

**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ  
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ  
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук  
Департамент программной инженерии

**СОГЛАСОВАНО**

Приглашенный преподаватель  
базовой кафедры ПАО Сбербанк  
факультета компьютерных наук

\_\_\_\_\_ А.И. Калинин  
«\_\_\_» \_\_\_\_\_ 2025 г.

**УТВЕРЖДАЮ**

Академический руководитель  
образовательной программы  
«Программная инженерия» старший  
преподаватель департамента  
программной инженерии

\_\_\_\_\_ Н.А. Павлов  
«\_\_\_» \_\_\_\_\_ 2025 г.

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл.	

**Приложение для создания пешеходных маршрутов**

**Пояснительная записка**

**ЛИСТ УТВЕРЖДЕНИЯ**

**RU.17701729.05.06-01 81 01-1-ЛУ**

Исполнитель:  
студент группы БПИ224  
\_\_\_\_\_/ А. Евсюков /  
«\_\_\_» \_\_\_\_\_ 2025 г.

УТВЕРЖДЕНО  
RU.17701729.05.06-01 81 01-1-ЛУ

**Приложение для создания пешеходных маршрутов**  
**Пояснительная записка**

**RU.17701729.05.06-01 81 01-1**

**Листов 29**

Инв. № подл	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

## АННОТАЦИЯ

Данный документ является пояснительной запиской к проекту «Приложение для построения пешеходных маршрутов» и посвящён описанию его серверной части. Разрабатываемое приложение предоставляет пользователям возможность как создавать собственные маршруты для прогулок, так и открывать для себя новые места, следуя по маршрутам, предложенным другими.

Раздел «Введение» включает в себя название программного продукта, обозначение темы проекта, ссылку на документ, на основе которого ведётся разработка, а также информацию об организации, утвердившей данный проект.

В разделе «Назначение и область применения» раскрываются основные функции и условия эксплуатации программного продукта, а также даётся краткое описание сферы его использования.

Раздел «Технические характеристики» содержит такие подразделы, как формулировка задач, стоящих перед программой, описание её работы, функциональных возможностей, обоснование способов организации ввода и вывода данных, а также выбор используемых технических и программных средств.

В разделе «Ожидаемые технико-экономические показатели» рассматриваются предполагаемые потребности в использовании приложения и его преимущества с экономической точки зрения по сравнению с аналогичными отечественными и зарубежными решениями.

Настоящий документ разработан в соответствии с требованиями:

- 1) ГОСТ 19.101-77 Виды программ и программных документов;
- 2) ГОСТ 19.102-77 Стадии разработки;
- 3) ГОСТ 19.103-77 Обозначения программ и программных документов;
- 4) ГОСТ 19.105-78 Общие требования к программным документам;
- 5) ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом;

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.06-01 81 01–1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

## СОДЕРЖАНИЕ

<b>1. ВВЕДЕНИЕ.....</b>	<b>5</b>
1.1. Наименование программы.....	5
1.2. Документ, на основании которого ведется разработка .....	5
<b>2. НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ .....</b>	<b>6</b>
2.1. Назначение программы.....	6
2.1.1. Функциональное назначение .....	6
2.1.2. Эксплуатационное назначение .....	6
2.2. Краткая характеристика области применения .....	6
<b>3. ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ.....</b>	<b>7</b>
3.1. Постановка задачи на разработку программы.....	7
3.2. Описание и обоснование выбора состава технических и программных средств .....	7
3.2.1. Состав технических и программных средств .....	7
3.2.2. Обоснование выбора технических и программных средств.....	8
3.3. Описание архитектуры программного продукта.....	14
3.4. Описание работы с базой данных.....	17
3.5. Особенности функционала.....	18
3.5.1. Основные контроллеры, реализующие API для взаимодействия с клиентом ...	18
3.5.2. Организация межсервисного взаимодействия .....	21
3.5.3. Мониторинг состояния сервисов .....	23
3.6. Описание и обоснование выбора метода организации входных и выходных данных	24
<b>4. ОЖИДАЕМЫЕ ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ.....</b>	<b>25</b>
4.1. Предполагаемая потребность.....	25
4.2. Ориентировочная экономическая эффективность.....	25
4.3. Экономические преимущества разработки по сравнению аналогами .....	25
<b>5. ИСТОЧНИКИ, ИСПОЛЬЗУЕМЫЕ ПРИ РАЗРАБОТКЕ .....</b>	<b>27</b>
<b>ПРИЛОЖЕНИЕ 1 ТЕРМИНОЛОГИЯ.....</b>	<b>29</b>
<b>ТЕРМИНОЛОГИЯ.....</b>	<b>29</b>
<b>ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ .....</b>	<b>30</b>

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.06-01 81 01–1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

**1. ВВЕДЕНИЕ****1.1. Наименование программы**

**Наименование темы разработки:** «Приложение для создания пешеходных маршрутов».

**Условное обозначение темы разработки:** «Пойдём Daily».

**1.2. Документ, на основании которого ведется разработка**

Основанием для разработки является учебный план подготовки бакалавров по направлению 09.03.04 «Программная инженерия» и утвержденная академическим руководителем программы, тема курсового проекта.

**Организация, утвердившая этот документ:** Национальный исследовательский университет «Высшая школа экономики», Факультет компьютерных наук, образовательная программа «Программная инженерия».

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.06-01 81 01–1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

## 2. НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ

### 2.1. Назначение программы

#### 2.1.1. Функциональное назначение

Приложение предназначено для создания, публикации и совместного использования пешеходных маршрутов, помогая пользователям эффективно планировать прогулочные маршруты и открывать новые интересные места вокруг. Для удобства навигации и персонализации опыта в приложении реализованы инструменты интерактивного построения маршрутов с возможностью сохранения промежуточных результатов.

Основные функции приложения включают:

- Конструктор маршрутов с добавлением ключевых точек интереса и сохранением черновиков
- Поиск с фильтрацией по параметрам сложности, длительности и рейтингу
- Режим прогулки с функцией паузы и сохранения текущего прогресса
- Сохранение понравившихся маршрутов в избранные

Приложение ориентировано исключительно на пешие маршруты, предоставляя инструмент для планирования прогулок и исследования новых мест.

#### 2.1.2. Эксплуатационное назначение

Данное приложение будет интересно пользователям, которые увлекаются пешими прогулками и исследованием новых мест. Оно представляет собой инструмент для обмена маршрутами с другими пользователями и создания персонализированных пешеходных маршрутов.

Приложение призвано улучшать качество прогулок, помогая пользователям находить новые места и делиться своими маршрутами с другими.

### 2.2. Краткая характеристика области применения

Программа представляет собой мобильное приложение, предназначенное для создания пешеходных маршрутов с возможностью поделиться ими с другими пользователями или найти готовые маршруты. Кроме того, приложение позволяет сохранять понравившиеся маршруты, осуществлять поиск по фильтрам, а также управлять режимом прогулки: ставить на паузу и сохранять прогресс.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.06-01 81 01–1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

### **3. ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ**

#### **3.1. Постановка задачи на разработку программы**

Разрабатываемая бэкенд-часть программы должна:

1. Хранить информацию о пользователях, маршрутах и связанных с ними данных.
2. Реализовывать функционал управления объектным хранилищем, которое хранит фотографии пользователей и маршрутов.
3. Предоставлять хранимую информацию посредством создания REST API.
4. Обеспечивать безопасность пользовательских данных:
  - Реализовывать механизмы аутентификации и авторизации пользователей
  - Создавать и валидировать JWT-токены пользователей
  - Реализовывать механизмы управления доступом на основе ролевой модели
5. Реализовывать единую точку входа в систему и валидацию входящих запросов.
6. Реализовывать функционал отправки уведомлений:
  - Реализовывать механизм отправки сообщений на электронную почту пользователя
  - Обеспечивать асинхронное взаимодействие между сервисами для повышения отказоустойчивости
7. Реализовывать функционал мониторинга состояния системы:
  - Обеспечивать сбор и визуализацию метрик производительности (CPU, RAM, дисковое пространство)
  - Настроить информативные дашборды для отображения системных метрик и ключевых показателей приложения

#### **3.2. Описание и обоснование выбора состава технических и программных средств**

##### **3.2.1. Состав технических и программных средств**

Для работы программы необходим следующий состав программных средств:

1. Docker 24.0.0 или выше.
2. Spring Boot версии не ниже, чем 3.4.0.

## 3. PostgreSQL 15 и выше, с расширением PostGIS.

Для работы программы необходим следующий состав технических средств:

1. Доступ к сети интернет.
2. Серверная инфраструктура, которая будет отвечать следующим минимальным требованиям:
  - Процессор: 2 ядра
  - Оперативная память: 4 Гб
  - Публичный IP-адрес
  - Дисковое хранилище: 20 Гб

### 3.2.2. Обоснование выбора технических и программных средств

Для разработки бэкенд-части приложения был выбран язык программирования Kotlin и фреймворк Spring Boot 3.4.0, так как данный фреймворк является одним из наиболее популярных решений для разработки бэкенд приложений и предоставляет полный набор инструментов для разработки сложных, масштабируемых и отказоустойчивых систем.

В качестве решения для создания REST API был использован Spring Web, который часто используется для разработки RESTful API и базируется на архитектурном паттерне MVC (Model-View-Controller) (см. рис. 1). Данный паттерн достигается при помощи слабо связанных готовых компонентов. Паттерн разделяет аспекты приложения (логику ввода, бизнес-логику и логику UI), обеспечивая при этом свободную связь между ними.

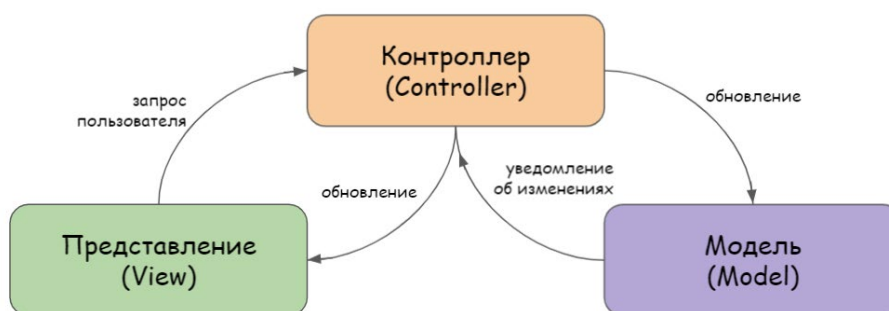


Рисунок 1 – Архитектурный паттерн MVC

Вся логика работы Spring MVC, входящего в Spring Web, строится вокруг DispatcherServlet (см. рис. 2), который принимает и обрабатывает все HTTP-запросы и ответы на них. Рабочий процесс обработки запроса DispatcherServlet'ом проиллюстрирован на следующей диаграмме:

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.06-01 81 01–1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата



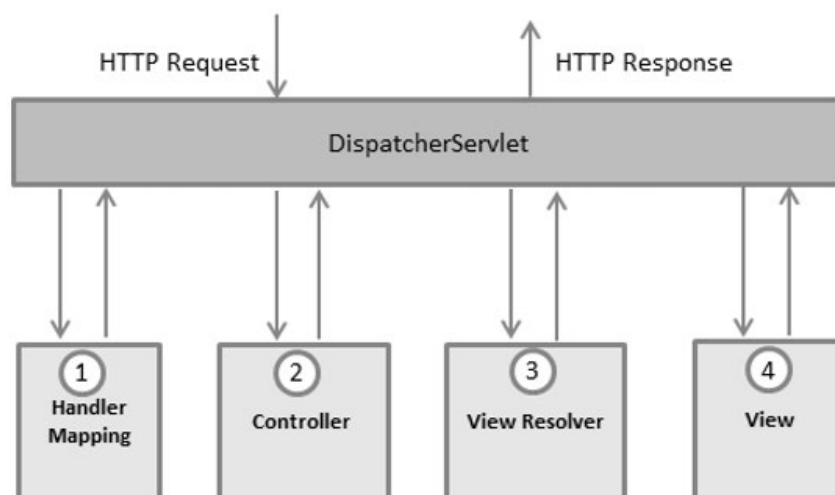


Рисунок 2 – Архитектура Spring MVC

Ниже приведена последовательность обработки входящего HTTP-запроса:

1. Маршрутизация запроса:

- DispatcherServlet получает HTTP-запрос
- Через интерфейс HandlerMapping определяется соответствующий контроллер
- Запрос перенаправляется в выбранный контроллер

2. Обработка в контроллере:

- Контроллер анализирует тип запроса (GET/POST/PUT/DELETE)
- Вызывается соответствующий метод сервиса для выполнения бизнес-логики
- Формируется модель данных для ответа

3. Формирование ответа:

- DispatcherServlet через ViewResolver определяет представление
- Модель передается в представление для визуализации

Однако, в отличие от классической MVC-архитектуры, Spring Boot включает в себя слой Service (см. рис. 3), который содержит всю бизнес-логику приложения, а также выступает в роли промежуточного звена между Controller и остальными элементами. Кроме того, в нашем случае, отсутствует слой View, так как за отображение отвечает мобильный

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.06-01 81 01-1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

клиент. Таким образом, этап формирования отчета имеет следующий вид:

- Слой Service обрабатывает бизнес-логику
- Данные возвращаются в формате JSON (без этапа View)
- Мобильное приложение самостоятельно обрабатывает ответ

## Spring Boot Flow Architecture

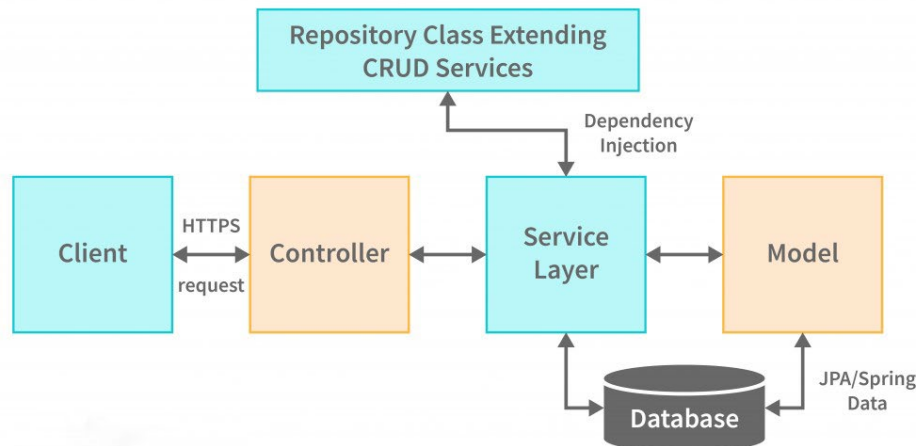


Рисунок 3 – Архитектура Spring Boot

. В качестве решения для аутентификации и авторизации был выбран Spring Security - стандартный фреймворк для защиты Spring-приложений, дополненный механизмом JWT-токенов (JSON Web Token). Данный подход обеспечивает безопасное взаимодействие между мобильным клиентом и сервером, соответствуя современным стандартам защиты REST API. Такой подход обеспечивает необходимую безопасность и гибкость: токен содержит данные пользователя, легко проверяется на клиенте и сервере, защищен от взлома подписью и сроком действия.

Для создания единой точки входа в систему был применён Spring Cloud Gateway, обеспечивающий маршрутизацию запросов между микросервисами (см. рис. 4). Этот подход основан на реактивной модели (WebFlux) и предоставляет гибкие механизмы для:

- Динамической маршрутизации – перенаправление запросов на соответствующие микросервисы на основе пути, заголовков или других параметров.
- Балансировки нагрузки – интеграция с Spring Cloud LoadBalancer для распределения трафика между инстансами сервисов.
- Фильтрации запросов – добавление/модификация заголовков, аутентификация

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.06-01 81 01–1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

(JWT), кэширование и логирование.

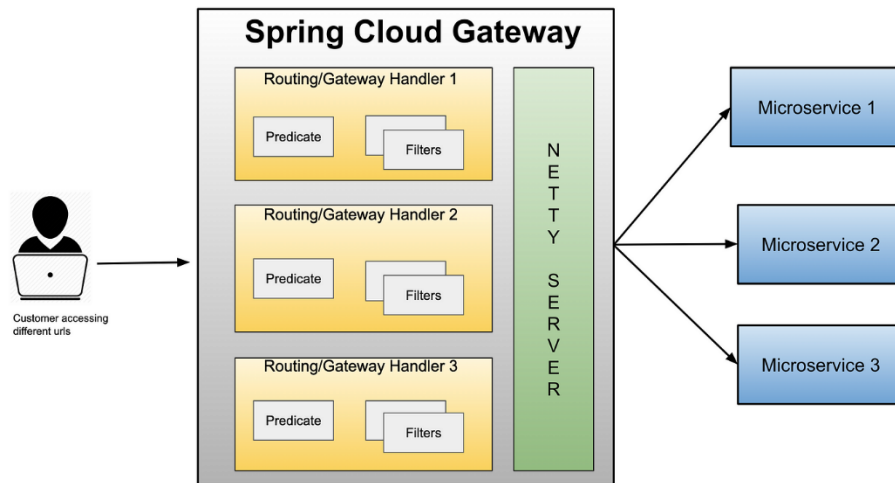


Рисунок 4 – Порядок работы маршрутизации в Spring Cloud Gateway

При организации отправки email-уведомлений был использован Apache Kafka. Этот инструмент является распределённым брокером сообщений, работающим по принципу публикации-подписки. Он выступает в качестве буфера между отправителями (producers) и получателями (consumers), записывая сообщения в различные топики (темы) (см. рис. 5). Данное решение позволило:

- Разгрузить основной сервис – отправка писем выполняется асинхронно через отдельный сервис уведомлений;
- Сократить время отклика – пользователь получает мгновенный ответ, не ожидая завершения отправки email;
- Обеспечить надёжность – сообщения сохраняются в очереди и гарантированно доставляются, даже при временной недоступности почтового сервиса.

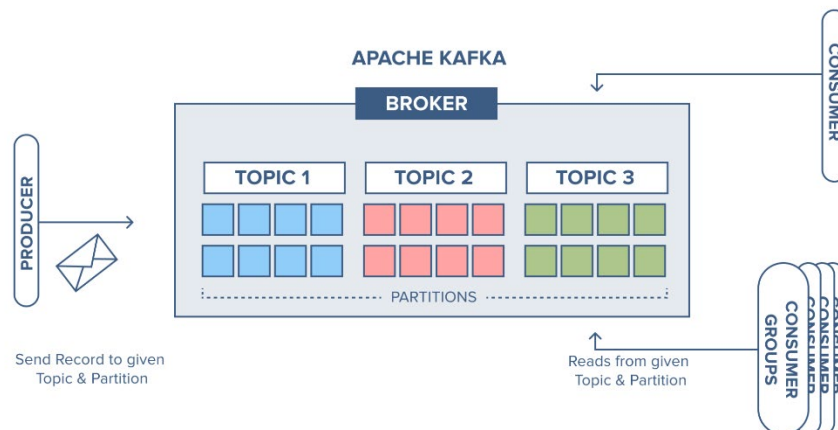


Рисунок 5 – Архитектура брокера сообщений Apache Kafka

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.06-01 81 01-1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

Для контейнеризации и упрощения процесса разработки был использован Docker — современный инструмент, позволяющий изолировать и воспроизводимо запускать программные компоненты в виде контейнеров. Такое решение позволяет разработчикам быстро поднимать все необходимые сервисы в локальной среде, минимизируя различия между окружениями и снижая риски "it works on my machine".

Применение Docker обеспечивает ряд ключевых преимуществ:

- Изоляция компонентов — каждый сервис (БД, сервер авторизации, бизнес-логика) работает в отдельном контейнере, не влияя на работу других.
- Ускорение локального тестирования — все зависимости можно поднять с помощью одной команды, что значительно упрощает проверку изменений в изолированной среде.
- Унификация среды — благодаря единой конфигурации в Dockerfile и docker-compose.yml разработчики, тестировщики и CI/CD-сервер используют одинаковую среду исполнения.
- Лёгкость масштабирования и деплоя — контейнеры легко разворачиваются как локально, так и в облачных средах, обеспечивая гибкость при развёртывании.

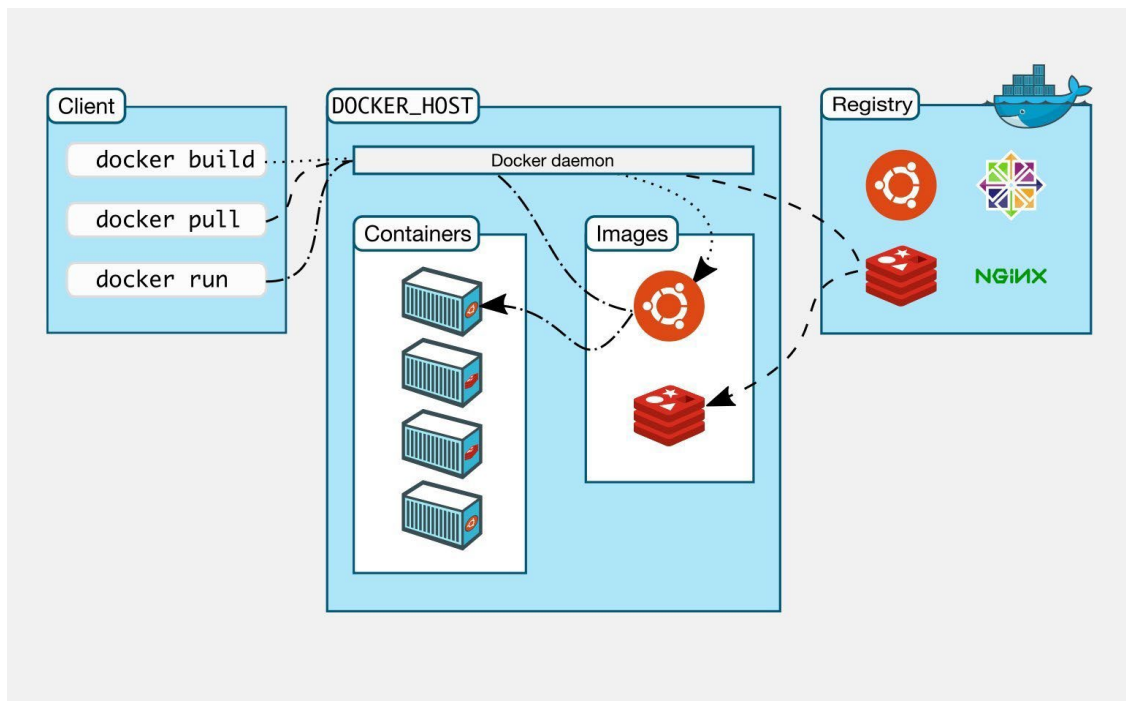


Рисунок 6 – Архитектура Docker

Для мониторинга состояния микросервисов в проекте использовалась связка Prometheus + Grafana, которая является одной из наиболее распространённых и гибких

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.06-01 81 01-1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

систем наблюдения за распределёнными приложениями.

Prometheus — это инструмент для сбора и хранения метрик в формате временных рядов. В контексте проекта он выполняет следующие функции:

- Периодически опрашивает зарегистрированные сервисы по HTTP (endpoint `actuator/prometheus`) и собирает метрики;
- Хранит метрики во внутреннем TSDB-хранилище;
- Позволяет формировать запросы к данным с помощью языка PromQL для анализа поведения системы во времени;
- Обеспечивает базовую визуализацию и возможность создания alert-правил.

Grafana используется в качестве визуального интерфейса поверх данных Prometheus. Она предоставляет:

- Наглядные дашборды для отображения состояния сервисов (нагрузка, ошибки, время ответа и др.);
- Гибкую настройку графиков, панелей и алертов;
- Возможность объединять данные из нескольких источников (в перспективе — Kafka, PostgreSQL и др.).

Каждый микросервис Spring Boot содержит встроенную поддержку сбора метрик благодаря библиотеке Micrometer, которая:

- Абстрагирует работу с системами мониторинга (Prometheus, Datadog, InfluxDB и др.);
- По умолчанию предоставляет метрики JVM, HTTP-запросов, пула потоков, ошибок и пользовательских таймеров;
- Позволяет легко добавлять собственные метрики (через Counter, Timer, Gauge и другие).

В сервисах была включена зависимость `micrometer-registry-prometheus`, благодаря которой метрики становятся доступными по пути `/actuator/prometheus`. Prometheus, в свою очередь, опрашивает эти эндпоинты по расписанию.

Причины выбора Prometheus и Grafana была обусловлена следующими факторами:

- Простота интеграции со Spring Boot благодаря Micrometer;
- Расширяемость и гибкость (возможность добавлять кастомные метрики и алерты);

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.06-01 81 01–1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

- Поддержка контейнеризированных сред и готовность к масштабируемым распределённым системам;
- Сообщество и наличие шаблонов дашбордов под большинство распространённых сценариев.

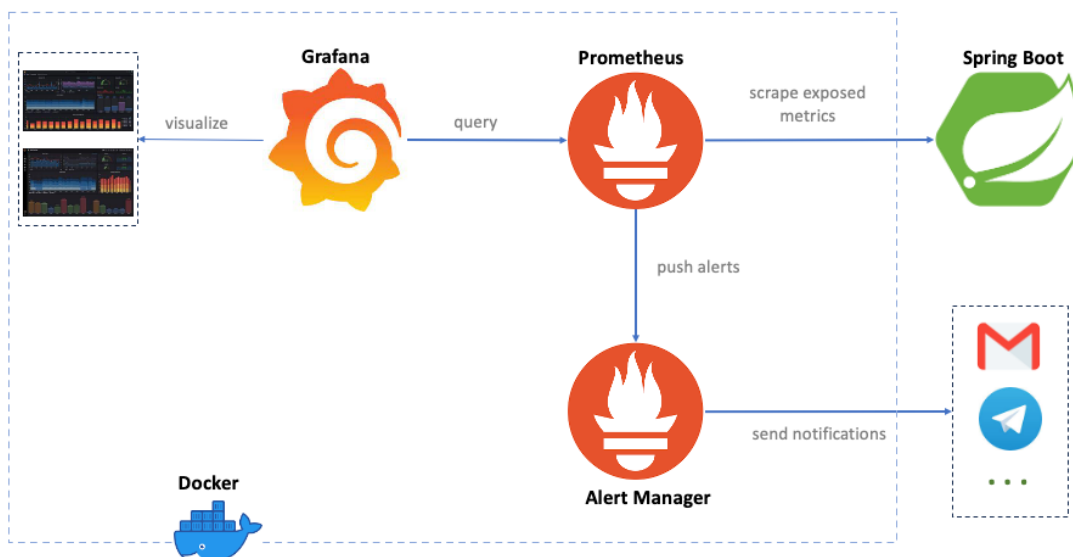


Рисунок 7 – Принцип сбора метрик Prometheus + Grafana в Spring Boot

### 3.3. Описание архитектуры программного продукта

В проекте использовалась микросервисная архитектура (см. рис. 8) с HTTP в качестве транспортного протокола. Система состоит из следующих ключевых сервисов:

#### 1. API Gateway

Единая точка входа, которая:

- Принимает и валидирует клиентские запросы
- Проверяет уровень доступа и аутентичность JWT-токенов
- Перенаправляет запросы на соответствующие сервисы

#### 2. Security Service

Обеспечивает:

- Аутентификацию и авторизацию пользователей
- Обработку всех запросов, связанных с учётными данными
- Генерацию JWT-токенов и кодов подтверждения email

#### 3. Data Provider

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.06-01 81 01-1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

Выполняет функции:

- Взаимодействия с базой данных
- Обработки всех запросов к БД
- Хранения фотографий пользователей и маршрутов в объектном хранилище

#### 4. Notification Service

Отвечает за:

- Отправку email-уведомлений
- Получение запросов от Security Service через Apache Kafka (основной канал) или HTTP (резервный)

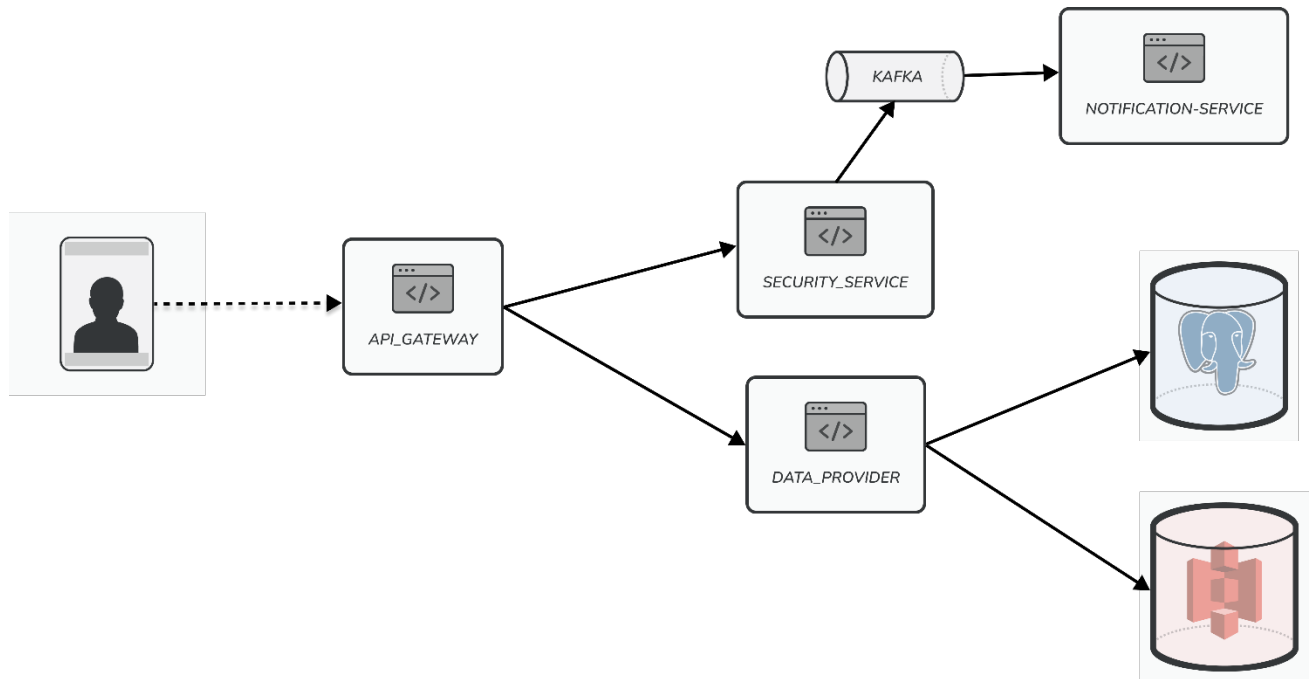


Рисунок 8 – Архитектура бэкенд-части приложения Пойдем.daily

На этапе проектирования рассматривался альтернативный подход, при котором функции Security Service могли быть реализованы непосредственно внутри API Gateway. Это позволило бы объединить логику проверки JWT, аутентификации, генерации токенов и маршрутизации в одном компоненте, что теоретически упрощает взаимодействие между сервисами и уменьшает задержки при авторизации.

Однако такой вариант был отклонён по ряду причин:

1. Технологическая несовместимость WebFlux и Spring Web (MVC):

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.06-01 81 01-1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

API Gateway реализован с использованием Spring Cloud Gateway, основанного на WebFlux, чтобы обеспечить высокую производительность и асинхронную обработку большого количества одновременных запросов. В то же время, логика безопасности (аутентификация, генерация токенов, отправка email-кодов подтверждения) требует использования Spring Security, который в полной мере работает только в контексте Spring Web (MVC). Попытка совместить реактивный и императивный подходы в одном приложении может привести к конфликтам зависимостей, ошибкам на этапе исполнения и затруднениям при конфигурировании фильтров безопасности.

## 2. Разделение ответственности и масштабируемость:

Выделение Security Service в отдельный сервис соответствует принципам Single Responsibility и Separation of Concerns. Это позволяет:

- изолировать чувствительную логику, связанную с учетными данными;
- гибко масштабировать компонент безопасности независимо от остальных частей системы;
- упростить сопровождение и настройку политики безопасности;
- обеспечить повышенную защищённость, в том числе при межсервисном взаимодействии через Kafka.

## 3. Разгрузка сервиса, работающего с БД (Data Provider):

Также рассматривался вариант совмещения логики безопасности с Data Provider, однако от него было решено отказаться, чтобы не перегружать сервис, отвечающий за всю работу с базой данных, включая хранение пользовательских данных и работу с объектным хранилищем. Такая комбинация нарушала бы модульность и затрудняла горизонтальное масштабирование.

## 4. Обособленность Notification Service:

Notification Service также был выделен в отдельный сервис, поскольку отправка email-сообщений — это длительная и потенциально нестабильная операция, особенно при взаимодействии с внешними SMTP-серверами. Для повышения отказоустойчивости и изоляции возможных сбоев эта задача выполняется асинхронно через Apache Kafka. Такой подход позволяет Security Service

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.06-01 81 01–1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата



быстро завершать обработку критических запросов, делегируя отправку писем в отдельный, устойчивый к задержкам и повторным попыткам сервис.

### 3.4. Описание работы с базой данных

Поскольку приложение подразумевает хранение и обработку геоданных, в качестве системы управления базами данных была выбрана PostgreSQL с расширением PostGIS, а для реализации сложных запросов – Spring Data JDBC.

PostGIS – это расширение для PostgreSQL, обеспечивающее поддержку географических объектов и пространственных данных. Оно позволяет выполнять пространственные запросы и анализировать местоположения.

PostGIS использует индексы на основе R-деревьев, что обеспечивает высокую производительность при выполнении пространственных операций, таких как поиск по близости, пересечение геометрий и проверка вложенности.

Поддерживаются различные типы геометрических объектов, включая точки, линии и полигоны. Расширение предоставляет широкий набор функций для пространственного анализа, включая буферизацию, объединение и определение пересечений.

Spring Data JDBC – это модуль проекта Spring Data, предоставляющий удобный и высокопроизводительный способ работы с реляционными базами данных через JDBC. Он сочетает простоту настройки с возможностью создавать сложные SQL-запросы без избыточного шаблонного кода. Spring Data JDBC ориентирован на прямое управление данными с помощью SQL, обеспечивая прозрачность, лёгкость отладки и полный контроль над выполнением запросов.

Ниже представлена итоговая структура базы данных (см. рис. 9).

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.06-01 81 01–1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

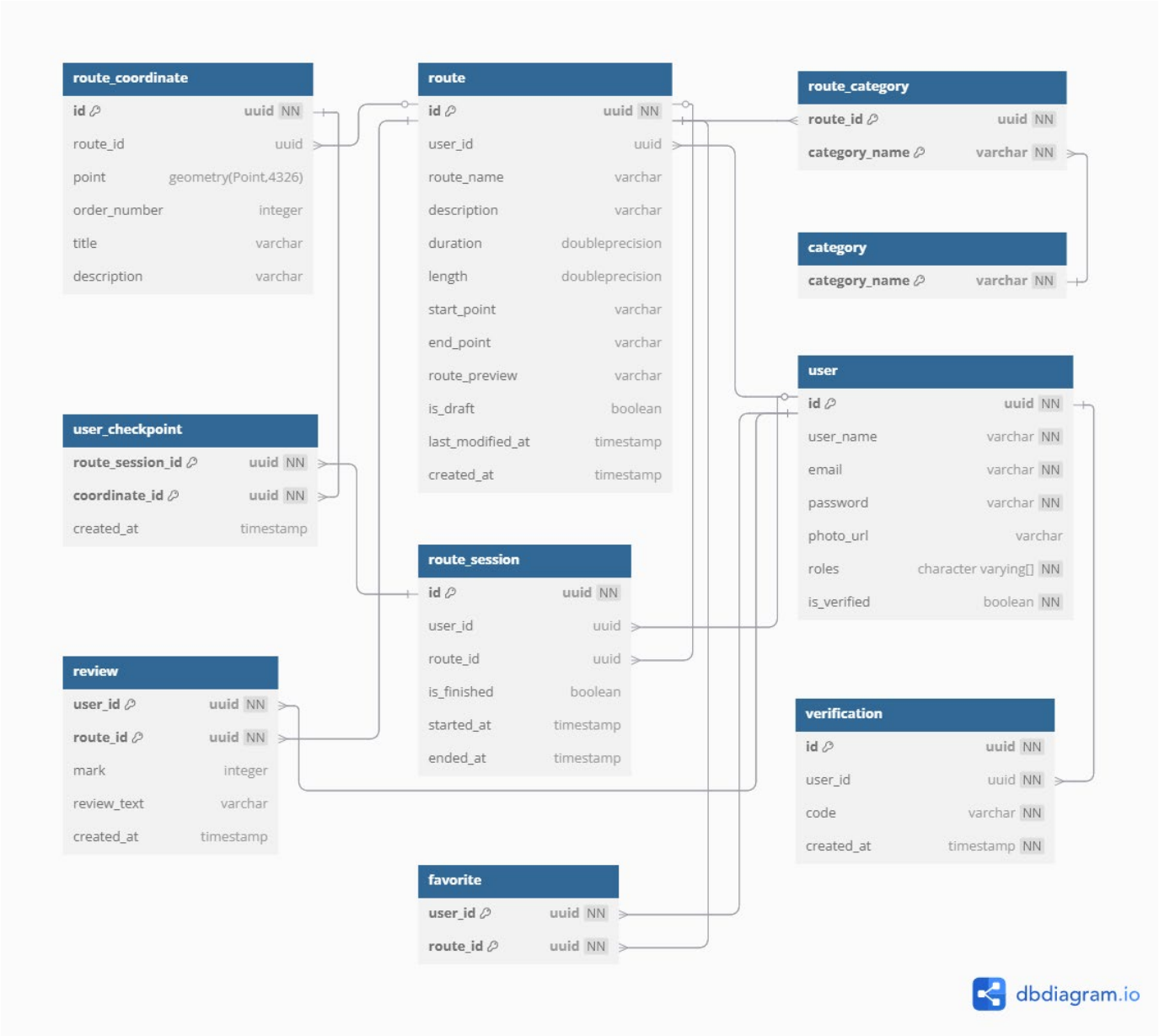


Рисунок 9 – Структура базы данных

3.5. Особенности функционала

3.5.1. Основные контроллеры, реализующие API для взаимодействия с клиентом

Все запросы, поступающие от пользователя, пройдя валидацию в API Gateway, попадают в следующие сервисы и их контроллеры:

- 1. Security Service:
  - UserController - контроллер для работы с пользователями: регистрация, вход, обновление информации и проверка статуса (см. рис. 10)

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.06-01 81 01-1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

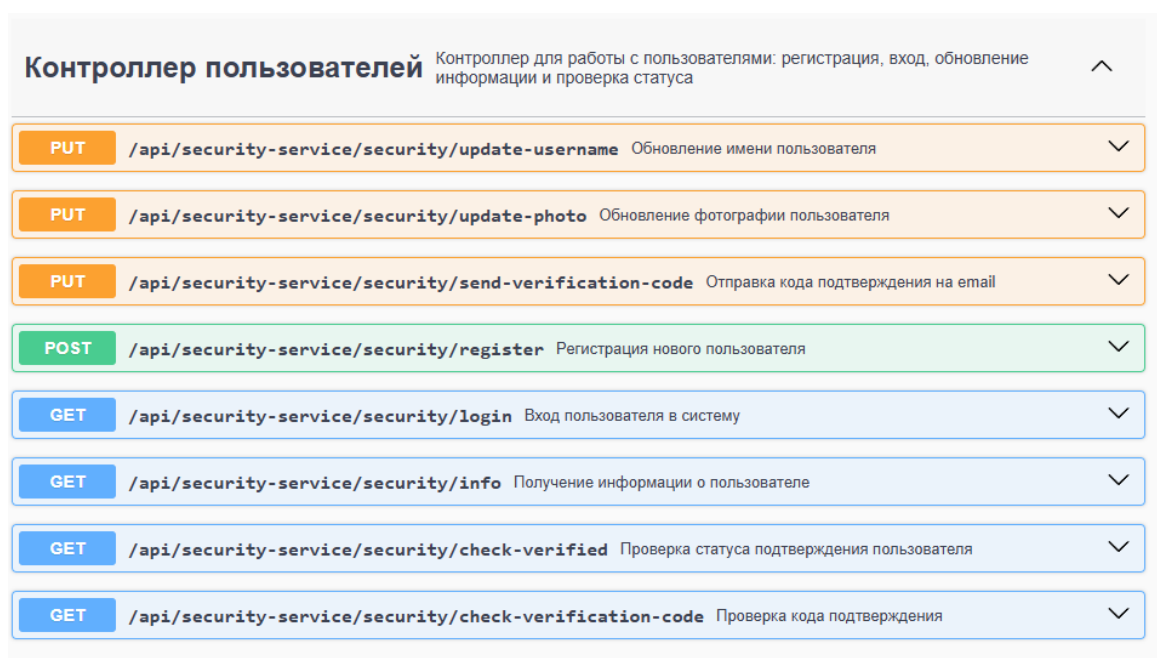


Рисунок 10 – Контроллер пользователей

## 2. Data Provider:

- RouteController - контроллер для работы с маршрутами: создание, удаление, поиск и получение маршрутов (см. рис. 11)

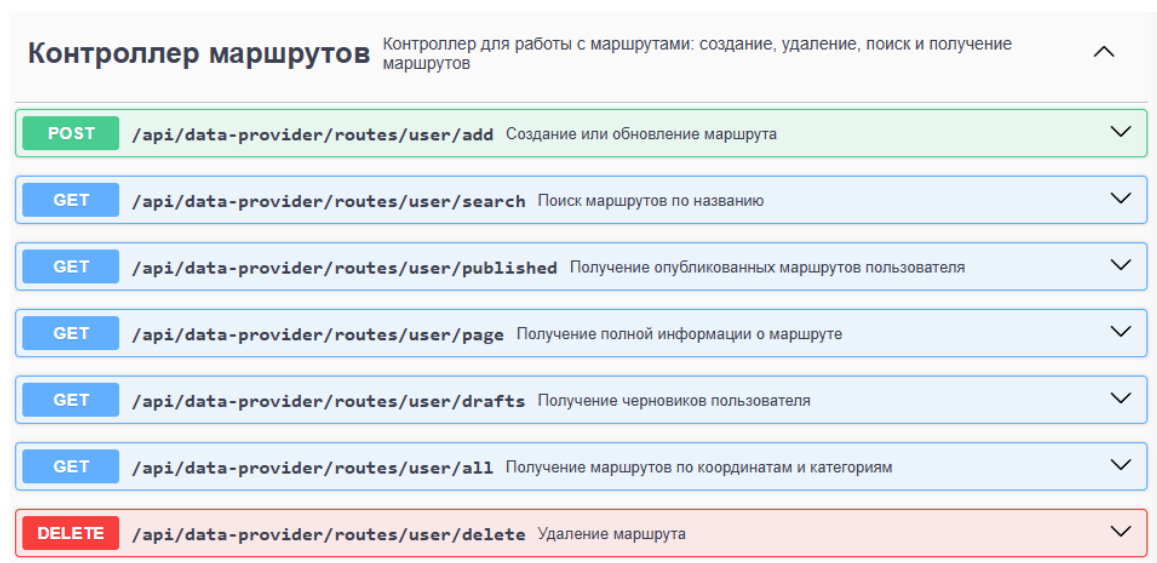


Рисунок 11 – Контроллер маршрутов

- RouteSessionController - контроллер для работы с сессиями маршрутов: получение завершённых/незавершённых, создание/обновление, получение по id (см. рис. 12)

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.06-01 81 01-1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

Контроллер сессий пользователей		Контроллер для работы с сессиями маршрутов: получение завершённых/незавершённых, создание/обновление, получение по id	^
POST	/api/data-provider/sessions/user/add	Создание или обновление сессии маршрута	▼
GET	/api/data-provider/sessions/user	Получение сессии маршрута по routeld	▼
GET	/api/data-provider/sessions/user/unfinished	Получение незавершённых сессий пользователя	▼
GET	/api/data-provider/sessions/user/finished	Получение пройденных маршрутов пользователя	▼

Рисунок 12 – Контроллер сессий пользователей

- ReviewController - контроллер для работы с отзывами к маршрутам: просмотр и добавление отзывов (см. рис. 13)

Контроллер отзывов		Контроллер для работы с отзывами к маршрутам: просмотр и добавление отзывов	^
POST	/api/data-provider/reviews/user/add	Добавление или обновление отзыва	▼
GET	/api/data-provider/reviews/user/all	Получение отзывов к маршруту	▼

Рисунок 13 – Контроллер отзывов

- FavoriteController - контроллер для получения, добавления и удаления избранных маршрутов пользователя (см. рис. 14)

Контроллер избранных маршрутов		Контроллер для получения, добавления и удаления избранных маршрутов пользователя	^
POST	/api/data-provider/favorites/user/add	Добавить маршрут в избранное	▼
GET	/api/data-provider/favorites/user/all	Получить избранные маршруты пользователя	▼
DELETE	/api/data-provider/favorites/user/delete	Удалить маршрут из избранного	▼

Рисунок 14 – Контроллер избранных маршрутов

- PhotoController - позволяет загружать и выгружать фотографии пользователей и маршрутов (см. рис. 15)

Контроллер загрузки фотографий		Позволяет загружать и выгружать фотографии пользователей и маршрутов	^
PUT	/api/data-provider/photos/user/upload	Загрузить фотографию	▼
GET	/api/data-provider/photos/user/download	Скачать фотографию	▼

Рисунок 15 – Контроллер фотографий

Полная спецификация эндпоинтов также доступна по адресу <http://localhost:8080/swagger-ui.html> при условии, что приложение запущено. Здесь localhost

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.06-01 81 01-1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

– хост вашей машины или сервера

3.5.2. Организация межсервисного взаимодействия

В архитектуре приложения используется API Gateway, через который проходят все внешние запросы. Он выполняет функции маршрутизации, предварительной валидации и, при необходимости, авторизации, после чего перенаправляет запросы в соответствующие внутренние сервисы.

Для наглядной демонстрации логики обработки пользовательских запросов и последовательности взаимодействия компонентов ниже приведены диаграммы последовательностей для некоторых из них. Остальные запросы обрабатываются аналогичным образом.

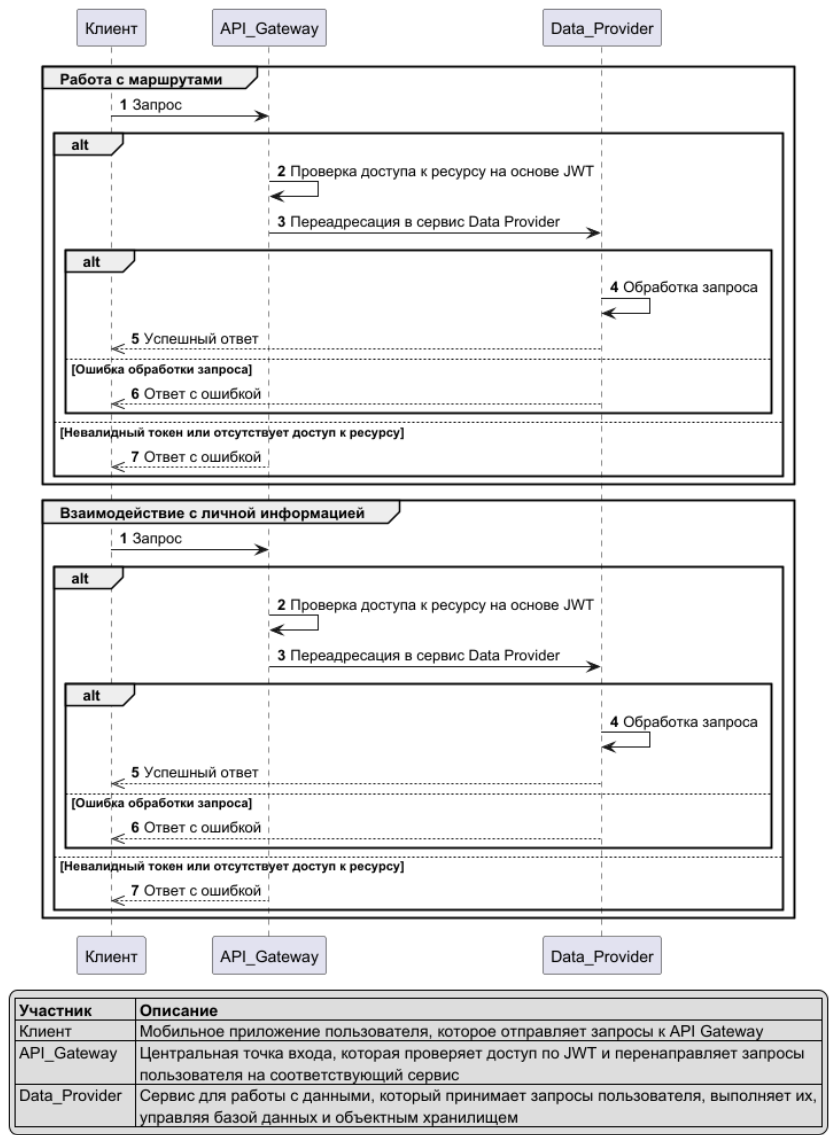


Рисунок 16 – Диаграммы последовательности для взаимодействия с данными и маршрутах

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.06-01 81 01–1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

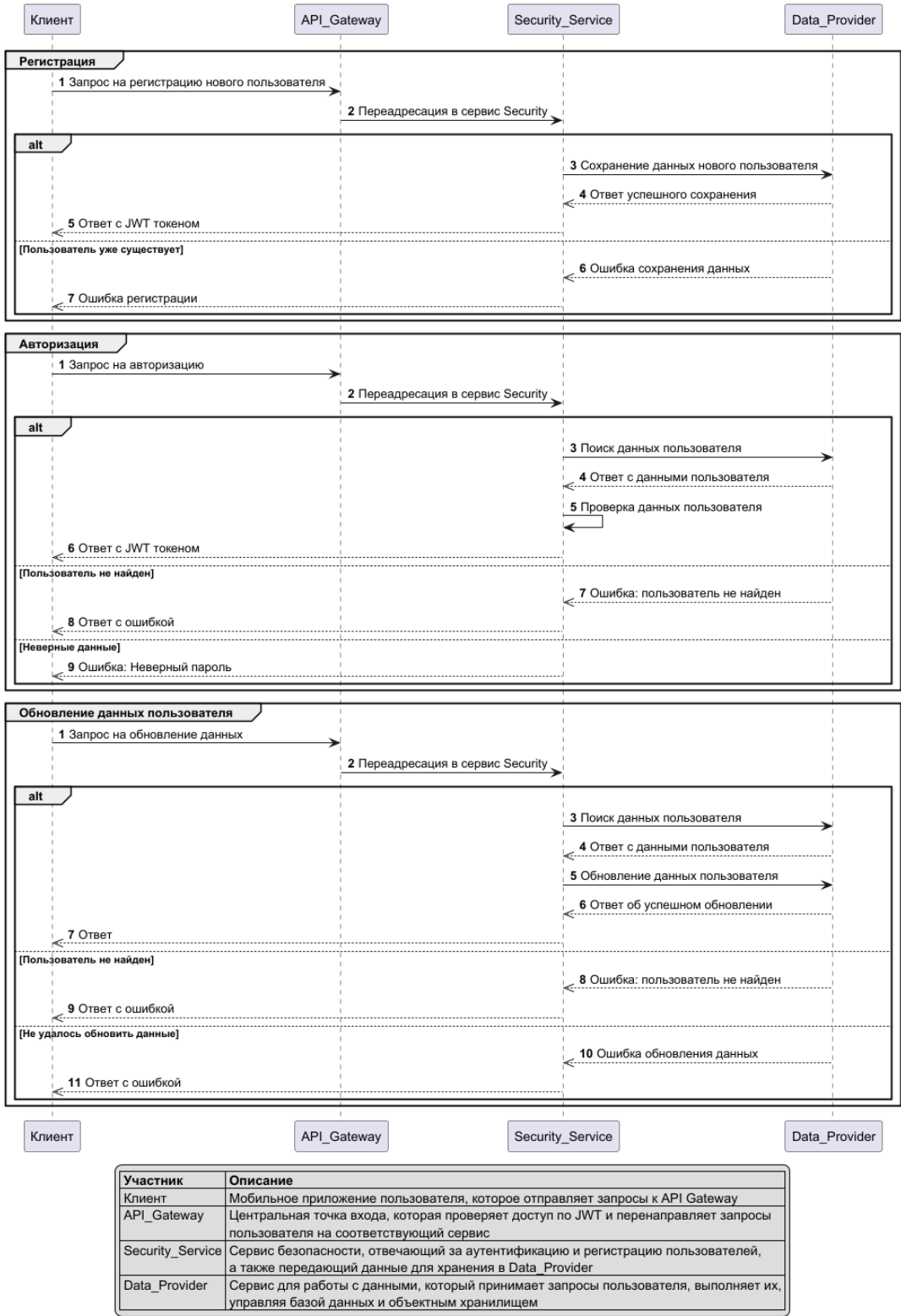


Рисунок 17 – Диаграммы последовательности для взаимодействия с данными пользователя

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.06-01 81 01-1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

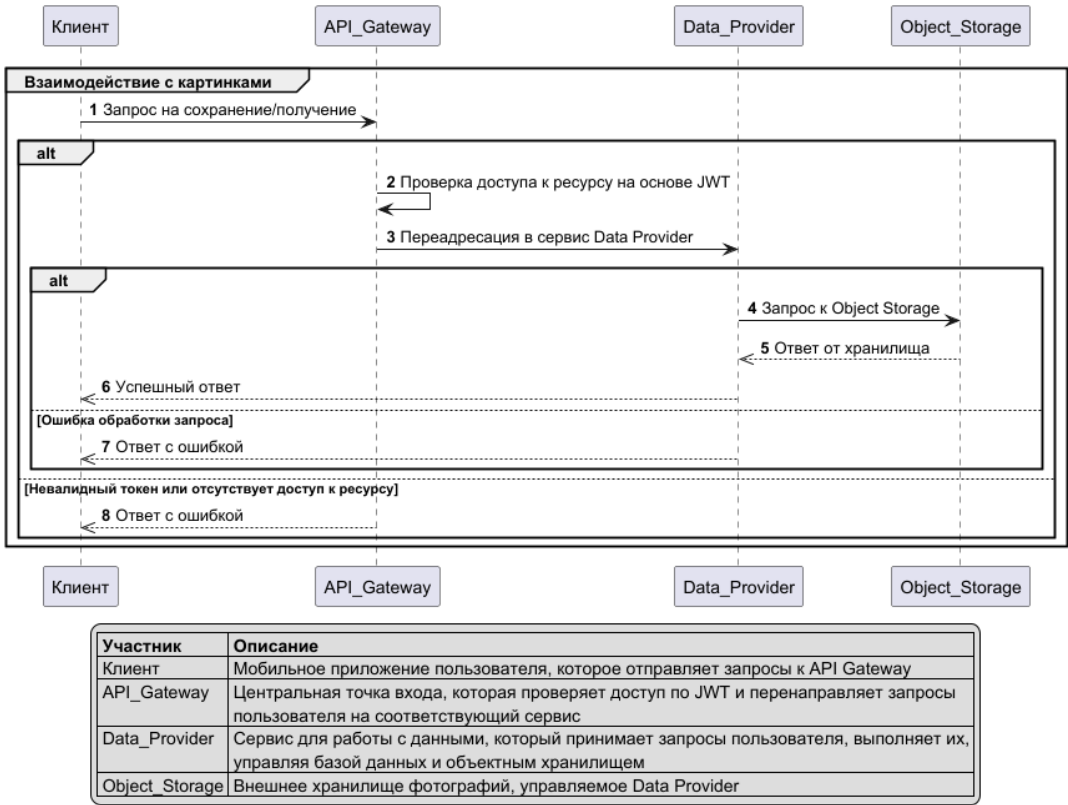


Рисунок 18 – Диаграммы последовательности для взаимодействия с объектным хранилищем

3.5.3. Мониторинг состояния сервисов

Для мониторинга состояния приложения были подключены Prometheus и Grafana. Prometheus – это система мониторинга и сбора метрик с открытым исходным кодом, предназначенная для отслеживания состояния приложений и инфраструктуры. В рамках данного проекта Prometheus осуществляет сбор основных метрик от каждого микросервиса, таких как время отклика, частота запросов, загрузка ресурсов и другие показатели.

Grafana – это платформа для визуализации и анализа данных мониторинга. Она предоставляет гибкие инструменты для создания информативных дашбордов, позволяющих в реальном времени отслеживать состояние системы на основе данных, собранных Prometheus (см. рис. 19).

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.06-01 81 01-1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

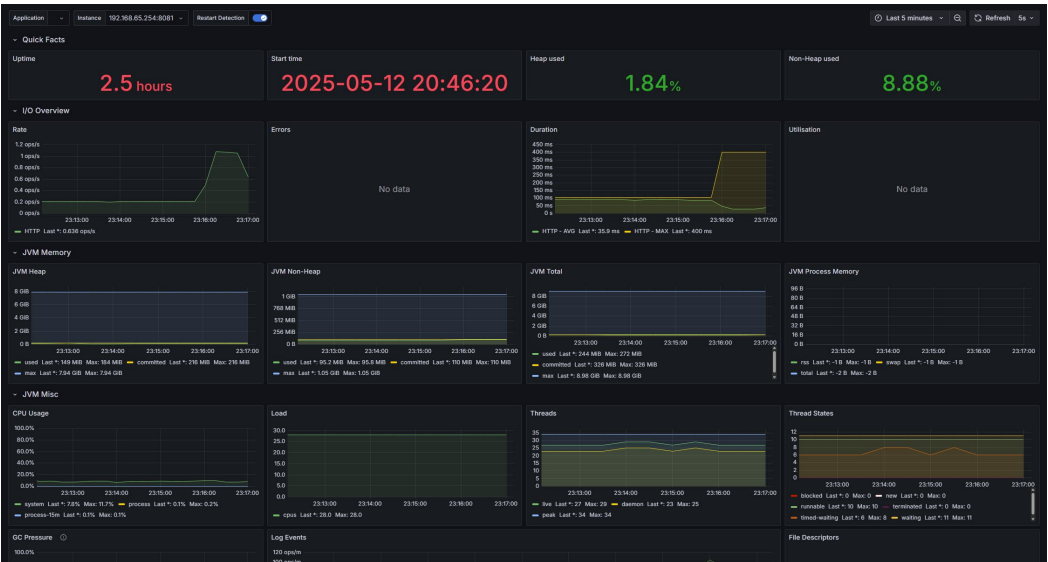


Рисунок 19 – Дашборд в Grafana с метриками Prometheus

**3.6. Описание и обоснование выбора метода организации входных и выходных данных**

Для обмена данными между клиентом и сервером в системе используется REST API, который является широко распространённым решением в разработке серверных приложений благодаря своей простоте, гибкости и соответствию современным стандартам. REST обеспечивает удобный способ взаимодействия, позволяя передавать данные, такие как JSON, с помощью стандартных HTTP-запросов и ответов.

Кроме того, REST API отличается высокой совместимостью с различными клиентскими приложениями и легко интегрируется с множеством платформ и технологий.

Таким образом, использование REST API в проекте обусловлено его универсальностью, модульностью и эффективной поддержкой современных подходов к построению клиент-серверных систем.

В данном приложении все запросы от клиента сначала поступают в API Gateway, где проходят валидацию, после чего перенаправляются в соответствующие сервисы. Там они обрабатываются контроллерами, запускающими дальнейшую бизнес-логику.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.06-01 81 01-1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата



#### 4. ОЖИДАЕМЫЕ ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ

##### 4.1. Предполагаемая потребность

Приложение ориентировано на любителей прогулок и активного отдыха, которым важно планировать пешеходные маршруты и делиться ими с другими. Возможности по созданию, редактированию, сохранению, поиску и обмену маршрутами сделают его привлекательным для данной аудитории.

##### 4.2. Ориентировочная экономическая эффективность

В рамках данного проекта оценка экономической эффективности создаваемого программного обеспечения не проводилась.

##### 4.3. Экономические преимущества разработки по сравнению аналогами

Для определения преимуществ разрабатываемого приложения был выполнен анализ его функциональности по сравнению с существующими аналогами. Сводные результаты приведены в таблице 1.

Таблица 1. Сравнение аналогов

Название	Strava[16]	AllTrails[17]	Яндекс Карты[18]	Outdooractive[19]	Wikiloc[20]	Komoot[21]	Пойдём Daily
Возможность создавать маршруты	+	+	+	+	+	+	+
Добавление ключевых точек на маршруте	—	+	+	+	+	—	+
Сохранение маршрута в черновик	—	—	—	—	—	—	+
Поиск маршрутов с фильтрами	+	+	—	—	+	+	+
Сортировка маршрутов	+	—	—	—	—	—	+
Возможность ставить маршрут на паузу	—	—	—	—	—	+	+

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.06-01 81 01–1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

Сохранение маршрутов в избранное	—	—	—	—	+	+	+
Возможность делиться маршрутами	+	+	+	+	+	+	+
Ориентация на пешеходные маршруты	—	+	—	—	+	+	+
Фокус на спорт / туризм	+	—	—	+	+	+	—
Фокус на создании маршрутов для прогулок	—	—	—	+	—	—	+
Доступно для использования в России	—	—	+	+	+	+	+
Карты регулярно обновляются и отображают актуальную информацию о маршруте и точках на нем	—	—	+	—	—	—	+
<b>Итого</b>	<b>4</b>	<b>5</b>	<b>5</b>	<b>6</b>	<b>8</b>	<b>8</b>	<b>12</b>

Сравнительный анализ показал, что разрабатываемое приложение «Пойдём Daily» имеет ряд существенных функциональных преимуществ по сравнению с существующими аналогами. Многие конкурирующие решения ориентированы преимущественно на англоязычных пользователей, а их картографические сервисы часто плохо подходят для использования на территории России. В отличие от них, «Пойдём Daily» предлагает интерфейс на русском языке и актуальные карты, адаптированные под российскую аудиторию. Это даёт приложению конкурентное преимущество на локальном рынке, где важна ориентация на потребности отечественных пользователей.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.06-01 81 01–1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

## 5. ИСТОЧНИКИ, ИСПОЛЬЗУЕМЫЕ ПРИ РАЗРАБОТКЕ

1. ГОСТ 19.101-77 Виды программ и программных документов. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
2. ГОСТ 19.102-77 Стадии разработки. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
3. ГОСТ 19.103-77 Обозначения программ и программных документов. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
4. ГОСТ 19.104-78 Основные надписи. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
5. ГОСТ 19.105-78 Общие требования к программным документам. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
6. ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
7. ГОСТ 19.201-78 Техническое задание. Требования к содержанию и оформлению. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
8. ГОСТ 19.603-78 Общие правила внесения изменений. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
9. ГОСТ 19.604-78 Правила внесения изменений в программные документы, выполненные печатным способом. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
10. ГОСТ 15150-69 Машины, приборы и другие технические изделия. Исполнения для различных климатических районов. Категории, условия эксплуатации, хранения и транспортирования в части воздействия климатических факторов внешней среды. – М.: Изд-во стандартов, 1997.
11. ГОСТ 19.301-79 Программа и методика испытаний. Требования к содержанию и оформлению. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
12. Docker Documentation. [Электронный ресурс], URL: <https://docs.docker.com/> (дата обращения: 05.05.2025).
13. Official PostgreSQL Documentation. [Электронный ресурс], URL: <https://www.postgresql.org/docs/> (дата обращения: 05.05.2025).
14. PostGIS Documentation [Электронный ресурс], URL: <https://postgis.net/documentation/> (дата обращения: 05.05.2025).

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.06-01 81 01-1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

15. Spring Framework Reference Documentation. [Электронный ресурс], URL: <https://docs.spring.io/spring-framework/docs/current/reference/html/> (дата обращения: 05.05.2025).
16. Strava. [Мобильное приложение], URL: <https://www.strava.com/> (дата обращения: 05.05.2025).
17. AllTrails. [Мобильное приложение], URL: <https://www.alltrails.com/> (дата обращения: 05.05.2025).
18. Яндекс Карты. [Мобильное приложение], URL: <https://yandex.ru/maps> (дата обращения: 05.05.2025).
19. Outdooractive. [Мобильное приложение], URL: <https://www.outdooractive.ru/> (дата обращения: 05.05.2025).
20. Wikiloc. [Мобильное приложение], URL: <https://ru.wikiloc.com/> (дата обращения: 05.05.2025).
21. Komoot. [Мобильное приложение], URL: <https://www.komoot.com/> (дата обращения: 05.05.2025).
22. Spring MVC – Framework. [Электронный ресурс], URL: [https://www.tutorialspoint.com/spring/spring\\_web\\_mvc\\_framework.htm](https://www.tutorialspoint.com/spring/spring_web_mvc_framework.htm) (дата обращения: 05.05.2025)
23. Spring Security. [Электронный ресурс], URL: <https://docs.spring.io/spring-security/reference/index.html> (дата обращения: 05.05.2025)
24. Spring Data JDBC. [Электронный ресурс], URL: <https://docs.spring.io/spring-data/jdbc/reference/> (дата обращения: 05.05.2025)
25. Spring Cloud Gateway. [Электронный ресурс], URL: <https://docs.spring.io/spring-cloud-gateway/reference/> (дата обращения: 05.05.2025)
26. Spring for Apache Kafka. [Электронный ресурс], URL: <https://docs.spring.io/spring-kafka/reference/> (дата обращения: 05.05.2025)
27. Prometheus для Spring Boot. [Электронный ресурс], URL: <https://micrometer.io/docs/registry/prometheus> (дата обращения: 05.05.2025)
28. Grafana. [Электронный ресурс], URL: <https://grafana.com/docs/grafana/latest/> (дата обращения: 05.05.2025)

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.06-01 81 01-1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

## ТЕРМИНОЛОГИЯ

Таблица 2

Термин	Определение
База данных	Совокупность данных, организованных в соответствии с концептуальной структурой, описывающей характеристики этих данных и взаимоотношения между ними, которая поддерживает одну или более областей применения
Бэкенд	Часть веб-приложения, отвечающая за обработку данных и бизнес-логику, которая скрыта от пользователя.
Программное обеспечение	Совокупность программных и документальных средств для создания и эксплуатации систем обработки данных средствами вычислительной техники.
RESTful API	Архитектурный стиль взаимодействия компонентов приложения с использованием стандартных HTTP-запросов для управления ресурсами.
Микросервис	Независимые компоненты приложения, каждый из которых выполняет определённую бизнес-задачу и взаимодействует с другими через API.
JWT-токен	Компактный формат передачи данных, используемый для аутентификации и авторизации, обеспечивающий безопасность взаимодействия между клиентом и сервером.
Дашборд (панель мониторинга)	Интерактивная визуальная панель, отображающая ключевые метрики системы или приложения в реальном времени, используемая для анализа состояния и выявления аномалий.
JSON (JavaScript Object Notation)	Лёгкий текстовый формат обмена данными, используемый для передачи информации между клиентом и сервером.
Load Balancer	Механизм распределения входящего сетевого трафика между несколькими серверами для обеспечения отказоустойчивости и производительности.
Producer (производитель)	Компонент, отправляющий сообщения в Kafka-топик.
Consumer (потребитель)	Компонент, подписывающийся на Kafka-топик и обрабатывающий поступающие сообщения.
Топик (Topic)	Категория или канал в Apache Kafka, куда публикуются и из которого читаются сообщения.
Микросервисная архитектура	Архитектурный стиль, при котором приложение состоит из набора мелких, независимых сервисов, взаимодействующих друг с другом через API.
R-дерево	Структура данных для индексирования многомерной информации, такой как географические координаты, используемая в PostGIS.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.06-01 81 01–1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

## ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

[illegible]