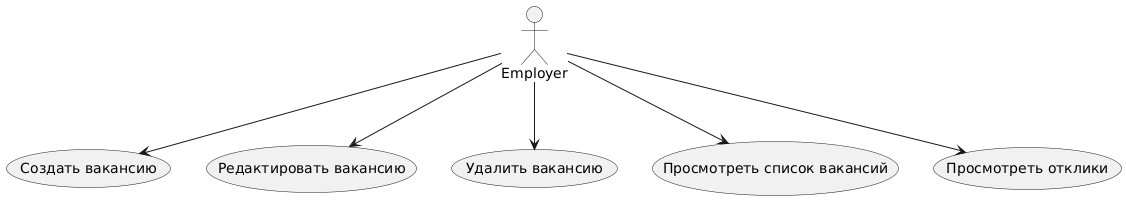
3.4 Диаграмма прецедентов (UML Use Case Diagram)

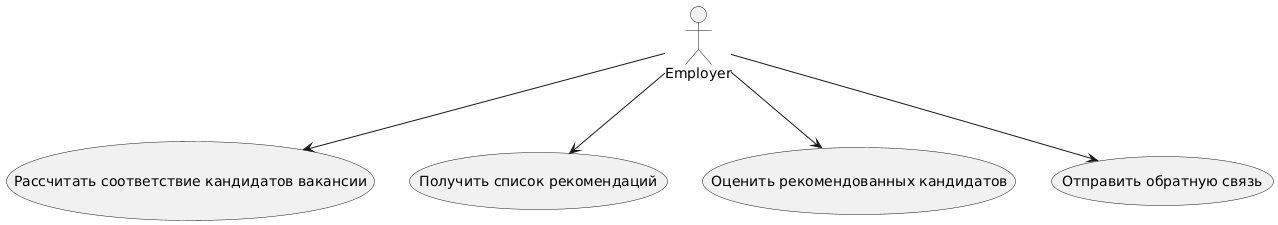
1. Модуль загрузки и обработки резюме (ResumeService)

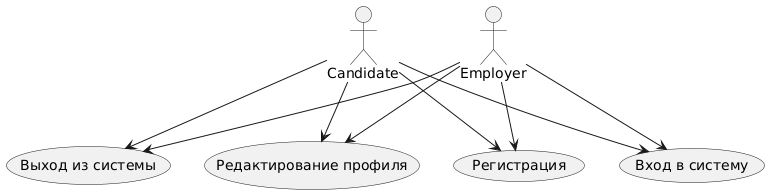
Модуль отвечает за приём, валидацию, хранение и обновление резюме. Также он координирует работу с модулем семантического анализа.

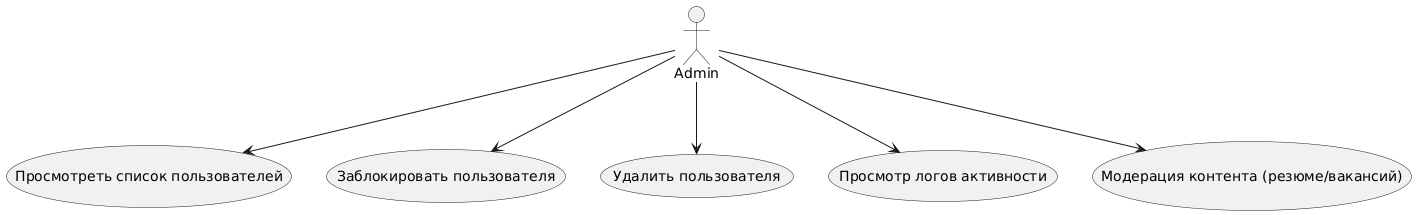
  
2. Модуль управления вакансиями (VacancyService)

Этот модуль предоставляет интерфейс для создания и редактирования вакансий работодателем, а также для получения откликов на них.

  
3. Модуль сопоставления кандидатов и вакансий (MatchingService)

Сервис вычисляет рейтинг соответствия кандидатов и вакансий, получая данные из двух других модулей.  
  
4. Модуль аутентификации и управления пользователями (User/Auth Service)

Модуль обслуживает регистрацию, вход, выход и управление профилем пользователя.  
  
5. Модуль администрирования (Admin Interface)

Модуль предназначен для работы администратора системы: управление пользователями, просмотр логов, модерация данных.  


Такой подход — декомпозиция диаграммы прецедентов по функциональным модулям — отражает архитектурную и логическую структуру системы. Каждая диаграмма сфокусирована на взаимодействии пользователей с отдельным компонентом, что упрощает анализ требований, тестирование и планирование функциональных расширений.

# 4. Уровни конкретизации

В соответствии с требованиями методологии архитектурного проектирования, модель веб-сервиса должна быть декомпозирована на уровни конкретизации, начиная с макроструктуры взаимодействия компонентов и заканчивая низкоуровневыми деталями реализации. Первый уровень конкретизации охватывает архитектурное взаимодействие микросервисов на уровне API. Это уровень интерсервисной коммуникации, реализуемый с помощью REST-протокола поверх HTTP, что обеспечивает совместимость, стандартизацию и модульность системы.

## 4.1 Взаимодействие микросервисов на уровне API (REST/HTTP)

Основная архитектурная модель реализована в виде набора специализированных микросервисов, каждый из которых отвечает за строго определённую область функциональности. Между собой они взаимодействуют через REST API, используя формат JSON для передачи данных и стандартные HTTP-методы (GET, POST, PUT, DELETE). Такой подход обеспечивает слабую связанность компонентов, их независимую разработку, масштабирование и тестирование.

1. Resume Service

Отвечает за приём, хранение и анализ резюме. Предоставляет API-эндпойнты:

1. POST /api/resume/ — загрузка и валидация резюме,
2. GET /api/resume/{id}/ — получение информации по резюме,
3. GET /api/resume/{id}/vector/ — вызов NLP-модуля для получения векторного представления.

Этот микросервис является источником данных для Matching Service и анализируется MLAnalyzer через REST-вызовы.

2. Vacancy Service

Обеспечивает управление вакансиями, их публикацию и редактирование. Эндпойнты:

1. POST /api/vacancy/ — создание новой вакансии,
2. GET /api/vacancy/{id}/ — просмотр подробностей вакансии,
3. PUT /api/vacancy/{id}/ — редактирование требований и описания,
4. GET /api/vacancy/{id}/match/ — получение списка подходящих кандидатов (посредством вызова Matching Service).

Входной точкой служит интерфейс HR-специалиста, а также внутренний вызов из Matching Service.

3. Matching Service

Координирует сопоставление между вакансиями и резюме. Эндпойнты:

1. POST /api/match/ — инициирует анализ соответствия по заданной вакансии,
2. GET /api/match/results/{vacancy\_id}/ — получение ранжированного списка кандидатов,
3. POST /api/feedback/ — запись пользовательской оценки сопоставления (для последующего обучения модели).

Matching Service активно взаимодействует с Resume Service (для получения профилей кандидатов) и Vacancy Service (для извлечения требований).

4. MLAnalyzer

Представляет собой обособленный ML-модуль, работающий в контейнере и обслуживающий запросы на анализ и векторизацию текстов. Эндпойнты:

1. POST /api/analyze/ — асинхронная отправка текста резюме или вакансии,
2. GET /api/analyze/{job\_id}/result — получение результата анализа.

MLAnalyzer вызывает модели на базе BERT и обрабатывает текстовые данные в формате JSON, возвращая эмбеддинги и интерпретации.

5. User Service (часть Auth / Identity Layer)

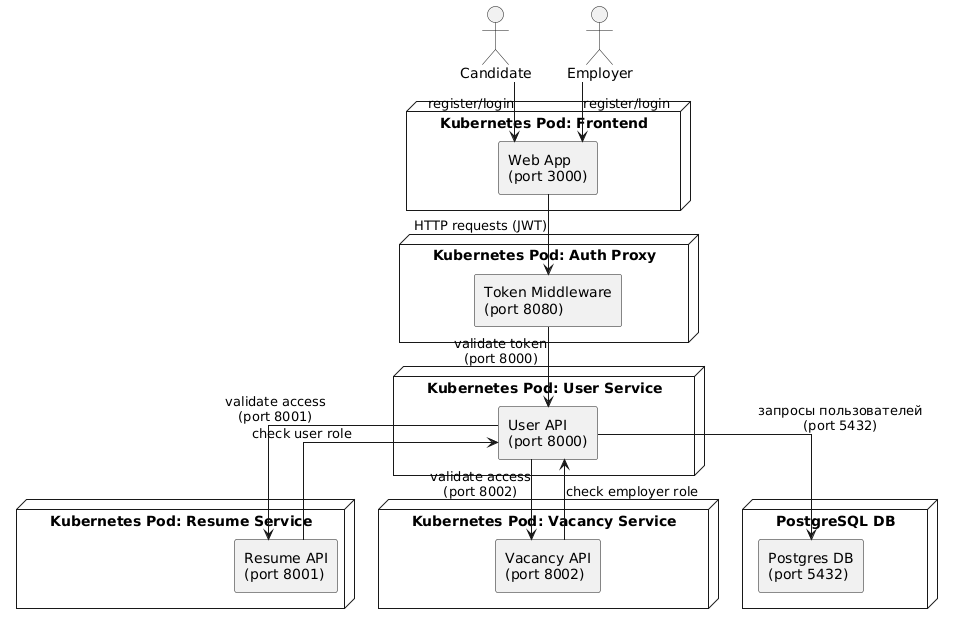
Отвечает за регистрацию, аутентификацию и авторизацию пользователей. Эндпойнты:

1. POST /api/user/register/ — регистрация пользователя,
2. POST /api/user/login/ — аутентификация,
3. GET /api/user/profile/ — получение данных текущего пользователя,
4. POST /api/user/role-switch/ — изменение роли (например, с кандидата на HR).

Этот сервис обслуживает и кандидатов, и работодателей, обеспечивая управление правами доступа к другим сервисам.

Принципы взаимодействия и защита

Все межсервисные вызовы защищены токенами аутентификации (JWT или OAuth 2.0), обеспечивая авторизованный доступ. Структура API соответствует стилю RESTful, а формат JSON позволяет эффективно сериализовать данные. Для интеграции между микросервисами также предусмотрена регистрация с помощью service discovery (в случае масштабирования через Kubernetes DNS).



## 4.2 Внутренние компоненты микросервисов: модели, контроллеры, хранилище и ML-интеграция

Второй уровень конкретизации архитектуры включает детализацию внутренней структуры микросервисов, их компонентов и связей. На этом уровне фокус переносится с внешних API-интерфейсов на внутреннюю реализацию: классы моделей, контроллеры, обработку бизнес-логики, реализацию репозиториев, взаимодействие с ML-компонентами и хранилищем данных. Такой уровень представления необходим для корректного применения принципов предметно-ориентированного проектирования (DDD), соблюдения SOLID-принципов и обеспечения устойчивости и расширяемости кода.

1. Resume Service

Компоненты:

1. Модель Resume: содержит поля education, experience, skills, bio, связанные с пользователем через ForeignKey.
2. Модель Skill: справочник soft/hard skills (многие-ко-многим).
3. Контроллер ResumeViewSet: реализует CRUD-операции над резюме, валидацию, связывание с пользователем.
4. Сервисный слой ResumeParserService: выполняет предварительную обработку текста (удаление HTML, токенизация).
5. Интеграция с ML: передача нормализованного текста в MLAnalyzer через REST (POST /analyze), при необходимости с асинхронным очередным выполнением (Celery + Redis).
6. Репозиторий: использование Django ORM для сохранения в PostgreSQL, оптимизация через select\_related и prefetch\_related.

Особенности: внутренняя логика разделена на «сервисный слой» и «слой контроллеров», соблюдая SRP и DDD (слой обработки находится вне контроллера).

2. Vacancy Service

Компоненты:

1. Модель Vacancy: поля title, requirements, location, salary\_range, связь с Employer.
2. Модель Employer: реквизиты, ИНН, контактные лица.
3. Контроллер VacancyViewSet: реализация API для вакансий, валидация структуры требований.
4. Сервисный слой VacancyRequirementAnalyzer: разбор требований по ключевым словам, извлечение значений для сопоставления.
5. Репозиторий: реализован через ORM, в некоторых случаях — с использованием RawSQL или Q()-выражений для гибкой фильтрации.

Особенности: хранимая процедура ранжирования может быть реализована на уровне PostgreSQL (например, с использованием ts\_rank для полнотекстового поиска).

3. Matching Service

Компоненты:

1. Класс MatchEngine: координирует сопоставление между вектором вакансии и векторами резюме.
2. Методы calculate\_similarity(), rank\_candidates(): реализация метрик (cosine similarity, soft alignment).
3. Слой интеграции ResumeFetcher, VacancyFetcher: извлекает необходимые данные по REST-запросам.
4. ML-интеграция: либо обращение к BERT через MLAnalyzer, либо обращение к кэшированным векторам (в Redis или Postgres).
5. Логирование результатов и обратной связи: сбор обратной связи от пользователя (например, выбран/не выбран кандидат) для дообучения модели.

Особенности: сервис может использовать joblib/scikit-learn-сериализованные модели или TorchServe/TFServing для онлайн-инференса.

4. MLAnalyzer

Компоненты:

1. Модель TextAnalyzer: реализует вызов модели трансформера (например, bert-base-uncased) через transformers API.
2. Метод analyze\_text(): токенизация, векторизация, агрегация CLS-токена.
3. Контроллер AnalyzeAPI: принимает POST-запрос, валидирует вход, выполняет обработку, возвращает JSON с векторами и ключевыми токенами.
4. Асинхронная очередь (опционально): при увеличении нагрузки анализ может переноситься в асинхронный воркер через Celery или FastAPI BackgroundTasks.
5. Особенности: хранение векторных представлений может производиться в Redis (в виде resume:{id}:vector) или в Postgres как бинарное поле (например, через BYTEA или array<float>).

5. User/Auth Service

Компоненты:

1. Модель User: email, пароль (хеш), роль (candidate, employer, admin), статус.
2. Механизм авторизации: JWT (используется библиотека djangorestframework-simplejwt).
3. Контроллеры AuthViewSet: login, register, token-refresh.
4. Permission-классы: разграничение доступа к функционалу в зависимости от роли пользователя.
5. Особенности: структура авторизации унифицирована и используется во всех сервисах через middleware либо прокси (API Gateway).