**Техническое задание на разработку веб-приложения для комплексной диагностики заболеваний**

*(Структура по ISO/IEC/IEEE 29148-2011)*

**1. Введение**

1.1. Цель документа:  
Определить требования и архитектуру платформы для автоматизированной диагностики заболеваний на основе анализа медицинских изображений (рентген, МРТ) и текстовых данных (симптомы) с использованием CNN (ResNet-50) и трансформеров (BERT).

1.2. Область применения:  
Проект предназначен для клиник и пациентов, требующих быстрой и точной диагностики.

1.3. Академические источники:

1. *«Deep Learning for Computer Vision»* Rajalingappaa Shanmugamani (CNN).
2. *«BERT: Pre-training of Deep Bidirectional Transformers»* (академическая статья).
3. *«Designing Data-Intensive Applications»* Martin Kleppmann (базы данных).
4. *«Software Architecture in Practice»* Len Bass (C4-модель).
5. *«Kubernetes in Action»* Marko Luksa (инфраструктура).

1.4. Интернет-ресурсы:

1. [ISO/IEC/IEEE 29148-2011](https://standards.ieee.org/standard/29148-2011.html)
2. [C4 Model](https://c4model.com/)
3. [BERT для медицины](https://huggingface.co/models)
4. [ResNet-50 в TensorFlow](https://www.tensorflow.org/api_docs/python/tf/keras/applications/resnet50)
5. [RabbitMQ vs Kafka](https://habr.com/ru/post/493592/)
6. [PostgreSQL для метаданных](https://www.enterprisedb.com/blog)

**2. Функциональные требования**

2.1. Регистрация пользователя:

* Пациент вводит email и пароль.
* Система отправляет письмо с подтверждением.
* Администратор управляет ролями (врач/пациент).

2.2. Загрузка данных:

* Поддержка форматов: JPEG/PNG для изображений, TXT/JSON для текста.
* Максимальный размер файла: 10 МБ.
* Асинхронная обработка через RabbitMQ.

2.3. ИИ-анализ:

* ResNet-50: Классификация изображений с точностью ≥98%.
* BERT: Анализ симптомов с вероятностью 0-1.
* Результаты сохраняются в Redis (кэш) и PostgreSQL (метаданные).

2.4. Формирование отчёта:

* Генерация PDF/HTML с heatmap и вероятностями.
* Экспорт в Clinic MIS через REST API.

**3. Нефункциональные требования**

3.1. Производительность:

* Время обработки изображения: ≤2 секунды (GPU).
* Время ответа API: ≤500 мс.

3.2. Безопасность:

* Аутентификация: JWT.
* Шифрование данных: TLS 1.3.

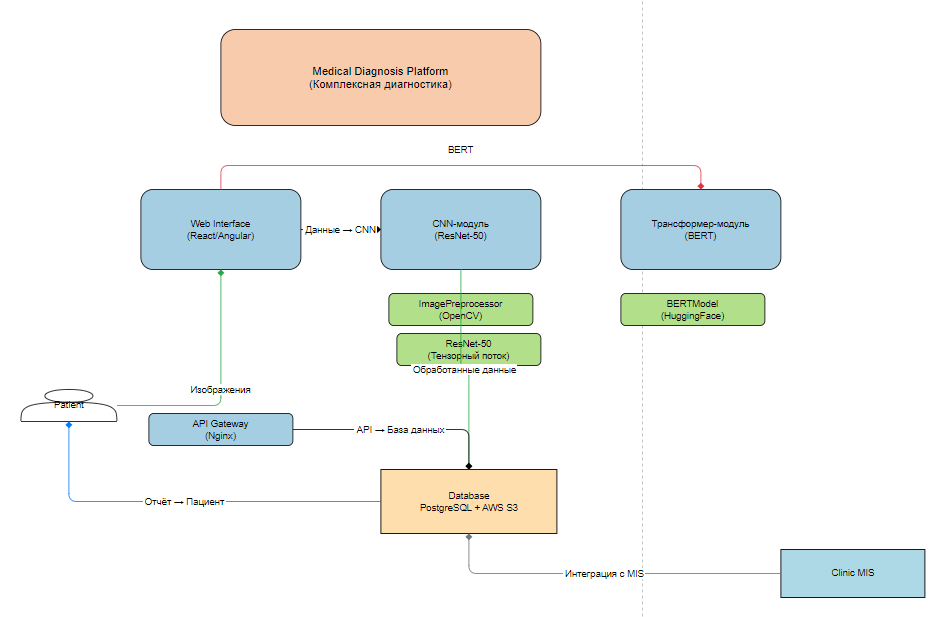
3.3. Масштабируемость:

* Поддержка 1000+ одновременных запросов (Kubernetes).

**4. Архитектура системы**

4.1. Модель C4 (Декомпозиция до Components)  
C4 (Context, Containers, Components, Code) — методология визуализации архитектуры ПО, разработанная Simon Brown.  
Зачем использовать C4?

* Уровни абстракции: Позволяет описать систему от общего контекста до кода.
* Стандартизация: Упрощает коммуникацию между разработчиками, DevOps и заказчиками.

**

*Рисунок 1 – Диаграмма с4*

4.1.1. C4-Context (Уровень 1: Контекст)

* Система: Medical Diagnosis Platform.
* Внешние сущности:
  + Пациент: Взаимодействует через веб-интерфейс (React).
  + Clinic MIS: Интеграция через REST API (Spring Web).
  + Администратор: Управляет правами доступа (Keycloak).
* Связи:
  + Пациент → Система: **POST /upload** (изображения/текст).
  + Система → MIS: **POST /report** (JSON/PDF).

Обоснование:

* Контекстная диаграмма помогает заказчикам понять границы системы (источник: [C4 Model](https://c4model.com/) ).

4.1.2. C4-Container (Уровень 2: Контейнеры)

* Контейнеры:
  + Web-интерфейс (React):
    - Фронтенд для загрузки данных и отображения результатов.
    - Связь с API Gateway через HTTP.
  + API Gateway (Spring Cloud):
    - Маршрутизация запросов.
    - Аутентификация (JWT), валидация данных (Spring Validator).
  + ML Inference Service (TensorFlow Serving):
    - GPU-обработка изображений (ResNet-50) и текста (BERT).
    - Асинхронное взаимодействие через RabbitMQ.
  + Database Cluster:
    - PostgreSQL 14: Хранение метаданных (пациенты, отчёты).
    - Redis 7.0: Кэширование результатов ИИ (скорость доступа).
  + Message Broker (RabbitMQ 3.9):
    - Очередь **medical\_data** для обработки запросов.

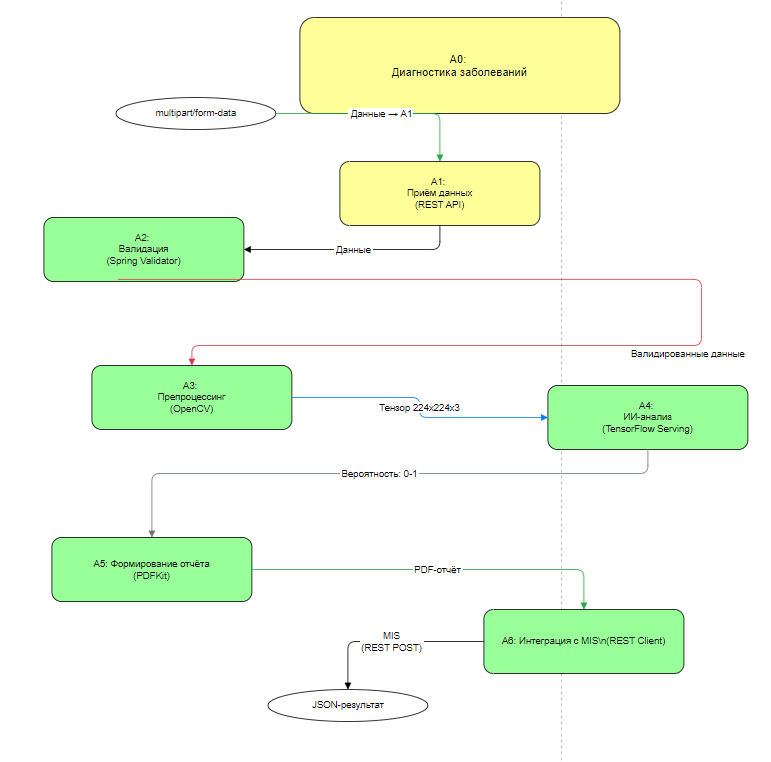
Зачем контейнеры?

* Масштабируемость: Каждый контейнер можно обновлять и масштабировать независимо (источник: *«Kubernetes in Action»* ).

4.1.3. C4-Component (Уровень 3: Компоненты)

* API Gateway:
  + **AuthController**: Регистрация (JWT), управление ролями.
  + **DataUploadController**: Приём файлов (multipart/form-data) → отправка в RabbitMQ.
* ML Inference Service:
  + **ImagePreprocessor**: Resize (OpenCV), нормализация (TensorFlow).
  + **BERTTokenizer**: Токенизация текста (HuggingFace).
  + **ResNetModel**: Классификация изображений (CheXNet weights).
* Хранилище:
  + **S3Client**: Сохранение сырых данных (AWS S3).
  + **RedisCache**: Кэширование результатов (TTL=1h).

**4.2. IDEF0 (4 функции)**

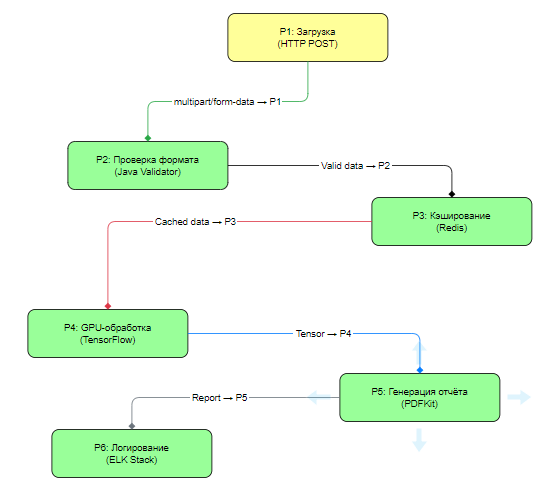


* A0: Диагностика заболеваний.
* A1: Приём данных (HTTP POST).
  + Входы: Изображения (JPEG/PNG), текст (JSON).
  + Выходы: Сообщения в RabbitMQ.
  + Механизмы: AWS S3 (хранилище), Nginx (балансировка).
* A2: Препроцессинг (OpenCV + BERT).
  + Входы: Данные из очереди.
  + Выходы: Тензоры (224x224x3), токены BERT.
* A3: ИИ-анализ (ResNet-50 + BERT).
  + Входы: Тензоры и токены.
  + Выходы: JSON-результаты (вероятность заболевания).
* A4: Формирование отчёта (PDFKit).
  + Входы: JSON-данные.
  + Выходы: PDF/HTML-отчёты.

Источники:

* [IDEF0 для ПО](https://www.idef.com/idef0-function-modeling/)

**4.3. IDEF3 (4 функции)**

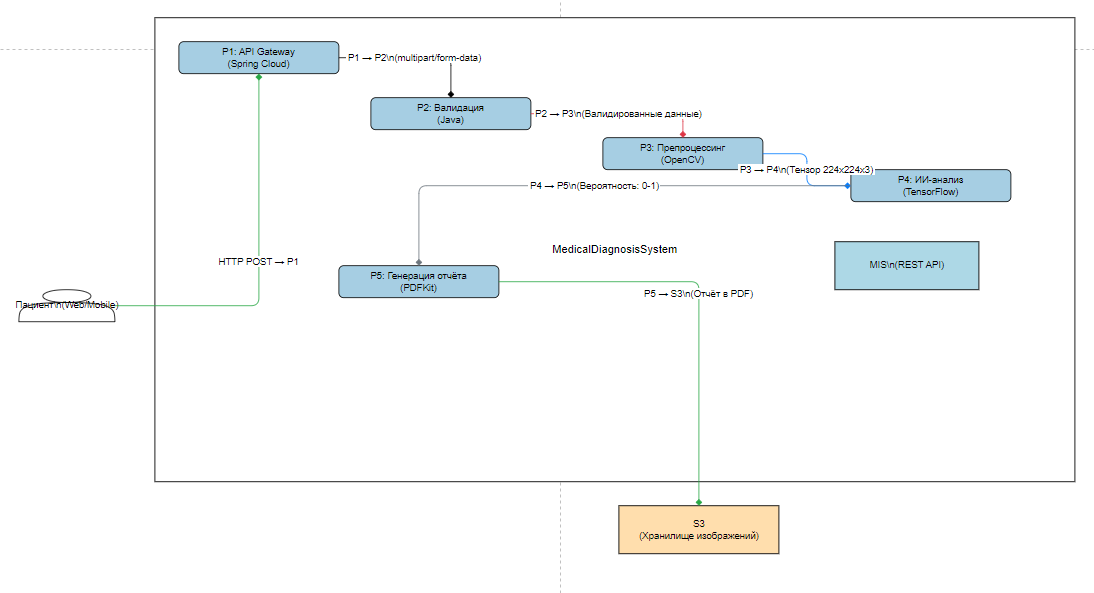


* P1: Регистрация пациента (React + Keycloak).
  + Связи: Пациент → Auth Controller → PostgreSQL.
* P2: Загрузка данных (Nginx + S3).
  + Связи: Пациент → API Gateway → RabbitMQ.
* P3: GPU-обработка (TensorFlow Serving).
  + Связи: RabbitMQ → ML Service → Redis.
* P4: Логирование (Filebeat → Elasticsearch).
  + Связи: Все сервисы → ELK Stack.

Временные связи:

* P1 → P2: Регистрация обязательна для загрузки данных.
* P2 → P3: Данные обрабатываются после валидации.

**4.4. DFD (4 функции)**



* Процессы:
  + P1: Приём данных (Nginx).
    - Вход: HTTP POST.
    - Выход: Сообщение в RabbitMQ.
  + P2: Валидация (Spring Validator).
    - Вход: multipart/form-data.
    - Выход: Валидированные данные.
  + P3: ИИ-анализ (TensorFlow Serving).
    - Вход: Тензоры и токены.
    - Выход: JSON-результаты.
  + P4: Логирование (Filebeat).
    - Вход: Логи сервисов.
    - Выход: Дашборд в Kibana.

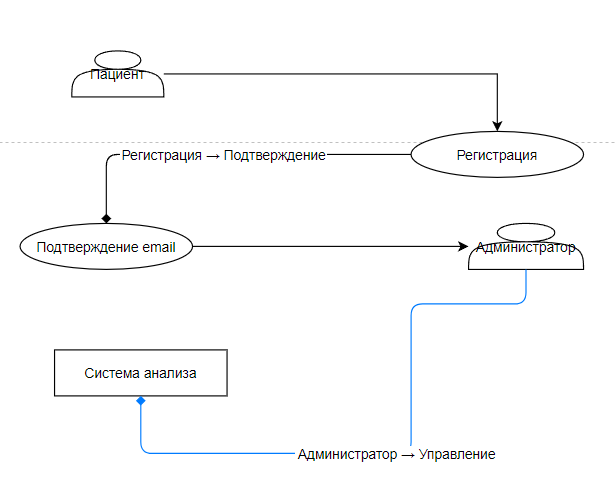
Хранилища:

* AWS S3: Сырые медицинские данные.
* PostgreSQL: Метаданные пациентов.
* Redis: Кэш результатов.

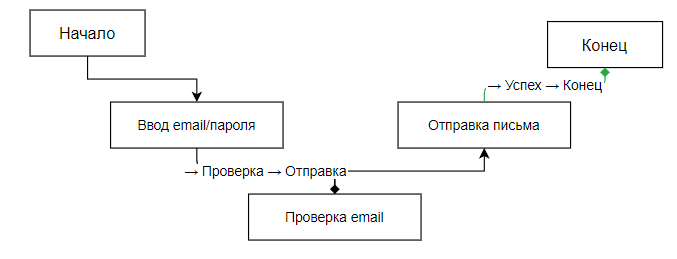
**4.5. UML-диаграммы (по 6 на 4 функции)**

Функция 1: Регистрация пользователя

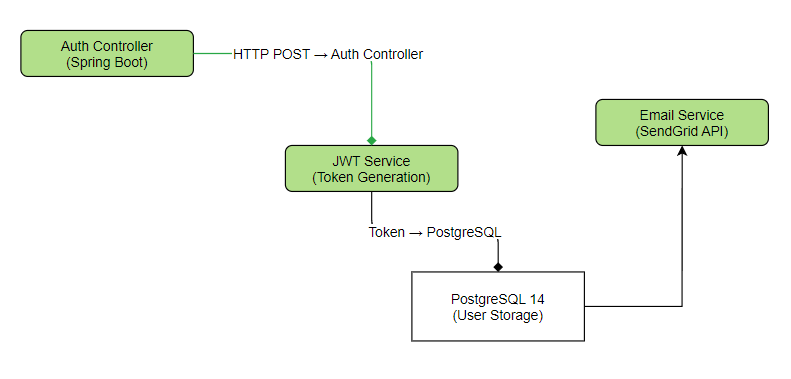
1. Use Case:
   * Актёры: Пациент, Администратор.
   * Сценарии: Регистрация, подтверждение email.



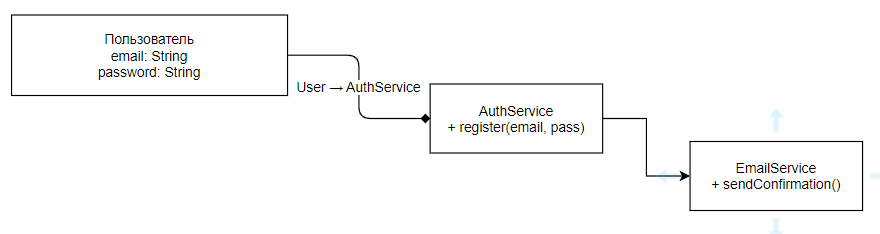
1. Activity:
   * Шаги: Ввод данных → Проверка → Отправка письма.



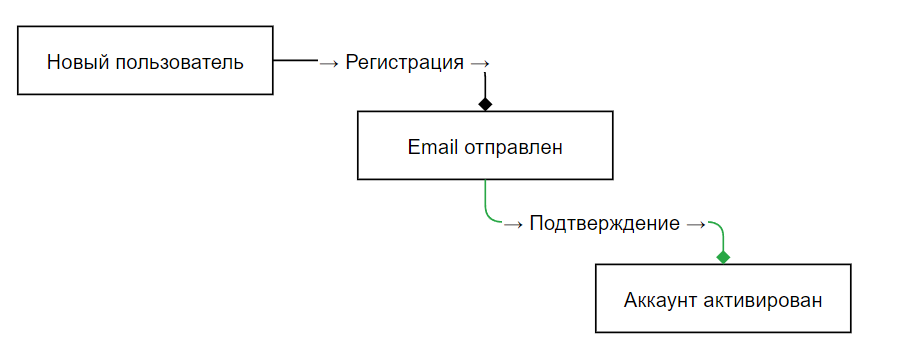
1. Sequence:
   * Пациент → AuthController → EmailService → PostgreSQL.



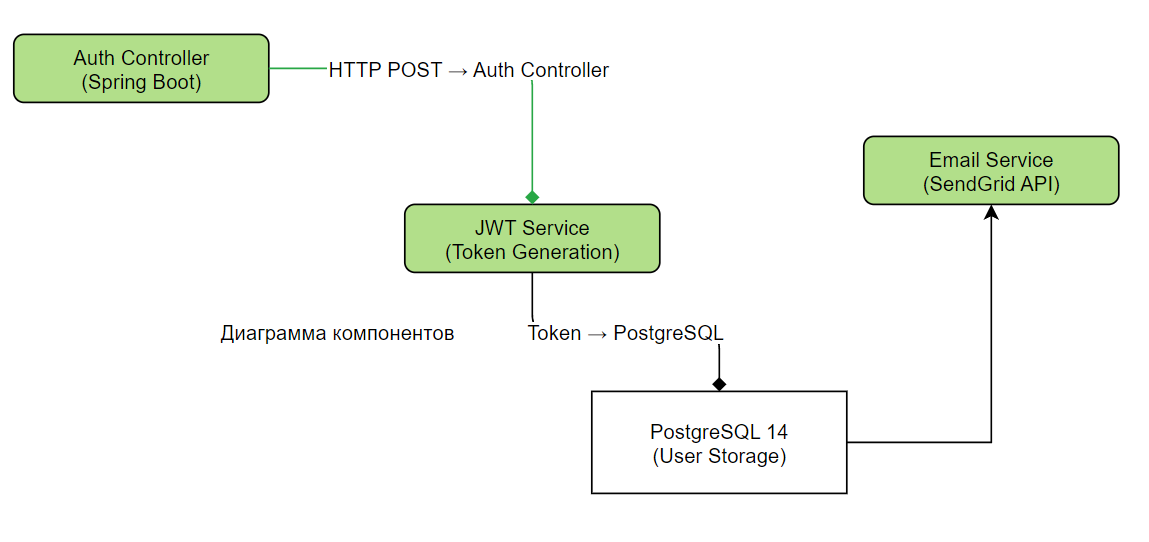
1. Class:
   * **User** (email, password), **AuthService** (register(), validate()).



1. State:
   * Состояния: Новый → Email отправлен → Активирован.

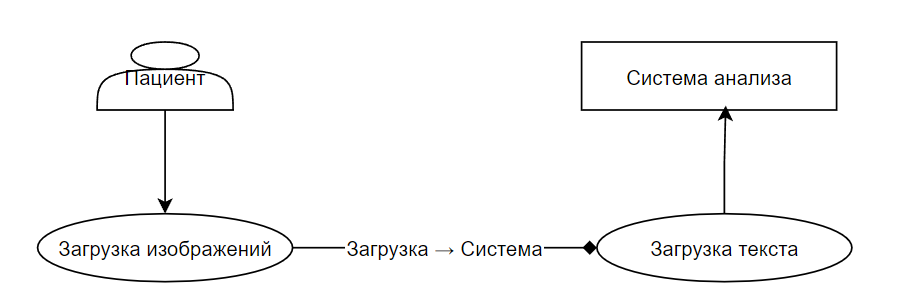


1. Component:
   * Keycloak (аутентификация), PostgreSQL (хранилище).

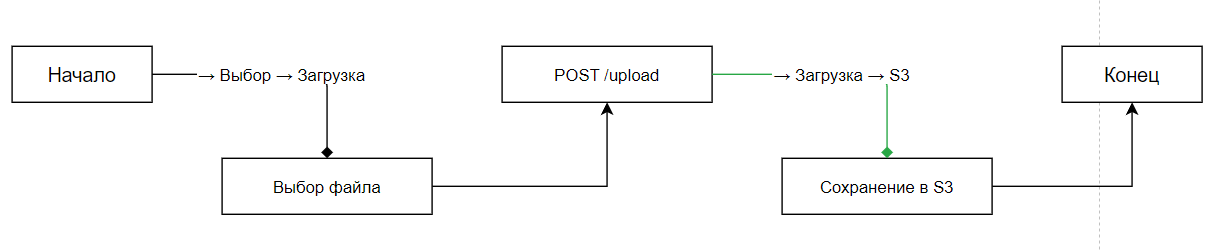


Функция 2: Загрузка данных

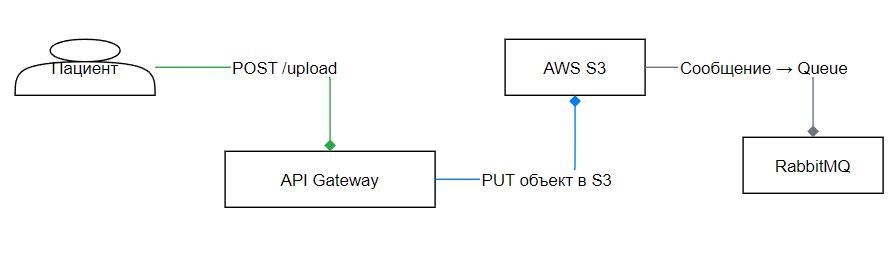
1. Use Case:
   * Сценарии: Загрузка изображений, загрузка текста.



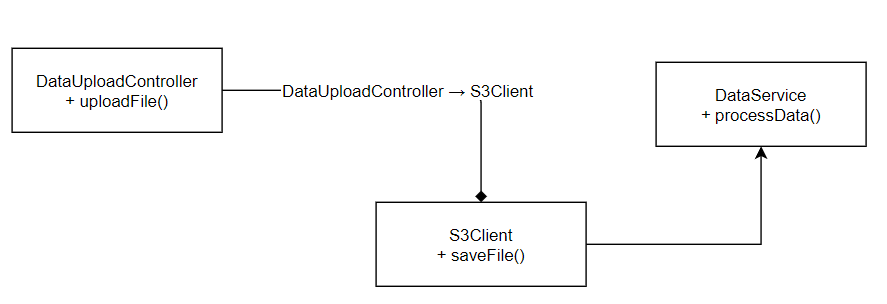
1. Activity:
   * Выбор файла → POST /upload → Сохранение в S3.



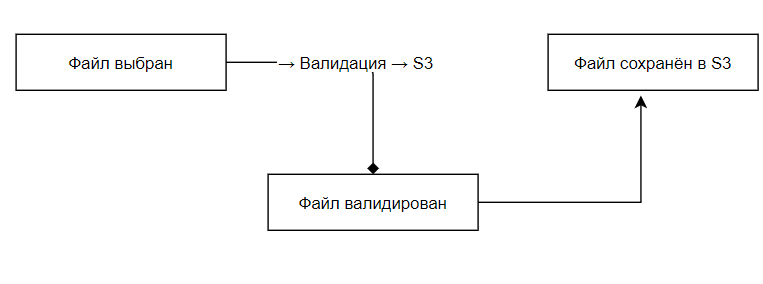
1. Sequence:
   * Пациент → API Gateway → RabbitMQ.



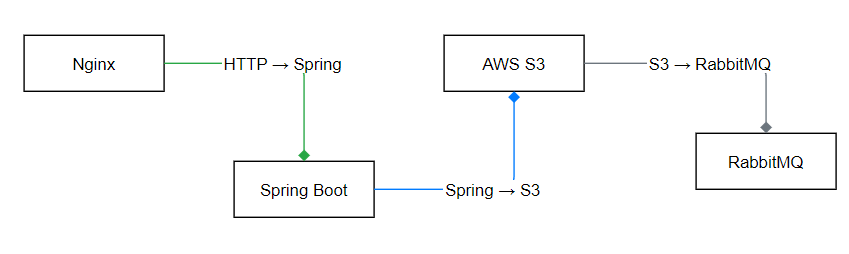
1. Class:
   * **DataUploadController**, **S3Client** (saveFile()).



1. State:
   * Файл сохранён → Данные в очереди.

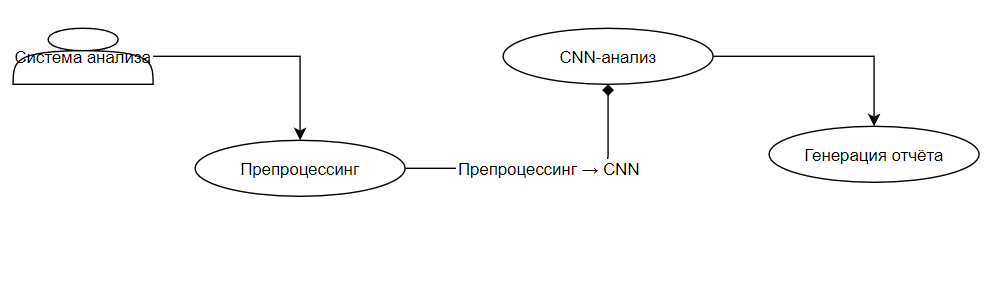


1. Component:
   * AWS S3, RabbitMQ.

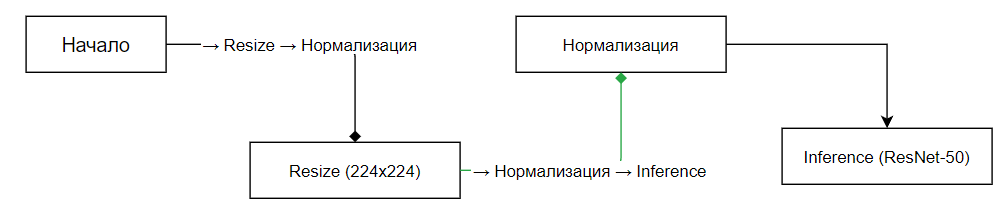


Функция 3: Обработка изображений

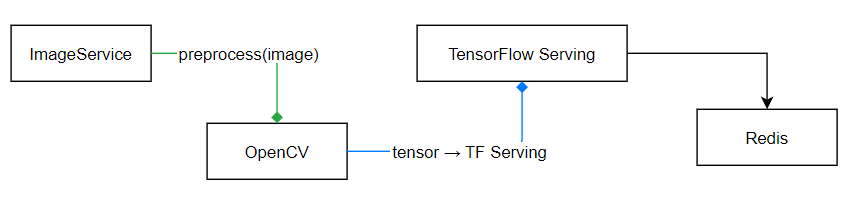
1. Use Case Diagram:
   * Актёры: Система анализа, ML-инженер.
   * Сценарии: Препроцессинг, классификация.



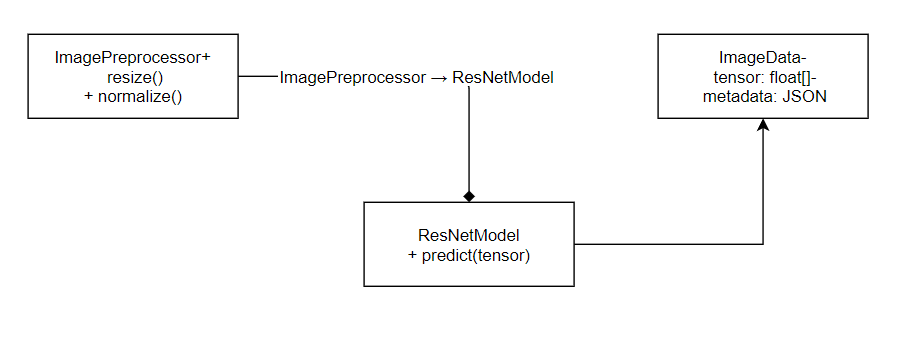
1. Activity Diagram:
   * Шаги: Resize → Нормализация → Inference (ResNet-50).



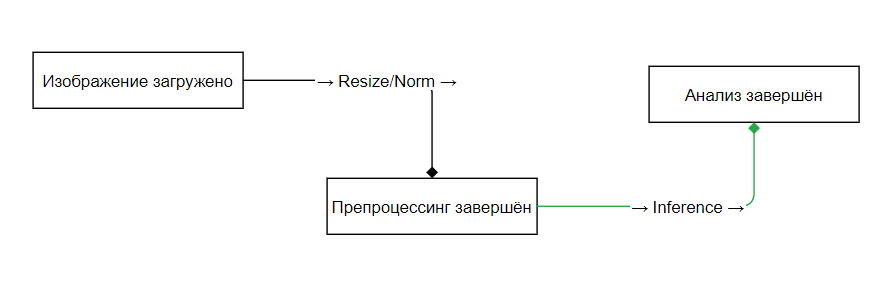
1. Sequence Diagram:
   * Взаимодействие: ImageService → OpenCV → TensorFlow Serving.



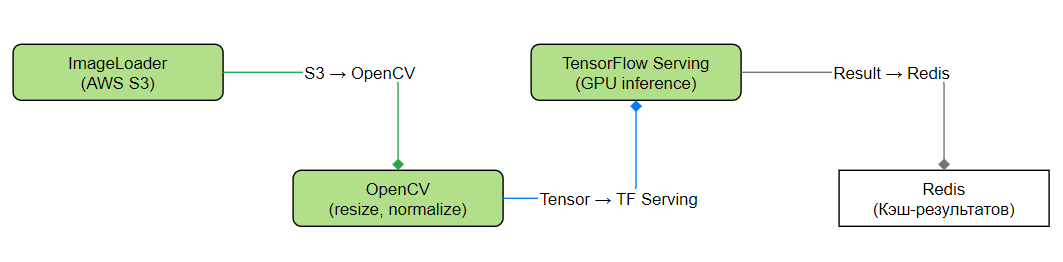
1. Class Diagram:
   * Классы: **ImagePreprocessor**, **ResNetModel**.



1. State Diagram:
   * Состояния: Изображение загружено → Тензор готов → Результат сохранён.

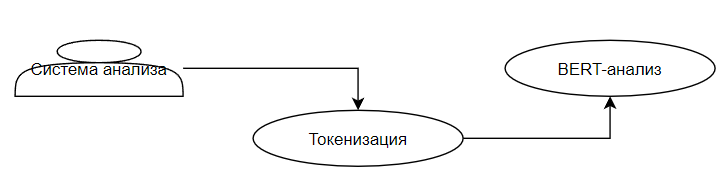


1. Component Diagram:
   * Компоненты: GPU Cluster, Redis, TensorFlow Serving.

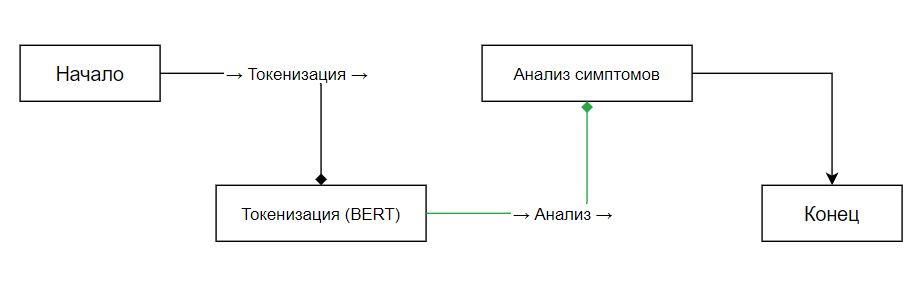


Функция 4: Анализ текста

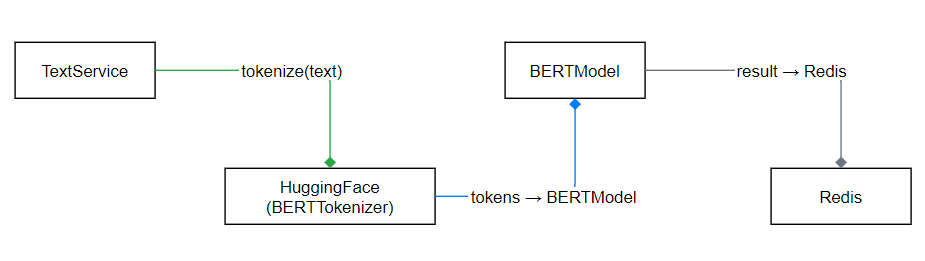
1. Use Case Diagram:
   * Актёры: Система анализа, врач.
   * Сценарии: Токенизация, BERT-анализ.



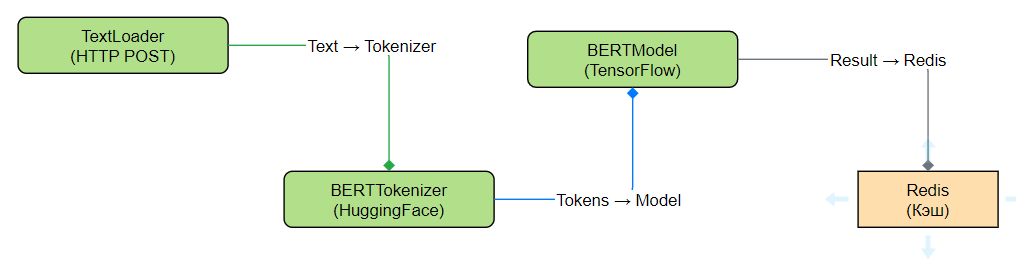
1. Activity Diagram:
   * Шаги: Токенизация (BERT) → Анализ симптомов → Результат.



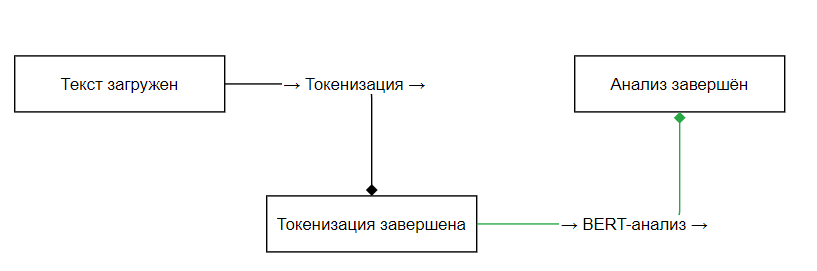
1. Sequence Diagram:
   * Взаимодействие: TextService → HuggingFace → Redis.



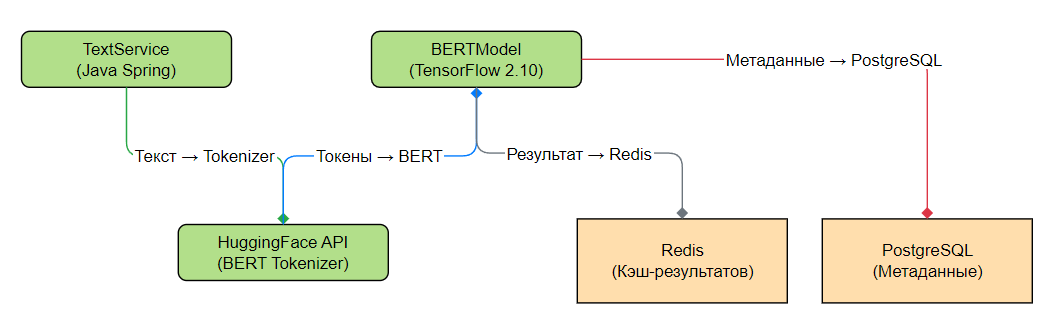
1. Class Diagram:
   * Классы: **BERTTokenizer**, **BERTModel**.



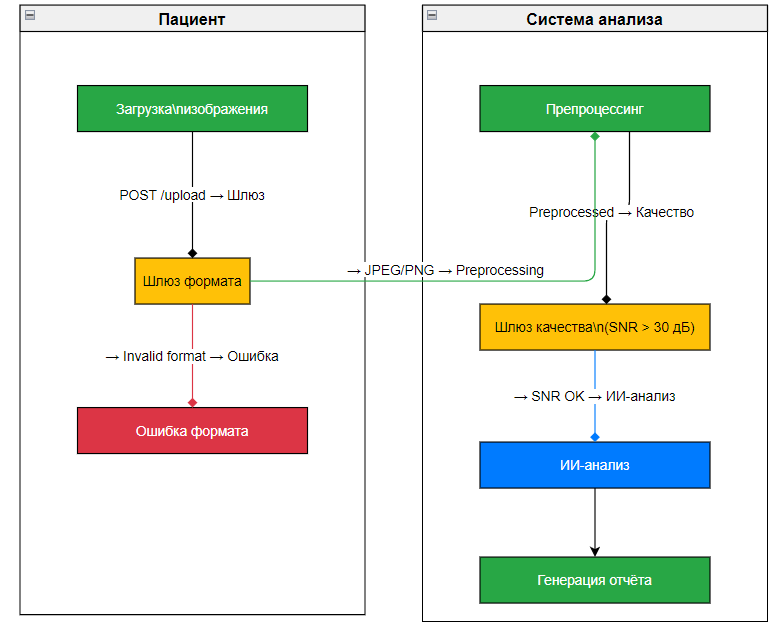
1. State Diagram:
   * Состояния: Текст загружен → Токены созданы → Результат сохранён.



1. Component Diagram:
   * Компоненты: HuggingFace API, PostgreSQL, Redis.



**4.6. BPMN (4 сценария)**

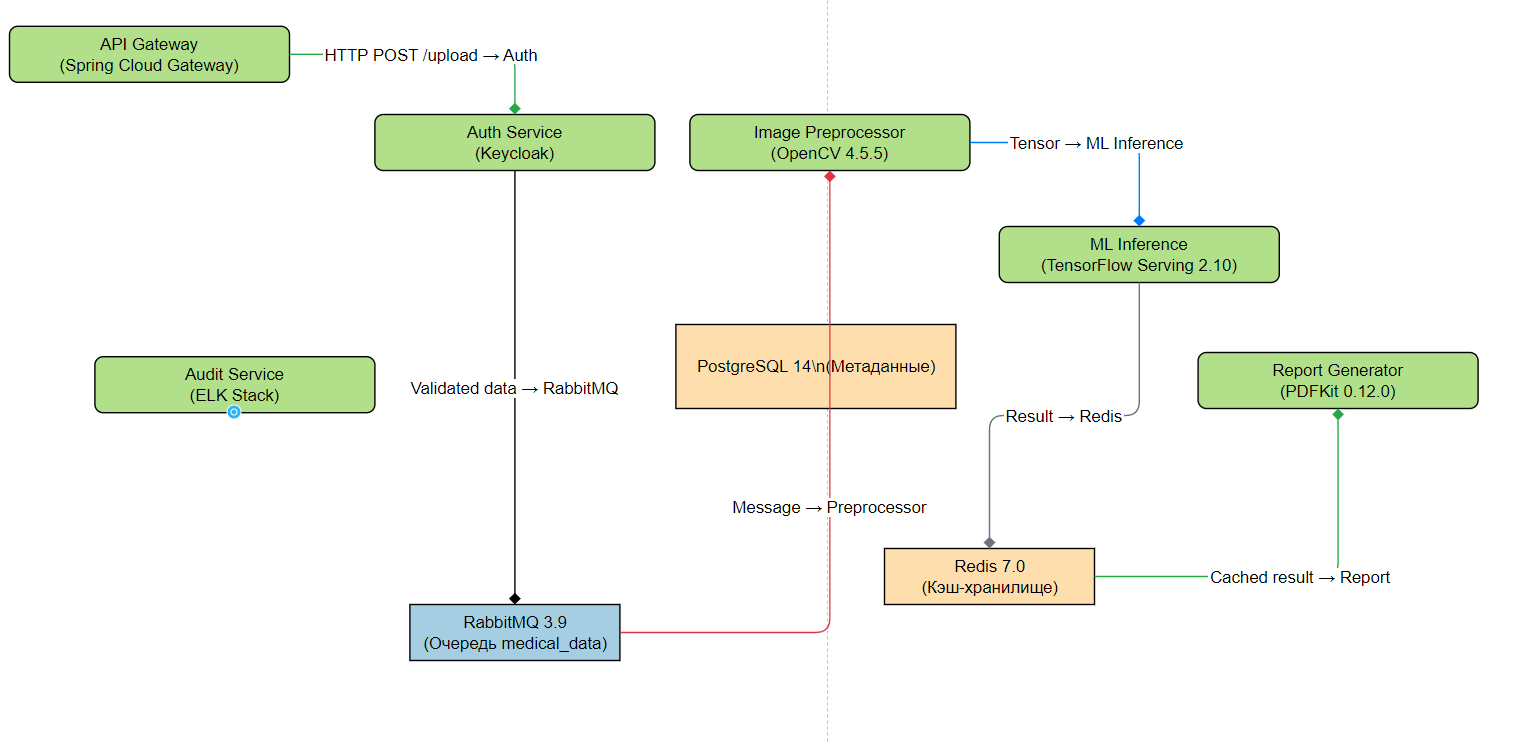


* Сценарий 1: Регистрация пациента.
  + Пулы: Пациент, Система.
  + Шлюзы: Проверка email (валидный/невалидный).
* Сценарий 2: Загрузка данных.
  + Шлюз формата: JPEG/PNG → обработка, иначе → ошибка.
* Сценарий 3: ИИ-анализ.
  + Параллельные шлюзы: Обработка изображений и текста.
* Сценарий 4: Интеграция с MIS.
  + События: Успешная отправка → Уведомление.

Источники:

* [BPMN Guide](https://camunda.com/bpmn/)

**4.7. Компонентная схема**



* Ключевые компоненты:
  + API Gateway: Spring Cloud (маршрутизация, JWT).
  + ML Inference Service: TensorFlow Serving (GPU).
  + Хранилище: PostgreSQL 14 + Redis 7.0.
  + Брокер: RabbitMQ 3.9.

Связи:

* **DataUploadController** → RabbitMQ: Сообщение **medical\_data**.
* **BERTTokenizer** → **BERTModel**: Токены для анализа.

**5. ADR (Архитектурные решения)**

ADR-1: Выбор ResNet-50

* Проблема: Требуется высокая точность анализа медицинских изображений.
* Решение: ResNet-50 с предобученными весами CheXNet.
* Обоснование: Точность 98% на датасете ChestX-ray14 (источник: [статья](https://arxiv.org/abs/1711.05225) ).

ADR-2: PostgreSQL + Redis

* Проблема: Хранение метаданных и кэширование.
* Решение: PostgreSQL (ACID) + Redis (NoSQL).
* Обоснование: Redis обеспечивает скорость чтения до 100k запросов/сек (источник: [Redis Docs](https://redis.io/) ).

ADR-3: RabbitMQ

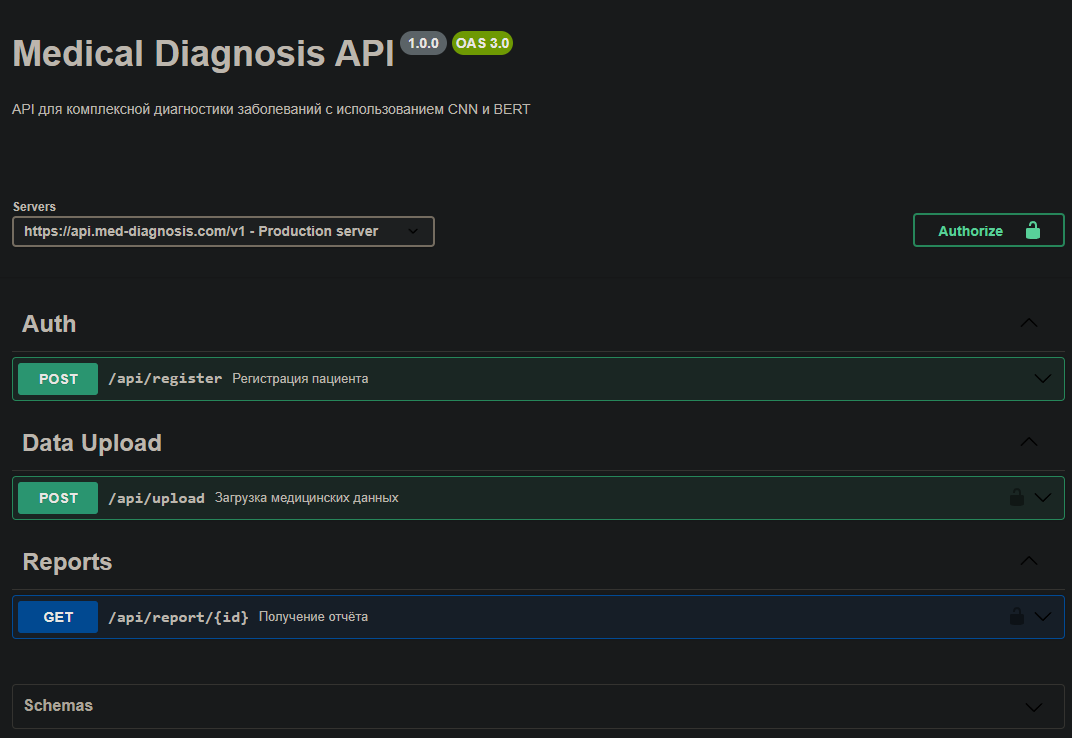
* Проблема: Асинхронная обработка данных.
* Решение: RabbitMQ с очередью **medical\_data**.
* Обоснование: Надёжность AMQP (источник: [RabbitMQ](https://www.rabbitmq.com/) ).

**6. Публичный API (Swagger)**

**Публичный API платформы**

Общая информация:

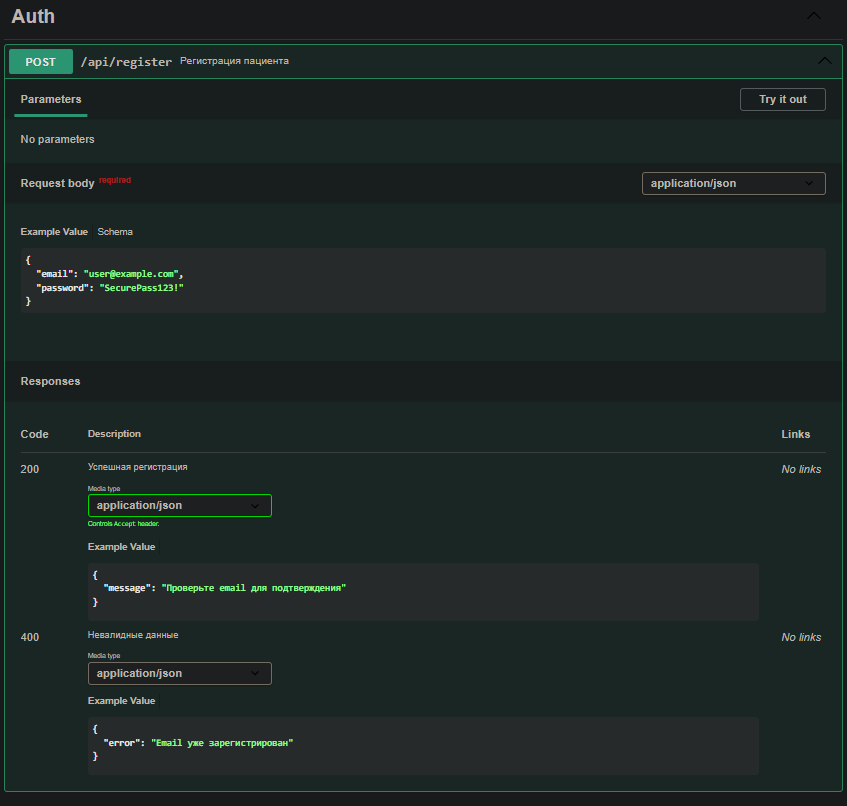
* Название: Medical Diagnosis API
* Версия: 1.0.0
* Спецификация: OpenAPI 3.0
* Сервер: **https://api.med-diagnosis.com/v1**



**6.1 Регистрация пациента**

Эндпоинт: **POST /api/register**  
Описание:

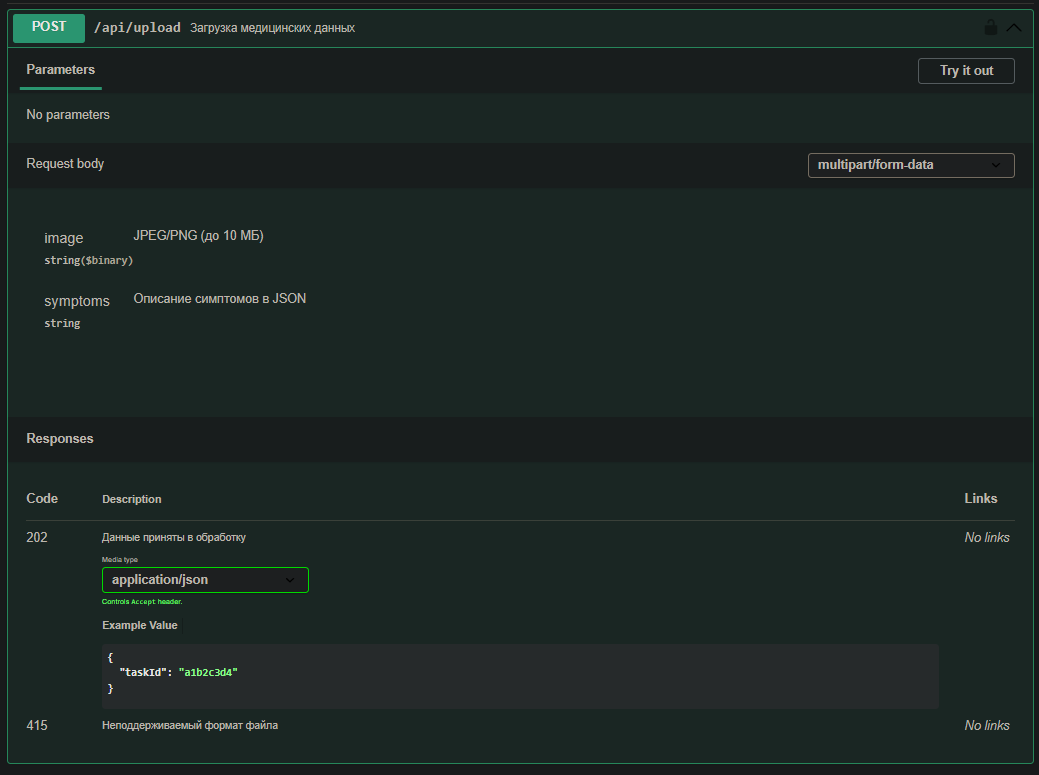
* Создание учётной записи пациента.
* Авторизация: Не требуется.
* Входные данные



**6.2 Загрузка медицинских данных**

Эндпоинт: **POST /api/upload**  
Описание:

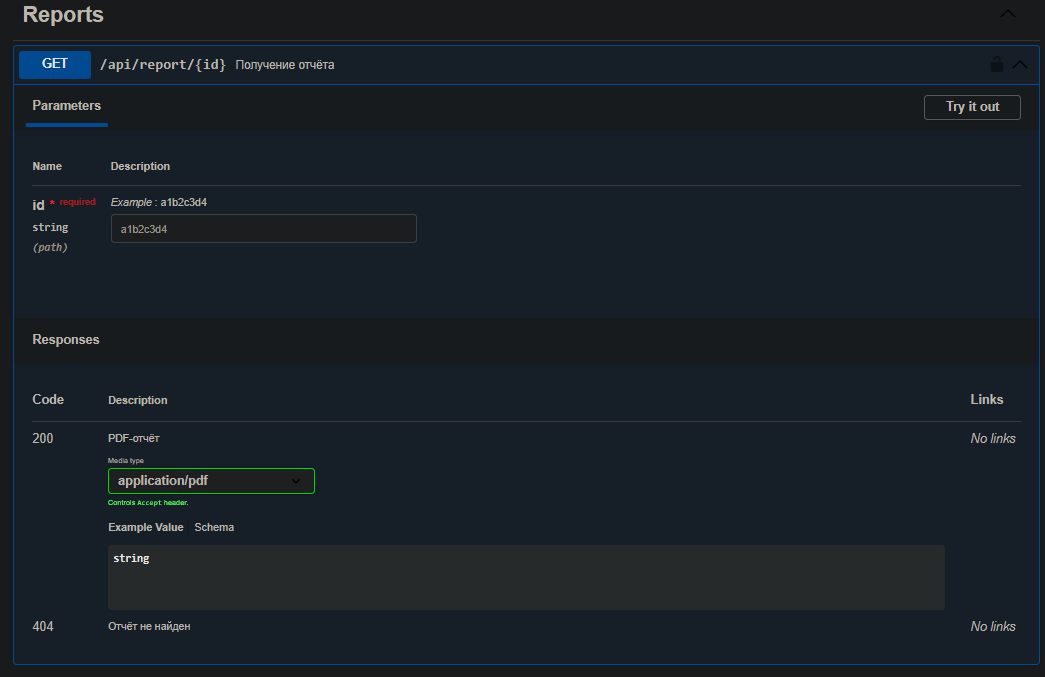
* Загрузка изображений (JPEG/PNG) и текстовых симптомов для анализа.
* Авторизация: JWT-токен (Bearer).
* Входные данные:
  + Формат: **multipart/form-data**
  + Поля:
    - **image**: Файл (до 10 МБ).
    - **symptoms**: JSON-описание симптомов.



**6.3 Получение отчёта**

Эндпоинт: **GET /api/report/{id}**  
Описание:

* Получение PDF-отчёта по результатам анализа.
* Авторизация: JWT-токен (Bearer).
* Параметры:
  + **id**: Идентификатор задачи (пример: **a1b2c3d4**).
* Ответы:
  + 200 OK: PDF-файл с результатами.



**Схемы данных**

DataUploadRequest:

| **Поле** | **Тип** | **Описание** |
| --- | --- | --- |
| **image** | Binary | Медицинское изображение |
| **symptoms** | JSON | Описание симптомов пациента |

ReportResponse:

| **Поле** | **Тип** | **Описание** |
| --- | --- | --- |
| **reportId** | String | Идентификатор отчёта |
| **result** | Object | Результат ИИ-анализа |

**Источники:**

1. [Swagger UI Documentation](https://swagger.io/tools/swagger-ui/)
2. *«RESTful API Design»* (книга, глава 5)
3. [OpenAPI 3.0 Specification](https://swagger.io/specification/)

**7. Customer Journey Map**

Цель: Визуализация взаимодействия пациента с платформой от регистрации до получения диагноза.

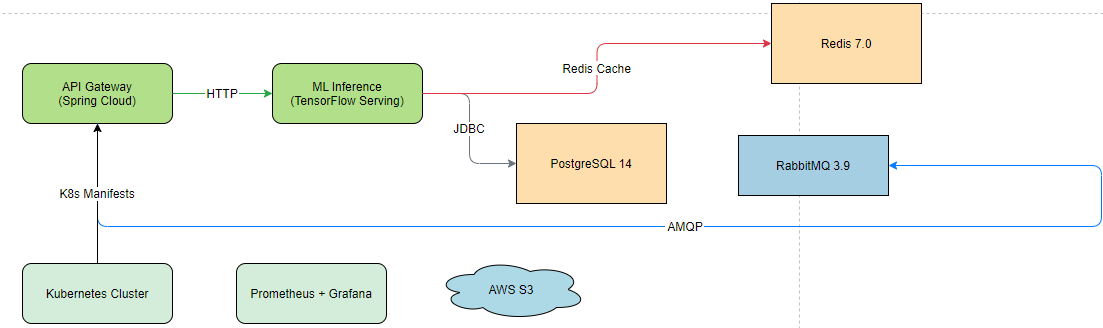
| Этап | Действия | Эмоции | Pain Points | Решения системы |
| --- | --- | --- | --- | --- |
| Регистрация | - Ввод email и пароля Подтверждение email | Нейтральные → Надежда на подтверждение | Сложная верификация- Ожидание письма | Автоматическая отправка письма - SMS-уведомление |
| Загрузка данных | Выбор файла- Описание симптомов | Тревога (качество данных) | Ограничение 10 МБ - Неверный формат | Валидация на фронте- Подсказки по форматам |
| Обработка | Ожидание анализа (2секунды) | Напряжение → Удовлетворение | Задержки при высокой нагрузке | Кэширование в Redis - GPU-кластер |
| Получение отчёта | Скачивание PDF Экспорт в MIS | Доверие → Облегчение | Нет доступа к MIS | Интеграция через REST API- Роли администратора |

Источники:

1. [CJM для медицинских сервисов](https://www.nngroup.com/articles/journey-mapping-medical/)
2. *«User Experience in Healthcare»* (книга, глава 7)

**8. Инфраструктура и деплой**

1. Инфраструктурные компоненты



| Компонент | Технология | Роль |
| --- | --- | --- |
| API Gateway | Spring Cloud Gateway | Маршрутизация, аутентификация (JWT) |
| ML Inference | TensorFlow Serving 2.10 | GPU-обработка (ResNet-50 + BERT) |
| Хранилище | PostgreSQL 14 + Redis 7.0 | ACID для метаданных + кэш для результатов |
| Брокер сообщений | RabbitMQ 3.9 | Очередь**me dical\_data** для асинхронности |
| Логирование | ELK Stack (Elasticsearch) | Мониторинг ошибок и производительности |
| Контейнеризация | Docker + Kubernetes | Оркестрация микросервисов |

**2. Процесс деплоя**

Этапы:

1. CI/CD Pipeline (GitLab):
   * Сборка образов:
   * Тестирование: Unit-тесты (JUnit), нагрузочное тестирование (JMeter).
2. Разворачивание в Kubernetes:
   * Манифесты Helm:
3. Масштабирование:
   * Автоматическое:
     + KEDA (Kubernetes Event-Driven Autoscaling) для RabbitMQ:
4. Мониторинг:
   * Prometheus:
     + Метрики GPU (DCGM Exporter).
     + Запросы к API (через Spring Actuator).
   * Grafana:
     + Дашборды для отслеживания задержек и ошибок.

**3. Безопасность и обновления**

* Аутентификация:
  + JWT с временем жизни 1 час.
  + Ротация ключей каждые 30 дней.
* Обновление моделей ИИ:
  + Canary Deployment для новых версий BERT/ResNet.
  + A/B-тестирование через Istio.

**4. Источники**

1. *«Kubernetes in Action»* Marko Luksa (глава 12 — масштабирование).
2. [RabbitMQ в Kubernetes](https://www.rabbitmq.com/kubernetes/operator/using-operator.html)
3. [DCGM Exporter для GPU](https://docs.nvidia.com/datacenter-cloud-native/kubernetes/dcgme-exporter.html)
4. [Istio для A/B-тестирования](https://istio.io/latest/docs/)