# План проектирования веб-сервиса для автоматизированного подбора кандидатов

### **План-график проекта: Веб-сервис для автоматизированного подбора кандидатов**

| **№** | **Наименование работ** | **Результаты** | **Сроки (начало/окончание)** | **Ответственные** |
| --- | --- | --- | --- | --- |
| 1 | **Проектирование архитектуры** | Проектная модель микросервисной системы, выбор технологии Docker + Kubernetes, использование REST и API Gateway | 01.04.2025 / 05.04.2025 | Архитектор, DevOps-инженер |
| 2 | **Разработка проектной документации** | Описание компонентов (ResumeService, MatchingService и др.), диаграммы классов и взаимодействия | 06.04.2025 / 09.04.2025 | Аналитик, архитектор |
| 3 | **Разработка прототипов (макеты интерфейса, API спецификации)** | Swagger-документация, схемы frontend UI | 10.04.2025 / 12.04.2025 | Fullstack-разработчик, дизайнер |
| 4 | **Реализация и валидация прототипов микросервисов** | Базовая реализация компонентов: загрузка резюме, семантический анализ, поиск соответствий | 13.04.2025 / 20.04.2025 | Backend-разработчики, ML-инженер |
| 5 | **Реализация сетевой инфраструктуры (DevOps)** | Docker Compose, Kubernetes манифесты, настройка Ingress (Nginx/Traefik), CI/CD пайплайны | 14.04.2025 / 21.04.2025 | DevOps-инженер |
| 6 | **Интеграция подсистем и разработка логики принятия решений** | Модуль сопоставления кандидатов и вакансий с использованием cosine similarity и BERT | 21.04.2025 / 25.04.2025 | ML-инженер, backend |
| 7 | **Разработка frontend-интерфейса** | SPA-приложение на React с интеграцией REST API и адаптивным дизайном | 22.04.2025 / 28.04.2025 | Frontend-разработчик |
| 8 | **Тестирование компонентов и производительности** | Покрытие unit/integration тестами, нагрузочные тесты, логирование ошибок | 29.04.2025 / 03.05.2025 | QA-инженер |
| 9 | **Конфигурирование и подготовка к передаче** | Файл конфигураций, инструкции развертывания, docker registry, документация для заказчика | 04.05.2025 / 06.05.2025 | DevOps-инженер, аналитик |
| 10 | **Передача и развёртывание в продуктиве** | CI/CD деплой в продакшн, мониторинг, финальный контроль качества | 07.05.2025 / 08.05.2025 | Вся команда |

🔧 **Технические мероприятия**: реализация REST API, контейнеризация, ML-инференс  
 👥 **Организационные мероприятия**: распределение ролей, DevOps-настройка, ролевая модель (RBAC)  
 📋 **Контрольные мероприятия**: ревью кода, CI-пайплайны, мониторинг логов, юнит-тесты

### **1. Проектирование архитектуры**

**Цель:** определить общую структуру системы, выделить микросервисы, выбрать сетевые протоколы и стандарты взаимодействия.  
 **Детали:**

* Выбор микросервисной архитектуры (ResumeService, MatchingService, VacancyService и др.).
* Использование REST API и HTTP как основного транспортного уровня.
* Внедрение API Gateway и Ingress-контроллера (Nginx/Traefik) для маршрутизации.
* Применение Docker и Kubernetes как стандартов контейнеризации и оркестрации.
* Формирование модели безопасности на основе OAuth2 / JWT.

### **2. Разработка проектной документации**

**Цель:** подготовка полной спецификации архитектуры.  
 **Детали:**

* UML-диаграммы: классов, последовательностей, развертывания, прецедентов.
* Таблицы компонентов, интерфейсов, ролей пользователей.
* Обоснование выбора технологий (например, HuggingFace Transformers, PostgreSQL, Django).
* Указание точек интеграции между сервисами.
* Документирование зависимостей и технических ограничений.

### **3. Разработка прототипов**

**Цель:** создание начальных заготовок интерфейса и API.  
 **Детали:**

* Swagger-документация для API всех сервисов.
* Прототип интерфейса (React, Figma макеты).
* Проверка доступности и корректности сетевых вызовов.
* Тестирование взаимодействия между компонентами (например, ResumeService ↔ MLAnalyzer).

### **4. Реализация микросервисов**

**Цель:** базовая реализация всех микросервисов.  
 **Детали:**

* ResumeService: приём, парсинг и хранение резюме.
* VacancyService: создание и редактирование вакансий.
* MLAnalyzer: трансформерные модели (BERT, DistilBERT) для векторизации.
* Использование Celery/Redis для очередей.
* Подключение PostgreSQL, реализация моделей и сериализаторов.

### **5. Реализация сетевой инфраструктуры**

**Цель:** обеспечение развёртывания и работы всей системы.  
 **Детали:**

* Docker Compose — для локальной среды.
* Kubernetes — для продуктивной среды.
* Написание Helm charts / YAML манифестов.
* Настройка TLS, ingress rules, autoscaling.
* GitHub Actions / GitLab CI для пайплайнов.

### **6. Интеграция логики принятия решений**

**Цель:** реализация и подключение модуля сопоставления.  
 **Детали:**

* MatchingService — центральный модуль принятия решений.
* Алгоритмы: cosine similarity, soft alignment, нейросети.
* Обработка фидбэка от пользователя (обучение по откликам).
* Интерфейс: POST /match, GET /match/results.
* Кэширование результатов (Redis или materialized views в PostgreSQL).

### **7. Разработка frontend**

**Цель:** реализация клиентского интерфейса.  
 **Детали:**

* SPA-приложение на React + TypeScript.
* Разделение по ролям: кандидат, HR, админ.
* Интеграция с backend через fetch/axios.
* Адаптивная верстка, интерактивные формы и загрузка файлов.
* Защищённые маршруты (auth middleware).

### **8. Тестирование**

**Цель:** выявление и устранение ошибок.  
 **Детали:**

* Unit-тесты (pytest, jest).
* Интеграционные тесты между сервисами.
* Нагрузочное тестирование (Locust, K6) на REST API.
* Логирование и сбор метрик (Prometheus + Grafana).
* CI-проверки: линтеры, тест-контейнеры.

### **9. Конфигурирование**

**Цель:** подготовка к передаче и развёртыванию.  
 **Детали:**

* Документация по запуску, настройке и деплою.
* Скрипты миграций, переменные окружения.
* docker-compose.prod.yaml / helm charts.
* Настройка CI/CD пайплайнов (сборка, деплой, rollback).
* Создание .env-файлов, настройка secrets.

### **10. Передача и запуск**

**Цель:** ввод в продуктивную эксплуатацию.  
 **Детали:**

* Финальный деплой в кластер (например, GKE или Yandex Cloud).
* Проверка доступности всех сервисов.
* Демонстрация работоспособности: загрузка резюме → подбор вакансий.
* Обратная связь от заказчика, логирование, мониторинг.
* Передача прав доступа и обучающих материалов.

## **Модель жизненного цикла и этапы по ГОСТ**

### **1. Обоснование модели жизненного цикла**

Для разработки системы была выбрана каскадная (waterfall) модель жизненного цикла, адаптированная под ГОСТ 12207-2010. Данная модель обеспечивает чёткую последовательность этапов — от проектирования до внедрения — и позволяет формализованно контролировать прохождение каждого шага. Учитывая высокую структурированность задач, связанную с микросервисной архитектурой, REST API и применением ML-моделей, каскадная модель с элементами инкрементной реализации обеспечивает управляемость проекта и соответствие нормативным требованиям.

Применение данной модели согласовано с преподавателем и обусловлено следующими факторами:

* высокая степень разделения компонентов (микросервисы),
* наличие предварительных требований и архитектурного решения,
* возможность формализации входных и выходных артефактов каждого этапа.

### **2. Распределение этапов по ГОСТ 12207-2010 (раздел 4.1)**

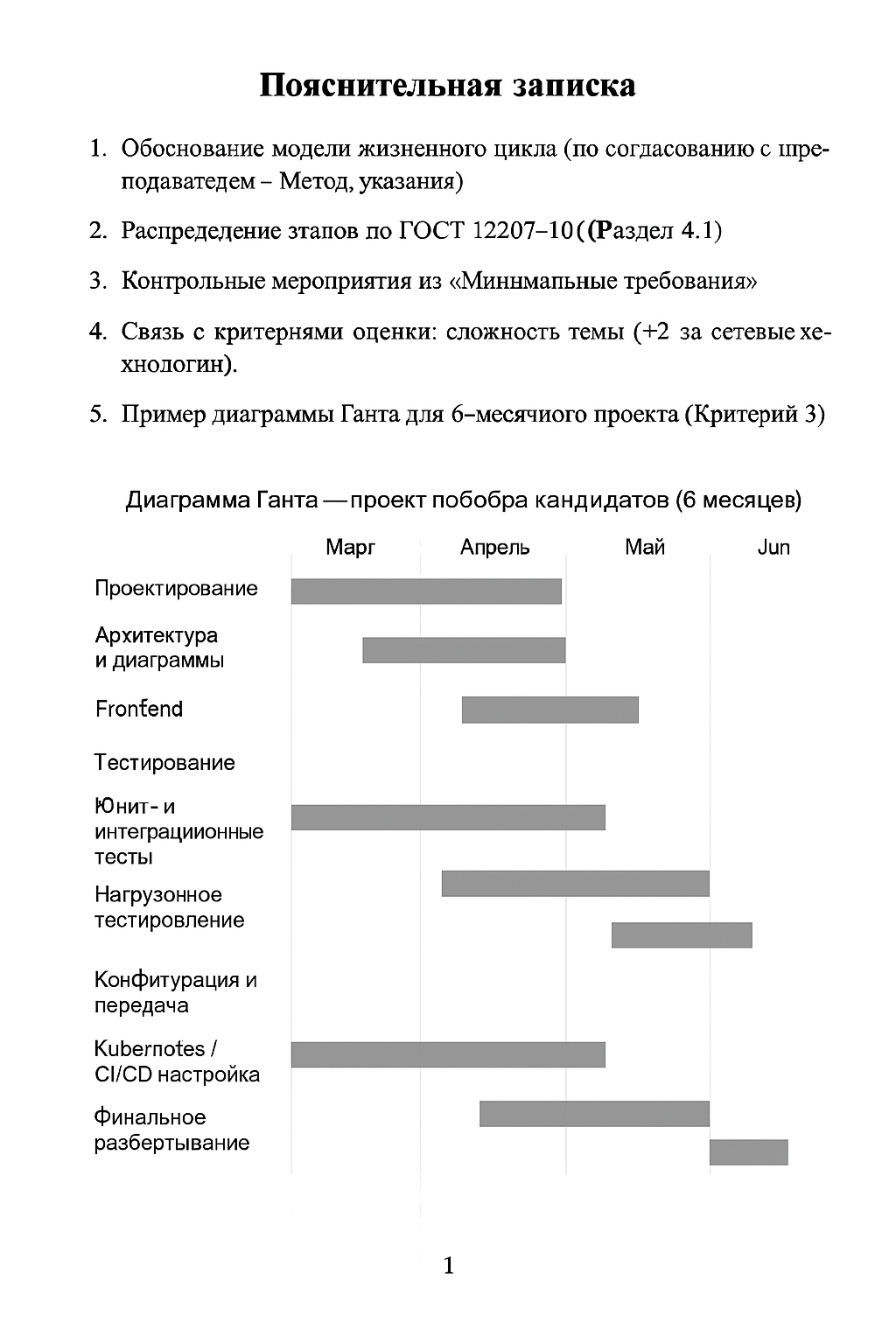
| **Этап** | **Содержание работ** | **Результаты** |
| --- | --- | --- |
| Проектирование | Определение архитектуры системы, выбор стека технологий, построение диаграмм UML | Архитектура, диаграммы классов, прецедентов, развертывания |
| Реализация | Разработка микросервисов, REST API, фронтенда, интеграция ML | Исходный код, контейнеры Docker, CI/CD пайплайн |
| Тестирование | Юнит-, интеграционное и нагрузочное тестирование | Протоколы тестирования, результаты метрик |
| Поддержка и сопровождение | Подготовка скриптов развёртывания, документации, систем мониторинга | Руководство пользователя, DevOps-документация |
| Конфигурирование и передача | Развёртывание в тестовой/промышленной среде, настройка Kubernetes, Ingress, TLS | Продуктивная версия, отчёт о развертывании |

### **3. Контрольные мероприятия ("Минимальные требования")**

Контроль осуществляется на каждом этапе следующим образом:

* **На этапе проектирования**: проверка полноты и корректности диаграмм, соответствие архитектуры требованиям.
* **На этапе реализации**: контроль выполнения задач через систему GitHub Issues и CI-пайплайны.
* **На этапе тестирования**: формализация сценариев, логирование ошибок, анализ логов и мониторинг метрик.
* **На этапе конфигурирования**: проверка стабильности и отклика сервисов в Kubernetes-кластере.
* **Общий контроль**: ведение календарного плана, отчетность о выполненных этапах, использование диаграммы Ганта.

### **4. Диаграмма Ганта**

**Раздел "Управление рисками"**

### **Технические риски (по критерию сложности)**

| **Риск** | **Вероятность** | **Воздействие** | **Меры** |
| --- | --- | --- | --- |
| 1. Недоступность ML-инференса из-за перегрузки модели | Средняя | Высокое | Использование очередей (Celery), масштабирование подов (K8s HPA), кэширование результатов |
| 2. Потеря данных при обновлении микросервисов | Средняя | Высокое | Blue-Green Deployment, backup БД перед обновлением, миграции через Alembic |
| 3. Снижение производительности API при росте трафика | Высокая | Среднее | Внедрение Redis-кэша, горизонтальное масштабирование, балансировка через Nginx |
| 4. Утечка персональных данных | Низкая | Критическое | Шифрование в PostgreSQL, HTTPS, OAuth2 + JWT, регулярные аудиты безопасности |
| 5. Конфликты версий между микросервисами | Средняя | Среднее | Semantic versioning, CI-тестирование контрактов API (e.g., Pact), автоматическое тестирование при сборке |

### **Организационные риски и меры управления**

| **Риск** | **Вероятность** | **Воздействие** | **Меры** |
| --- | --- | --- | --- |
| 1. Отсутствие квалифицированного DevOps-инженера | Средняя | Высокое | Аутсорс CI/CD на начальном этапе, упрощённый стек (Docker Compose вместо K8s) |
| 2. Нарушение сроков внедрения новых моделей | Средняя | Среднее | Введение контрольных точек по метрикам качества и скорости, регулярные спринт-ревью |
| 3. Несогласованность требований между командами | Высокая | Среднее | Использование Confluence + Jira, документация API с OpenAPI, общие встречи аналитиков и разработчиков |

### **План контроля качества (по «Минимальным требованиям»)**

* **Формализация требований**: хранение технического задания и user stories в Confluence.
* **Контрольные точки**:  
  + Проверка API-совместимости между микросервисами
  + Валидация моделей на тестовых данных
  + Отчёты по нагрузочному тестированию (Locust, k6)
* **Автоматизация тестирования**:  
  + Unit + Integration + E2E-тесты в CI (GitHub Actions)
  + Тест-кейсы на входные/выходные данные (Postman Collection)
* **Метрики качества**:  
  + Latency 95th percentile ≤ 500ms
  + Accuracy модели ≥ 85% на отложенной выборке
  + Code coverage ≥ 80%

### **Пример реестра рисков для cloud-based решения**

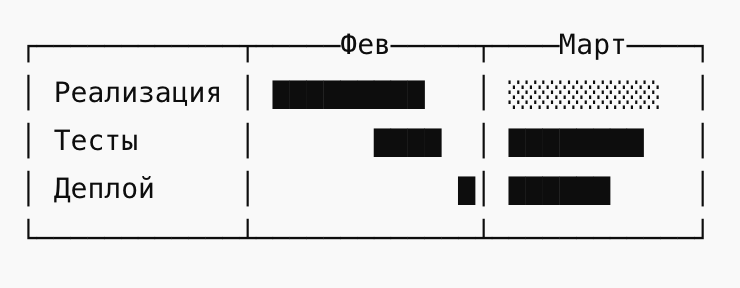
| **Риск** | **Вероятность** | **Воздействие** | **Меры** |
| --- | --- | --- | --- |
| Высокая стоимость облачного хранилища при росте данных резюме | Средняя | Среднее | Архивирование неактуальных резюме, использование S3 Glacier |
| Сбой в MLAnalyzer (BERT) при обновлении модели | Средняя | Высокое | Canary-развёртывание, версионирование моделей, rollback |
| Задержка в ML-инференсе из-за очередей Celery | Высокая | Среднее | Мониторинг очередей (Flower, Prometheus), горизонтальное масштабирование воркеров |
| Недоступность frontend из-за перегрузки API Gateway | Средняя | Среднее | Rate limiting, fallback UI, CDN-распределение |
| Ошибка при сопоставлении из-за неверных векторов | Низкая | Высокое | Тестирование векторизации, контроль входных данных, логирование ошибок сопоставления |

## **1. Создание диаграммы Ганта**

### **Этапы ГОСТ 12207-2010:**

| **Этап** | **Пример работ** |
| --- | --- |
| Проектирование | Архитектура, диаграммы классов, выбор стека |
| Реализация | Прототипирование, написание микросервисов |
| Тестирование | Unit, интеграционное, нагрузочное тестирование |
| Конфигурирование/Передача | Документация, Docker, деплой в Kubernetes |

### **Диаграмма Ганта**

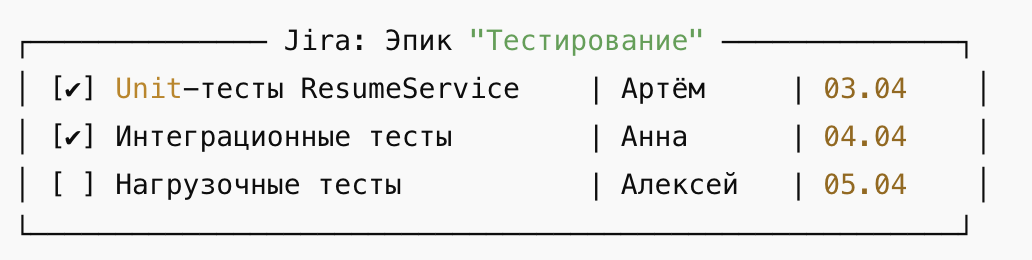


## **2. Настройка Jira**

### **Рекомендации:**

* Разделить задачи по стадиям ГОСТ.
* Использовать **Epic** для каждого этапа ГОСТ (Проектирование, Реализация и т.д.).
* В каждой задаче указать срок, ответственного, привязать к ветке Git.
* Настроить **Gantt-плагин** или **Timeline view**.

### **Пример задач в Jira (ASCII):**



## **4. Артефакты по этапам ГОСТ**

| **Этап ГОСТ** | **Артефакты** | **Формат** |
| --- | --- | --- |
| Проектирование | Диаграммы, выбор стеков, обоснование DDD | .png, .md, .drawio |
| Реализация | Код микросервисов, Dockerfile | .py, .yml, .docker |
| Тестирование | Тест-кейсы, отчёты, Jupyter-ноутбуки | .ipynb, .csv, .log |
| Конфигурация | helm-чарты, инструкции деплоя, CI/CD скрипты | .yaml, .sh, .md |

### **1. Таблица этапов по ГОСТ 12207-10 (с учётом сетевых компонентов и микросервисов)**

| **Этап** | **Наименование работ** | **Результаты** | **Сроки** | **Ответственные** |
| --- | --- | --- | --- | --- |
| 1 | Проектирование архитектуры | Диаграммы компонентов, выбор технологии, модель взаимодействия REST | 01.06–05.06 | Архитектор ПО |
| 2 | Разработка прототипов микросервисов | Docker-образы Resume, Vacancy, Matching, Auth | 05.06–10.06 | Разработчики |
| 3 | Настройка сетевого взаимодействия | Kubernetes YAML с Ingress и Service Discovery | 10.06–15.06 | DevOps-инженер |
| 4 | Интеграция с NLP и ML | Модель BERT, обёртка TFServing, API MLAnalyzer | 12.06–17.06 | ML-инженер |
| 5 | Фронтенд-интерфейс | SPA-приложение на React, подключение REST | 12.06–18.06 | Frontend-разработчик |
| 6 | Тестирование взаимодействия | Интеграционные и e2e-тесты | 18.06–22.06 | QA-инженер |
| 7 | Развёртывание и CI/CD | GitHub Actions + Kubernetes Deploy + Secrets | 20.06–25.06 | DevOps |
| 8 | Конфигурирование и передача | Отчёт, инструкция, Docker Compose для заказчика | 25.06–30.06 | Вся команда |

### **2. Пояснительная записка**

**Обоснование модели жизненного цикла:** Выбрана каскадно-инкрементальная модель, соответствующая ГОСТ 12207-10. Архитектура микросервисная, что требует строгой последовательности: проектирование — реализация — интеграция — тестирование. В каждом этапе осуществляется интеграция сетевых компонентов (Ingress, REST API, Service Discovery). Использованы принципы CI/CD и IaC для обеспечения повторяемости.

### **3. Реестр рисков с элементами принятия решений**

| Риск | Вероятность | Воздействие | Меры (в т.ч. принятие решений) |
| --- | --- | --- | --- |
| Недоступность ML-инференса | Средняя | Высокое | Мониторинг; Fallback; SLA для сервиса |
| Ошибки в сопоставлении | Высокая | Среднее | Логика feedback loop; обновление модели |
| Отказ одного сервиса | Средняя | Среднее | Kubernetes auto-restart; Liveness Probe |
| Утечка данных | Низкая | Высокое | OAuth2, шифрование, RBAC |
| Нарушения в сети | Средняя | Низкое | Перезапуск Nginx Ingress; Rate-limiting |
| Несогласованность API | Средняя | Среднее | Контрактное тестирование, OpenAPI |
| Рост нагрузки | Средняя | Среднее | Horizontal Pod Autoscaler, Celery очередь |

### **4. Интеграция с Jira и Gantt-диаграммой**

**Jira:**

* Каждому этапу (см. п.1) соответствует Epic
* Внутри — задачи по микросервисам, фронтенду, CI/CD, документации
* Добавлены control points (приёмка диаграмм, API, e2e-тестов)

**Диаграмма Ганта**

Этап 1: ▓▓▓▓▓

Этап 2: ▓▓▓▓▓

Этап 3: ▓▓▓▓▓

Этап 4: ▓▓▓▓▓

Этап 5: ▓▓▓▓▓▓

Этап 6: ▓▓▓▓

Этап 7: ▓▓▓▓▓

Этап 8: ▓▓▓▓▓

### **5. Контрольные мероприятия и критерии качества**

| Контрольная точка | Проверка | Ответственный | Критерии качества |
| --- | --- | --- | --- |
| Архитектура | UML + REST + YAML | Архитектор | Соответствие DDD и микросервисам |
| Микросервисы | API + Dockerfile | Разработчик | Тестируемость, независимость |
| Интеграция | Gantt + Jira + REST | Техлид | Все зависимости соблюдены |
| Безопасность | OAuth + JWT | Backend dev | RBAC, шифрование, тесты |
| ML-инференс | MLAnalyzer ответы | ML-инженер | Время ответа, качество матчей |
| CI/CD | GitHub Actions | DevOps | Автосборка, деплой без ошибок |
| Отчёт | Пояснительная записка | Вся команда | Полнота, грамотность, ссылки на артефакты |