Geekbrains

**Исследование особенностей разработки веб-приложения для управления проектами на примере использования фреймворка Spring**

Программа: Разработчик

Специализация: Программист. Веб-разработка на Java

Козиков Андрей Викторович

Трехгорный

2024

Оглавление

[Введение 3](#_Toc183725952)

[1 Анализ предметной области 4](#_Toc183725953)

[2 Проектная часть 8](#_Toc183725954)

[3 Реализация задачи 15](#_Toc183725955)

[4 Тестирование приложения 24](#_Toc183725956)

[5 Интеграция мониторинга с Prometheus и Grafana 26](#_Toc183725957)

[6 Документация с использованием Javadoc 29](#_Toc183725958)

[7 Использование Spring Boot Starter Validation 30](#_Toc183725959)

[8 Использование Spring AOP 33](#_Toc183725960)

[Заключение 34](#_Toc183725961)

[Список использованной литературы 36](#_Toc183725962)

# Введение

Современные информационные технологии стремительно развиваются, предоставляя мощные инструменты для управления данными, оптимизации бизнес-процессов и автоматизации повседневных задач. В условиях динамичного рынка, где скорость и эффективность играют ключевую роль, необходимость в надежных и удобных системах управления проектами становится все более актуальной.

В данной пояснительной записке представлено описание веб-приложения для управления проектами, разработанного с использованием фреймворка Spring. Это приложение предназначено для облегчения управления проектами и взаимодействия между участниками команды, предоставляя интуитивно понятный интерфейс и мощные функции для работы с проектами и пользователями.

Основной целью разработки является создание системы, которая позволит пользователям эффективно управлять проектами, отслеживать их статус и взаимодействовать с другими участниками команды.

Разработка данного веб-приложения направлена на решение актуальных задач в области управления проектами. С его помощью команды смогут более эффективно организовывать свою работу, улучшать коммуникацию между участниками и повышать общую продуктивность. Использование современных технологий разработки обеспечивает надежность и масштабируемость приложения, что делает его подходящим для использования как в малых, так и в крупных организациях.

# Анализ предметной области

1.1 Понятие веб-сайт

Веб-сайт представляет собой совокупность связанных веб-страниц, доступных в сети Интернет и служащих для предоставления информации или услуг пользователям. Веб-сайты могут иметь различное назначение и функциональность, включая:

Информационные сайты: Предоставляют пользователям доступ к различным видам информации, таким как статьи, новости, исследования и справочные материалы. Эти сайты часто используются для распространения знаний и новостей.

Коммерческие сайты: Ориентированы на продажу товаров и услуг. Они могут включать интернет-магазины, платформы для бронирования и другие коммерческие сервисы, позволяющие пользователям совершать покупки онлайн.

Социальные сети: Платформы, которые позволяют пользователям взаимодействовать друг с другом, обмениваться информацией, фотографиями и видео. Примеры включают Facebook, Instagram и Twitter.

Образовательные сайты: Предоставляют учебные материалы и курсы для пользователей, желающих получить знания в различных областях. Эти сайты могут включать онлайн-курсы, вебинары и ресурсы для самообучения.

Развлекательные сайты: Включают платформы для игр, видео и других форм развлечений. Они предназначены для отдыха и досуга пользователей.

Веб-приложение для управления проектами

В рамках данного проекта разработано веб-приложение для управления проектами, которое сочетает в себе элементы различных типов веб-сайтов. Это приложение предоставляет пользователям возможность эффективно управлять проектами и взаимодействовать с другими участниками команды. Основные функции приложения включают:

* Создание и управление проектами: Пользователи могут создавать новые проекты, добавлять задачи и отслеживать их выполнение.
* Управление пользователями: Приложение позволяет администраторам управлять учетными записями пользователей, назначать роли и права доступа.
* Интуитивно понятный интерфейс: Разработанный интерфейс обеспечивает легкость навигации по приложению, что позволяет пользователям быстро находить необходимые функции.

Данное веб-приложение иллюстрирует как современные технологии могут быть использованы для создания эффективных инструментов управления, способствующих улучшению взаимодействия между членами команды и повышению общей продуктивности работы.

1.2 Клиент-серверные технологии разработки

Клиент-серверная архитектура является основой большинства современных веб-приложений. Этот подход предполагает разделение функций между клиентом (пользовательским интерфейсом) и сервером (логикой обработки данных). В такой архитектуре клиент отвечает за взаимодействие с пользователем, предоставляя интерфейс для ввода данных и отображения результатов, тогда как сервер обрабатывает запросы, выполняет бизнес-логику и управляет данными.

Принципы клиент-серверной архитектуры

* Разделение обязанностей: Клиент и сервер выполняют разные роли, что позволяет оптимизировать их работу. Клиент обрабатывает пользовательский интерфейс и взаимодействие, а сервер управляет данными и логикой приложения.
* Коммуникация через протоколы: Клиент и сервер общаются между собой с помощью стандартных протоколов, таких как HTTP/HTTPS. Это обеспечивает совместимость между различными платформами и устройствами.
* Масштабируемость: Клиент-серверная архитектура позволяет легко масштабировать приложение. Например, можно добавить больше серверов для обработки увеличенного объема запросов или улучшить клиентскую часть без изменения серверной логики.
* Безопасность: Разделение клиента и сервера позволяет реализовать более надежные механизмы безопасности, такие как аутентификация и авторизация пользователей на сервере.

Использование фреймворка Spring

Фреймворк Spring, используемый в данном проекте, предоставляет мощные средства для разработки серверной части веб-приложений. Он предлагает множество компонентов и библиотек, которые упрощают создание RESTful API, управление зависимостями и настройку безопасности. Основные особенности Spring включают:

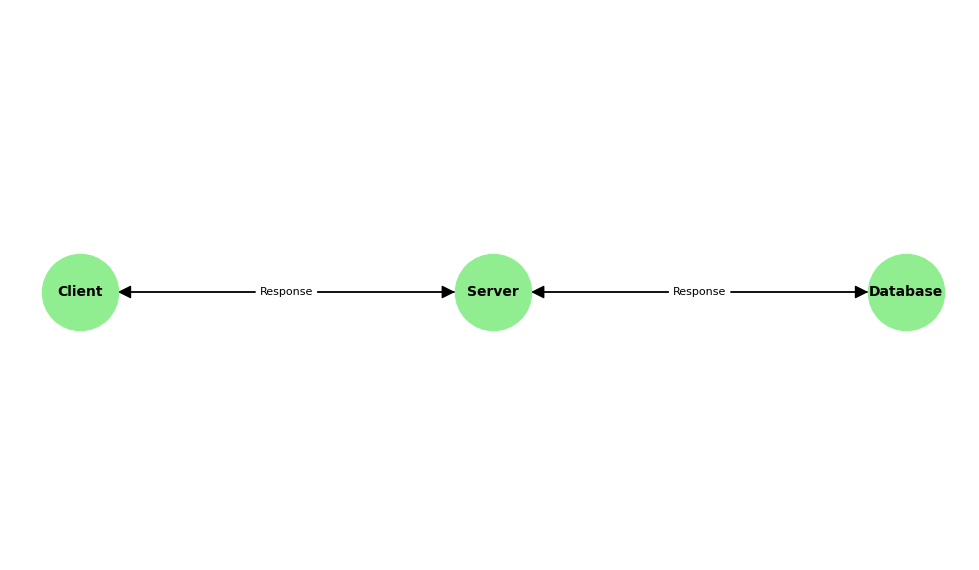
* Spring MVC: Позволяет создавать веб-приложения с использованием паттерна Model-View-Controller (MVC), обеспечивая четкое разделение между представлением, контроллерами и бизнес-логикой.
* Spring Security: Обеспечивает безопасность приложения через аутентификацию и авторизацию пользователей. С помощью JWT (JSON Web Tokens) можно реализовать безопасный доступ к ресурсам приложения.
* Spring Data JPA: Упрощает работу с базами данных, позволяя разработчикам сосредоточиться на бизнес-логике вместо написания большого количества кода для доступа к данным.
* Spring Boot: Обеспечивает быструю настройку и развертывание приложений, позволяя разработчикам быстро создавать и тестировать прототипы.

Рисунок 1. Схема клиент-серверной архитектуры

1.3 Постановка задачи на разработку

Задачей разработки является создание веб-приложения, которое будет служить эффективным инструментом для управления проектами и взаимодействия между пользователями. Основные требования к функциональности приложения включают:

* Управление проектами и связями между пользователями и проектами:

Приложение должно предоставлять пользователям возможность создавать, редактировать и удалять проекты, а также управлять связями между пользователями и проектами. Это включает в себя возможность добавления пользователей к проектам, удаления их из проектов и просмотра информации о текущих участниках.

* Аутентификация пользователей с использованием JWT:

Для обеспечения безопасности доступа к приложению необходимо реализовать систему аутентификации пользователей. Использование JSON Web Tokens (JWT) позволит безопасно идентифицировать пользователей и управлять их сессиями. При успешной аутентификации пользователю будет выдан токен, который он будет использовать для доступа к защищенным ресурсам приложения.

* Редактирование, удаление и поиск данных о проектах и пользователях:

Приложение должно обеспечивать функции редактирования и удаления информации как о проектах, так и о пользователях. Пользователи смогут искать проекты и пользователей по различным критериям, что упростит навигацию по системе и улучшит пользовательский опыт.

* Дополнительные требования

Безопасность данных: Важно обеспечить защиту данных пользователей и проектов от несанкционированного доступа. Это включает в себя реализацию механизмов шифрования для хранения конфиденциальной информации.

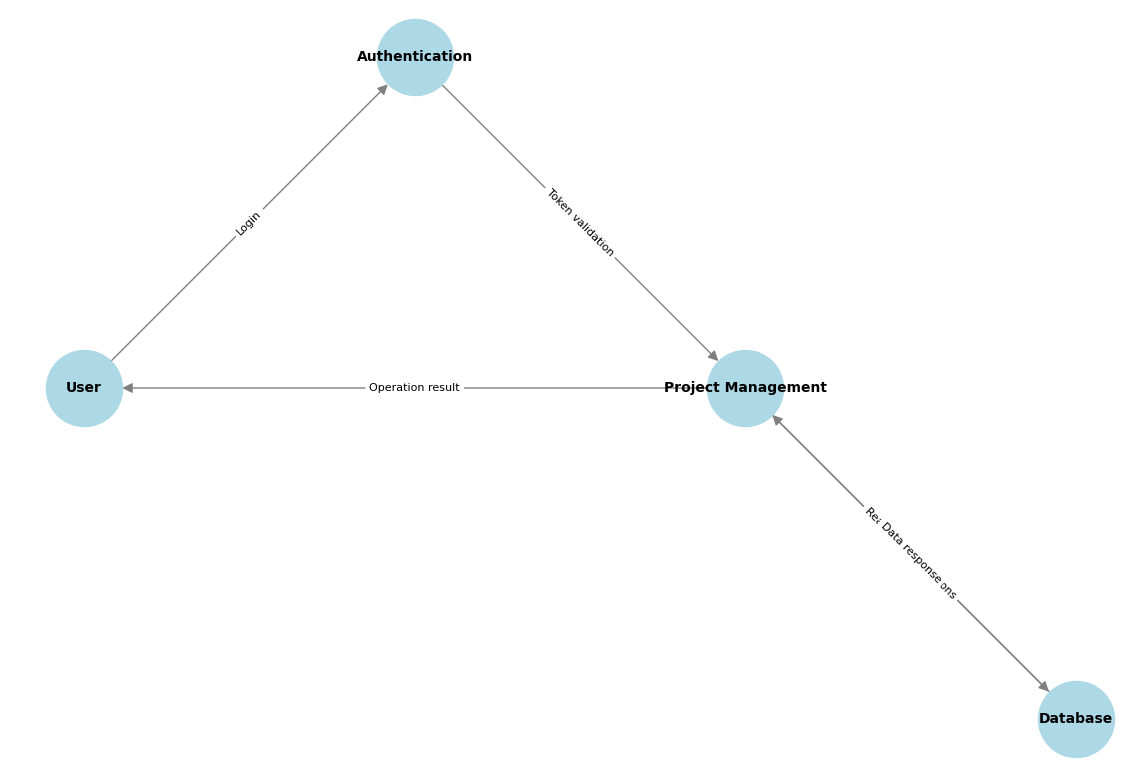
Масштабируемость: Архитектура приложения должна быть спроектирована таким образом, чтобы в будущем можно было легко добавлять новые функции или расширять существующие без значительных изменений в кодовой базе.

Рисунок 2. Схема взаимодействия пользователей и системы

# 2 Проектная часть

2.1 Обоснование проектных решений по техническому обеспечению

Для реализации проекта выбрано следующее техническое обеспечение, которое обеспечивает надежность, безопасность и масштабируемость веб-приложения:

Сервер: Java 17 с фреймворком Spring.

Java 17 является одной из последних версий языка, предлагающей улучшенные возможности производительности и новые функции, такие как паттерны записи и улучшенная работа с потоками. Фреймворк Spring предоставляет мощные инструменты для разработки серверной части веб-приложений, включая поддержку создания RESTful API, управление зависимостями и интеграцию с различными базами данных.

База данных: H2 для удобства тестирования и разработки.

H2 — это легковесная реляционная база данных, которая идеально подходит для разработки и тестирования. Она позволяет быстро настраивать и запускать базу данных без необходимости установки сложного программного обеспечения. H2 поддерживает SQL и может быть легко интегрирована с приложениями на Spring, что делает ее удобным выбором для разработки.

Сервер приложений: встроенный Tomcat.

Tomcat — это популярный сервер приложений для Java, который поддерживает сервлеты и JSP. Встроенный Tomcat в Spring Boot позволяет легко развертывать приложение без необходимости дополнительной настройки сервера. Это упрощает процесс разработки и тестирования, так как разработчики могут запускать приложение локально с минимальными усилиями.

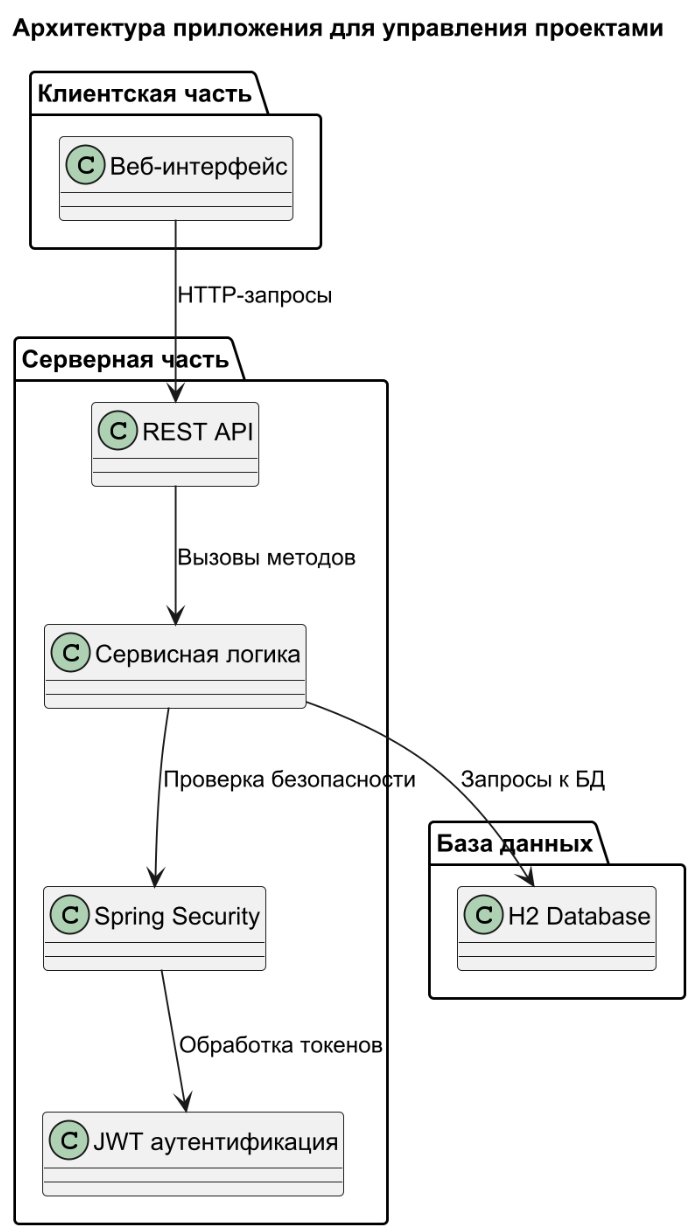


Рисунок 3. Архитектура приложения

2.2 Обоснование проектных решений по информационному обеспечению

Информационное обеспечение проекта играет ключевую роль в его успешной реализации и включает в себя несколько важных аспектов, касающихся хранения и обработки данных. В данном проекте выбраны следующие решения для обеспечения эффективного управления данными о пользователях, проектах и их связях:

Хранение данных о пользователях, проектах и их связях в реляционной базе данных:

Для хранения информации о пользователях и проектах выбрана реляционная база данных. Это решение обеспечивает структурированное хранение данных, позволяя легко управлять связями между различными сущностями. Реляционные базы данных, такие как H2, позволяют использовать SQL для выполнения запросов, что делает работу с данными более понятной и удобной. В рамках данного проекта реализованы таблицы для пользователей, проектов и связей между ними (например, таблица users\_projects), что позволяет эффективно управлять данными.

Структура данных разработана для обеспечения простоты запросов и поддержки:

Структура базы данных спроектирована с учетом удобства выполнения запросов и легкости в поддержке. Каждая таблица содержит необходимые поля для хранения информации, а связи между таблицами реализованы с использованием внешних ключей. Это позволяет выполнять сложные запросы с минимальными усилиями. Например, при необходимости получить все проекты, связанные с конкретным пользователем, можно легко выполнить соответствующий SQL-запрос.

Дополнительные аспекты информационного обеспечения

Использование DTO (Data Transfer Object):

Для передачи данных между клиентом и сервером используются классы DTO, что позволяет изолировать внутреннюю структуру модели от внешнего интерфейса. Это упрощает процесс разработки и тестирования, а также обеспечивает безопасность данных.

Обработка исключений:

В проекте реализован глобальный обработчик исключений, который позволяет централизованно управлять ошибками и исключениями. Это обеспечивает более стабильную работу приложения и улучшает пользовательский опыт.

Безопасность данных:

Для защиты данных пользователей применяется аутентификация с использованием JWT (JSON Web Tokens). Это решение обеспечивает безопасность доступа к ресурсам приложения и защиту от несанкционированного доступа.

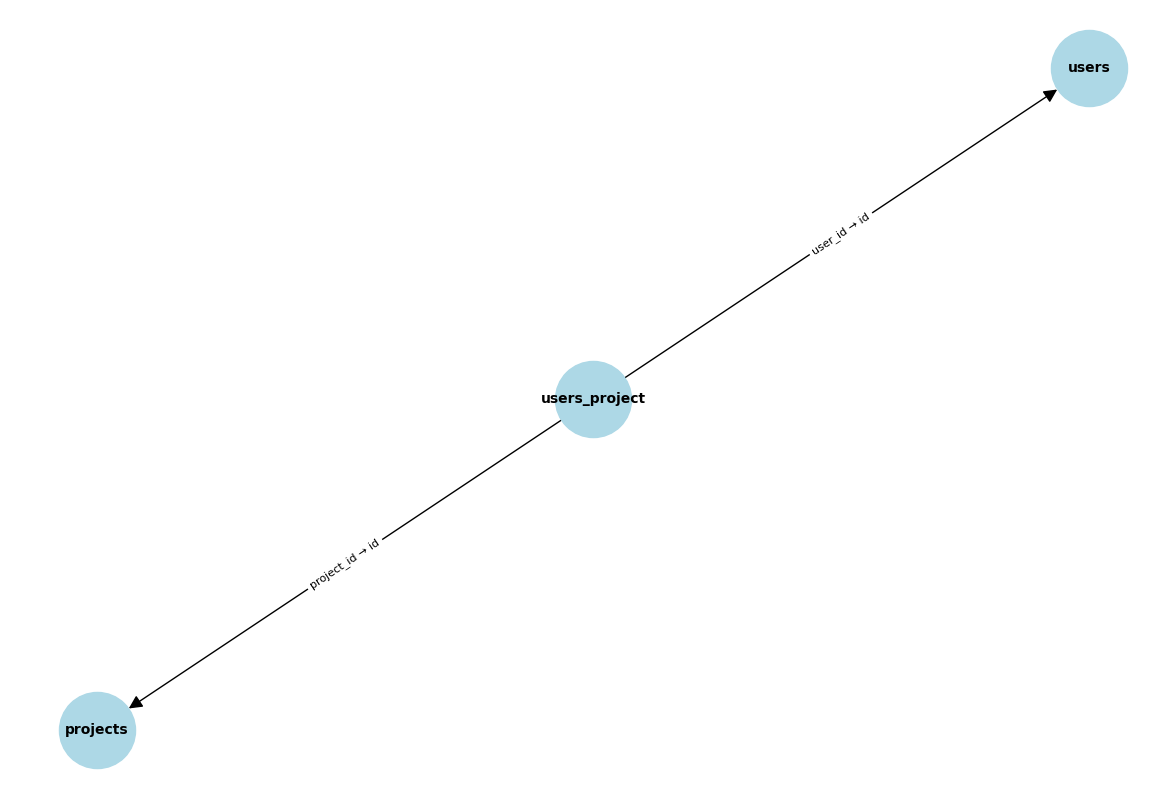


Рисунок 4. ER-диаграмма базы данных

Cтруктура базы данных включает следующие таблицы:

1. **users**:
   * id: Уникальный идентификатор пользователя.
   * user\_name: Имя пользователя.
   * password: Пароль.
   * email: Электронная почта.
   * role: Роль пользователя.
2. **projects**:
   * id: Уникальный идентификатор проекта.
   * name: Название проекта.
   * description: Описание проекта.
   * created\_date: Дата создания проекта.
3. **users\_project**:
   * id: Уникальный идентификатор связи.
   * related\_entity\_id: Идентификатор связанной сущности.
   * project\_id: Идентификатор проекта.
   * user\_id: Идентификатор пользователя.

Связи:

* + - project\_id ссылается на id из таблицы projects.
    - user\_id ссылается на id из таблицы users.

2.3 Обоснование проектных решений по программному обеспечению

Программное обеспечение, использованное в проекте, включает в себя следующие ключевые компоненты:

Фреймворк Spring для создания REST API:

Spring является мощным фреймворком для разработки Java-приложений, который предоставляет множество инструментов и библиотек для создания надежных и масштабируемых веб-приложений. В данном проекте Spring используется для разработки RESTful API, что позволяет клиентским приложениям взаимодействовать с сервером через стандартные HTTP-запросы. Это обеспечивает гибкость и возможность интеграции с различными клиентскими приложениями, такими как веб-интерфейсы и мобильные приложения.

Thymeleaf для серверного рендеринга страниц:

Thymeleaf — это современный серверный шаблонизатор, который позволяет создавать динамические веб-страницы. Он интегрируется с Spring и позволяет легко генерировать HTML-код на стороне сервера. Использование Thymeleaf упрощает создание пользовательского интерфейса, так как разработчики могут использовать стандартные HTML-теги и атрибуты, а также добавлять динамическое содержимое с помощью специальных атрибутов Thymeleaf.

JUnit и Mockito для тестирования:

Для обеспечения качества кода и проверки функциональности приложения используются JUnit и Mockito. JUnit — это популярный фреймворк для написания и выполнения тестов в Java, который позволяет разработчикам создавать юнит-тесты для проверки отдельных компонентов приложения. Mockito, в свою очередь, используется для создания мок-объектов и имитации поведения зависимостей, что упрощает тестирование сервисов и контроллеров. Эти инструменты позволяют проводить как юнит-тестирование, так и интеграционное тестирование, что способствует выявлению ошибок на ранних стадиях разработки.

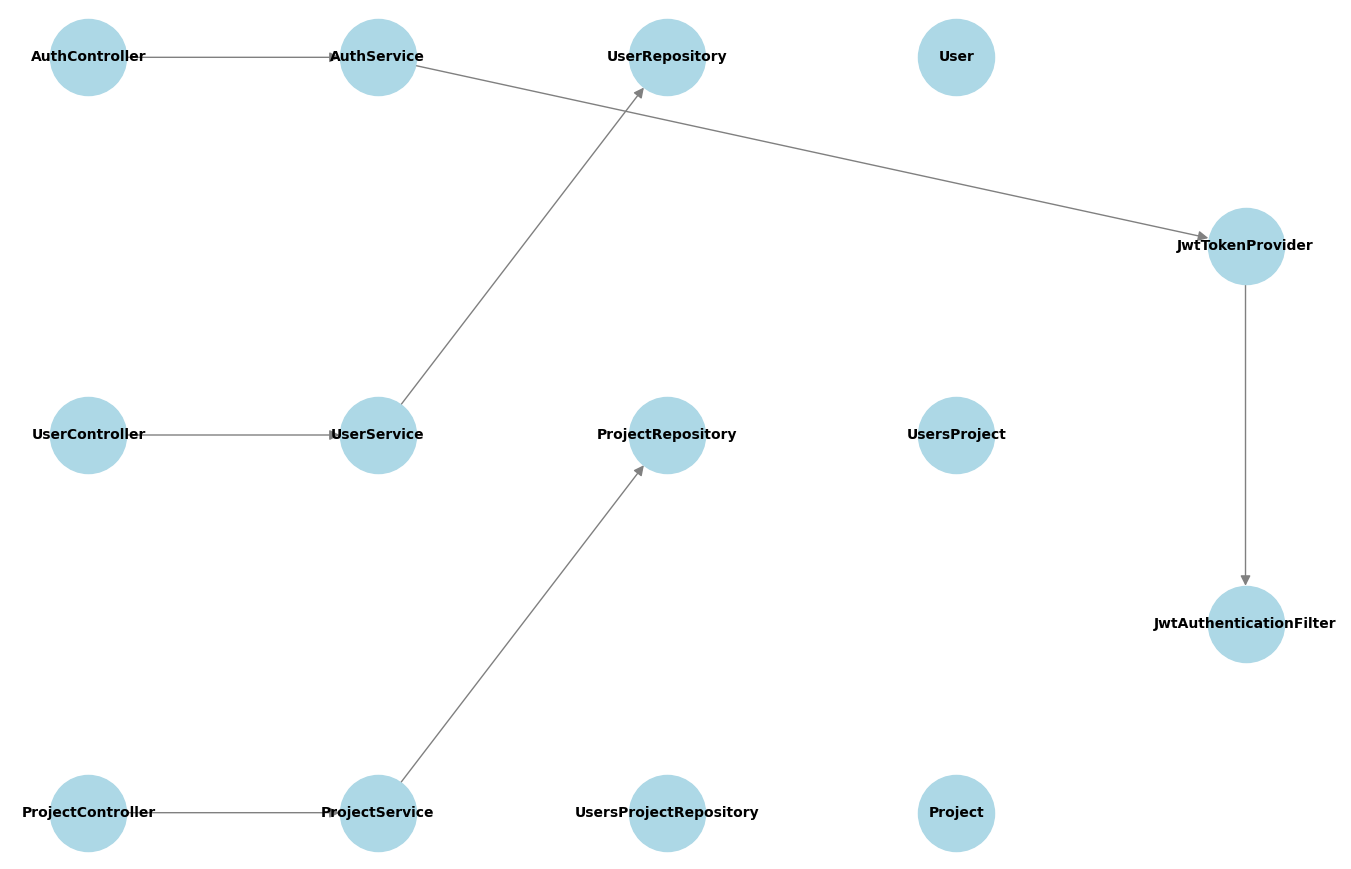


Рисунок 5. Логическая структура приложения

2.4 Обоснование выбора программных средств

Выбор программных средств для разработки веб-приложения был основан на их функциональности, популярности и удобстве использования. Основные компоненты, выбранные для реализации проекта, включают:

Фреймворк Spring:

Spring является одним из самых популярных фреймворков для разработки Java-приложений благодаря своему мощному функционалу и поддержке современных стандартов. Он предоставляет обширные возможности для создания RESTful API, управления зависимостями, интеграции с различными базами данных и обеспечения безопасности приложения. Использование Spring позволяет разработчикам сосредоточиться на бизнес-логике, минимизируя время на решение рутинных задач. Кроме того, Spring активно поддерживается сообществом и имеет множество готовых решений и библиотек, что значительно упрощает процесс разработки.

База данных H2:

Выбор H2 в качестве базы данных обусловлен её простотой интеграции и использования в учебных и тестовых целях. H2 — это легковесная реляционная база данных, которая может быть запущена в памяти или в виде файла. Это делает её идеальным выбором для разработки и тестирования, так как она не требует сложной настройки и позволяет быстро развернуть среду разработки. H2 поддерживает стандартный SQL, что упрощает миграцию к другим реляционным базам данных в будущем.

Дополнительные аспекты выбора программных средств

Thymeleaf:

Для серверного рендеринга страниц выбран Thymeleaf, который позволяет легко создавать динамические HTML-шаблоны. Он интегрируется с Spring и обеспечивает удобный способ генерации веб-страниц с использованием стандартных HTML-тегов.

JUnit и Mockito:

Для тестирования приложения используются JUnit и Mockito. JUnit предоставляет мощные инструменты для написания юнит-тестов, а Mockito позволяет создавать мок-объекты для имитации поведения зависимостей. Это сочетание обеспечивает высокое качество кода и надежность приложения.

# 3 Реализация задачи

3.1 Организация базы данных web-приложения

База данных содержит три основные таблицы: users, projects и users\_projects, связанные между собой отношением "многие ко многим".

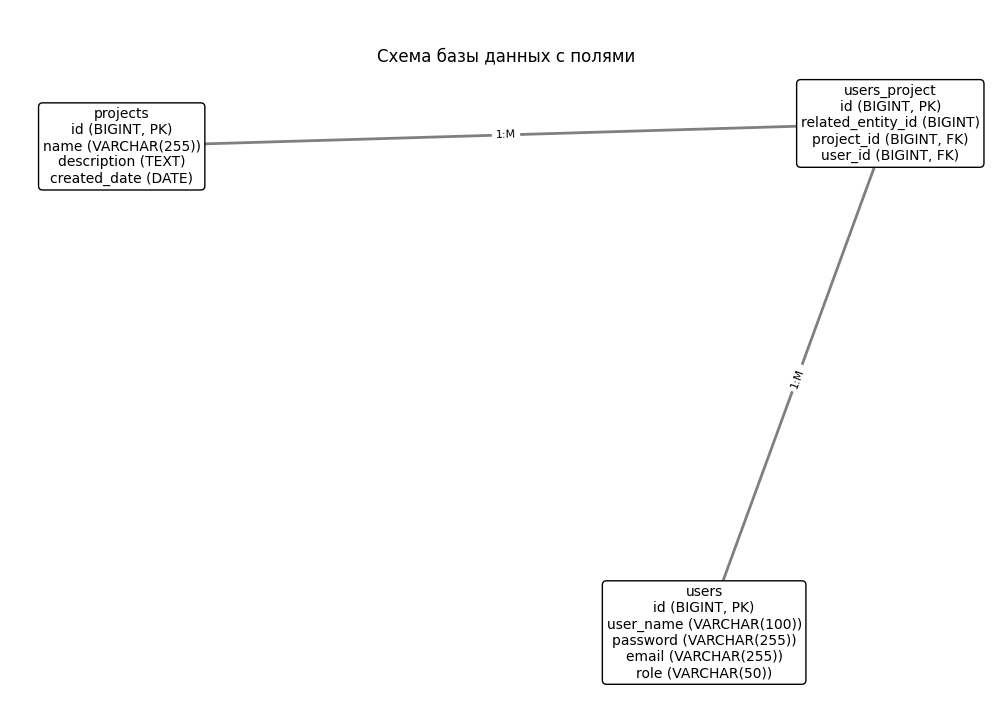


Рисунок 6. Таблицы базы данных

3.2 Принцип работы web-приложения

Веб-приложение, разработанное с использованием фреймворка Spring, реализует несколько ключевых функций, обеспечивающих эффективное управление пользователями и проектами. Основные функции приложения включают:

Аутентификация пользователей с использованием JWT:

Приложение использует JSON Web Tokens (JWT) для аутентификации пользователей. При успешной аутентификации пользователю выдается токен, который затем используется для доступа к защищенным ресурсам приложения. Токен хранится в куках и передается с каждым запросом, что обеспечивает безопасность и защиту от основных атак. У токена установлено время жизни, по истечение которого он становится недействительным

Управление пользователями и проектами через API и веб-интерфейс:

Приложение предоставляет RESTful API для выполнения операций создания, чтения, обновления и удаления (CRUD) как пользователей, так и проектов. Веб-интерфейс позволяет пользователям взаимодействовать с системой через графический интерфейс, что делает управление проектами более удобным.

Схема последовательности запросов

Для лучшего понимания принципа работы приложения представим схему последовательности запросов. Эта схема иллюстрирует процесс аутентификации пользователя и взаимодействия с API для управления проектами..

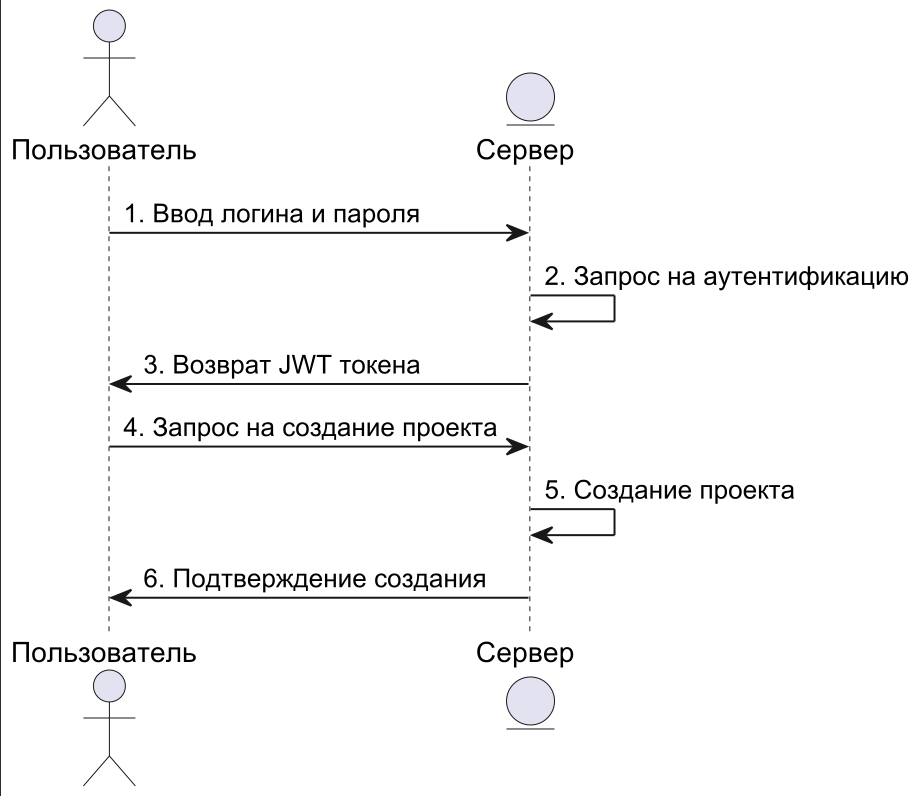


Рисунок 7. Схема последовательности запросов.

Описание схемы

* Пользователь вводит логин и пароль: Пользователь заполняет форму аутентификации на веб-странице.
* Запрос на аутентификацию: Веб-приложение отправляет данные на сервер для проверки учетных данных.
* Возврат JWT токена: Если учетные данные верны, сервер генерирует JWT токен и отправляет его пользователю.
* Запрос на создание проекта: Пользователь отправляет запрос на создание нового проекта, используя полученный токен для аутентификации.
* Создание проекта: Сервер обрабатывает запрос и создает новый проект в базе данных.
* Подтверждение создания: Сервер отправляет подтверждение о создании проекта обратно пользователю.

3.3 Файловая структура web-приложения

Файловая структура организована по пакетам, включая контроллеры, сервисы, репозитории и модели данных.

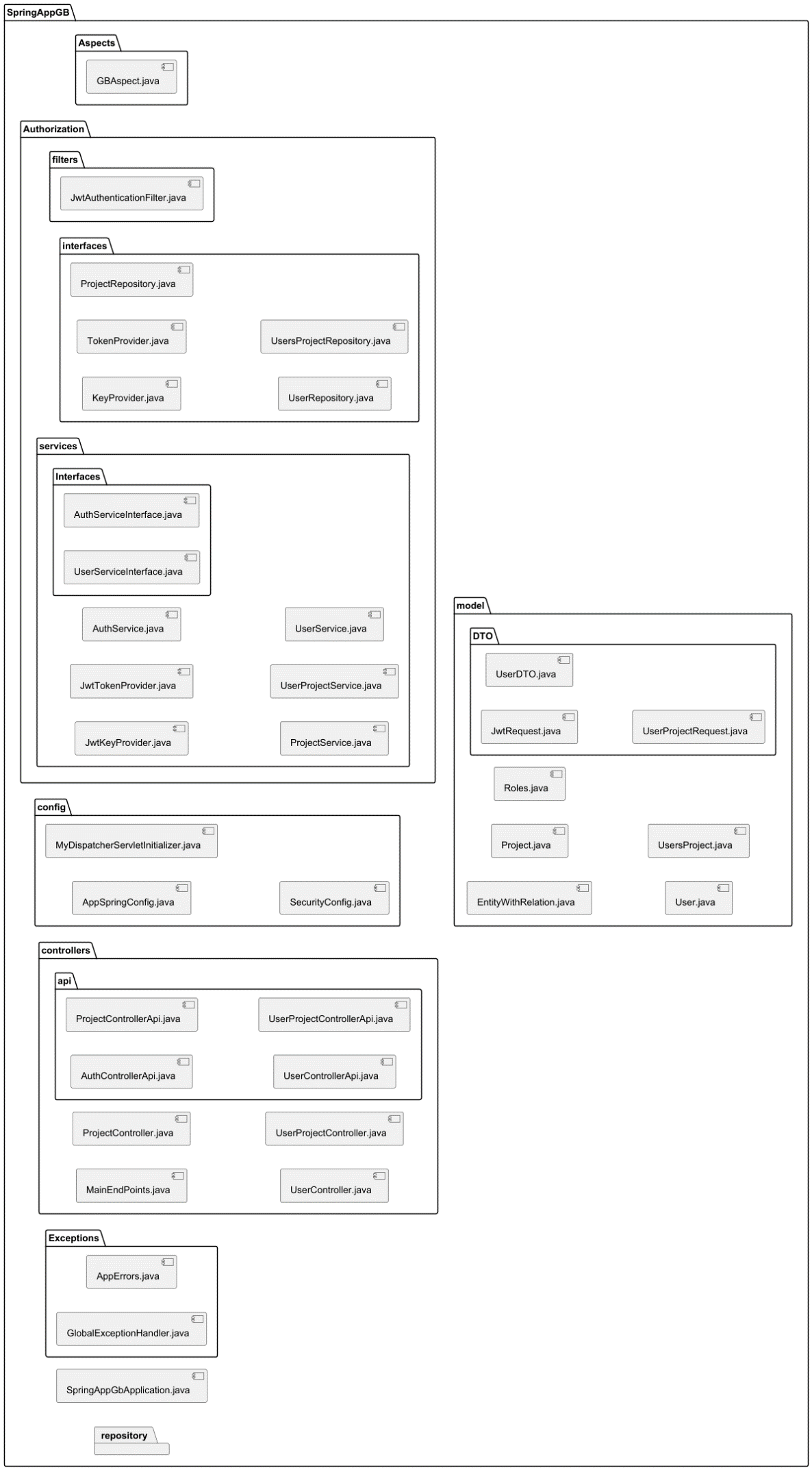


Рисунок 9. Дерево файлов проекта.

Описание структуры папок проекта

Структура папок проекта организована по принципу модульности и четкого разделения ответственности, что позволяет легко управлять кодом и поддерживать его. Ниже представлено детальное описание структуры папок веб-приложения на основе фреймворка Spring.

Корневая директория проекта

* **SpringAppGB/**: Корневая директория проекта, содержащая основные компоненты и настройки приложения.

Директория src/

* **src/**: Основная директория исходного кода приложения.
  + **main/**: Директория, содержащая основной код приложения.
    - **java/**: Директория с исходным кодом на Java.
      * **com/**: Корневой пакет для вашего приложения.
        + **example/**: Пакет для организации кода по примерам или модулям.

**SpringAppGB/**: Основной пакет приложения, содержащий все компоненты.

**Aspects/**: Пакет для аспектов (AOP), например, для логирования операций.

**Authorization/**: Пакет для аутентификации и авторизации пользователей.

**filters/**: Фильтры для обработки запросов и проверки JWT токенов.

**interfaces/**: Интерфейсы для сервисов аутентификации.

**services/**: Реализация логики аутентификации пользователей и генерации JWT токенов.

**config/**: Пакет конфигурации приложения Spring, включая настройки MVC и безопасность.

**controllers/**: Пакет контроллеров для обработки запросов от клиентов, включая API и веб-контроллеры.

**Exceptions/**: Пакет для обработки исключений в приложении.

**model/**: Пакет моделей данных (сущностей), включая классы DTO (Data Transfer Object) для передачи данных между клиентом и сервером.

**repository/**: Пакет репозиториев для работы с базой данных, реализующий паттерн Repository.

**services/**: Пакет сервисов бизнес-логики приложения, управляющий взаимодействием между контроллерами и репозиториями.

* + **resources/**: Директория ресурсов приложения, содержащая файлы конфигурации и шаблоны:
    - Конфигурационные файлы для настройки приложения и подключения к базе данных.
    - Статические ресурсы (CSS, JavaScript).
    - Шаблоны Thymeleaf, используемые в веб-интерфейсе.
* **test/**: Директория тестов приложения:
  + Содержит тесты на Java, включая юнит-тесты и интеграционные тесты для проверки функциональности сервисов.

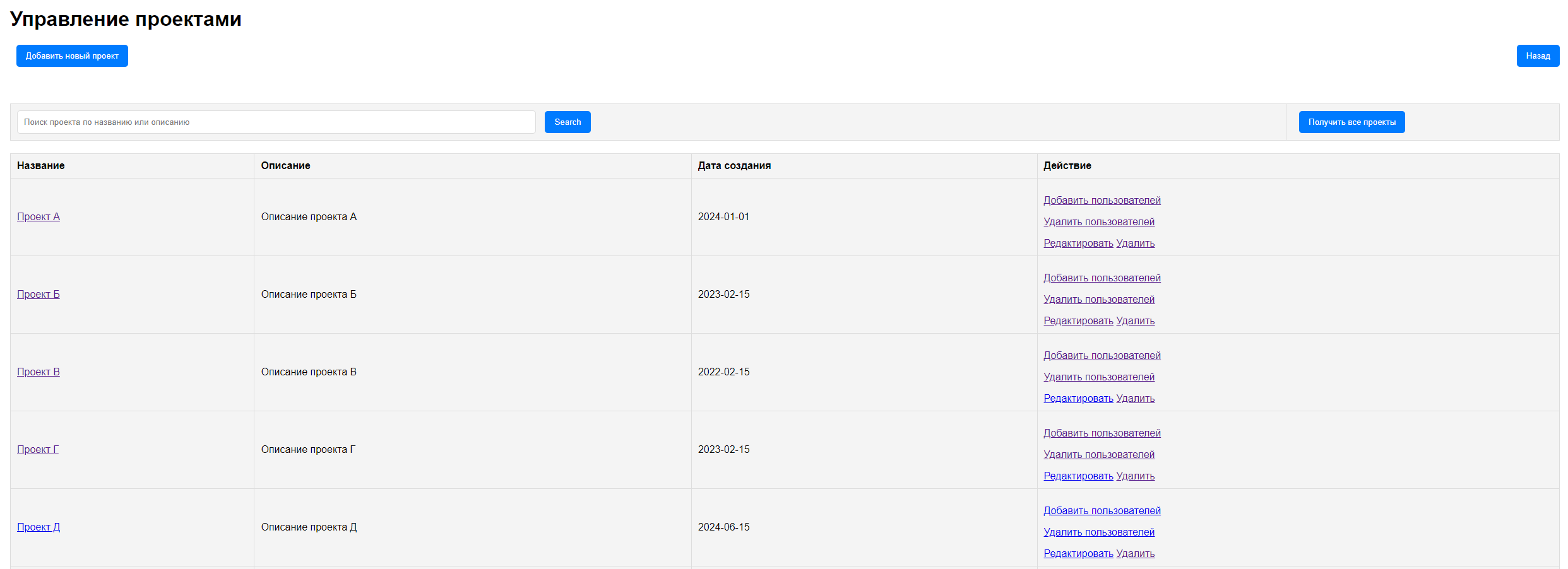
3.4 Демонстрация разработанного сайта

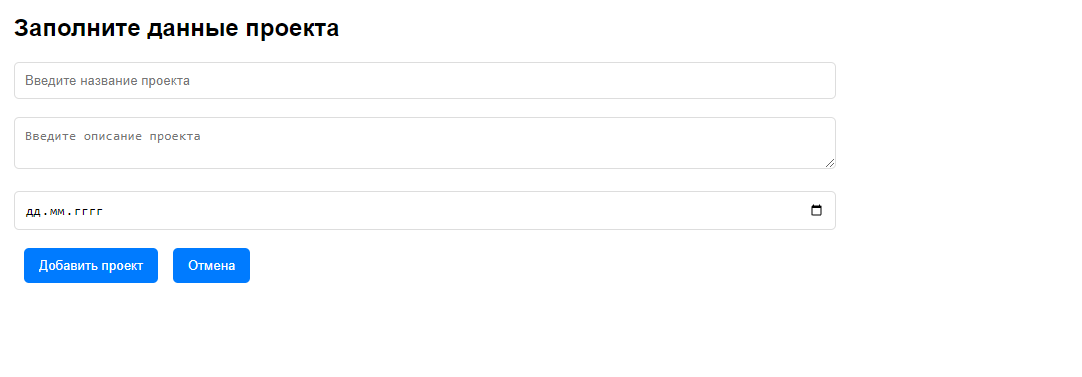
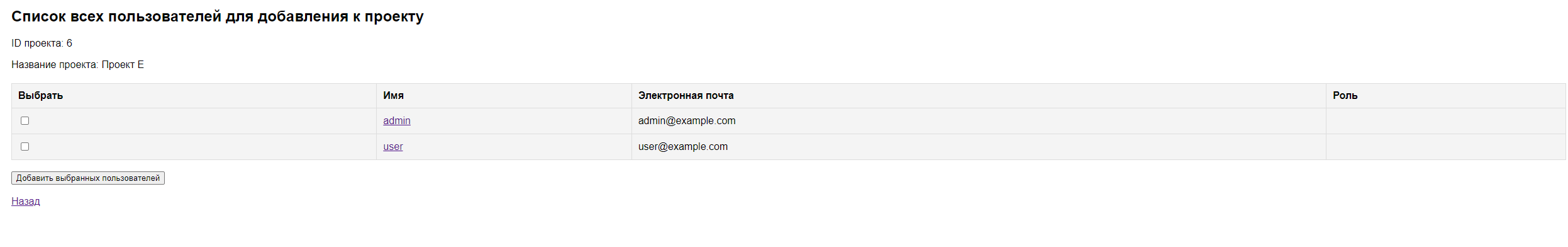
Разработанное веб-приложение предоставляет интуитивно понятный интерфейс для управления проектами и пользователями, что упрощает взаимодействие пользователей с системой. Основные функции приложения включают:

Аутентификация пользователей: Пользователи могут легко войти в систему, используя безопасный процесс аутентификации с поддержкой JWT.

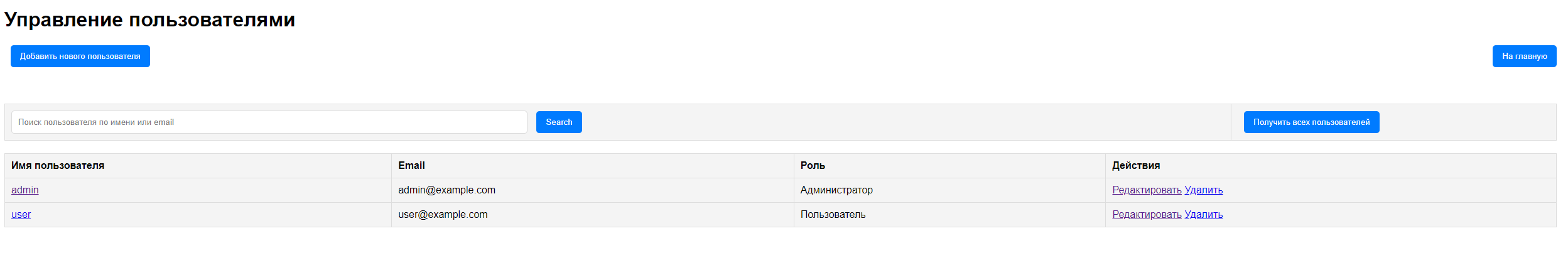


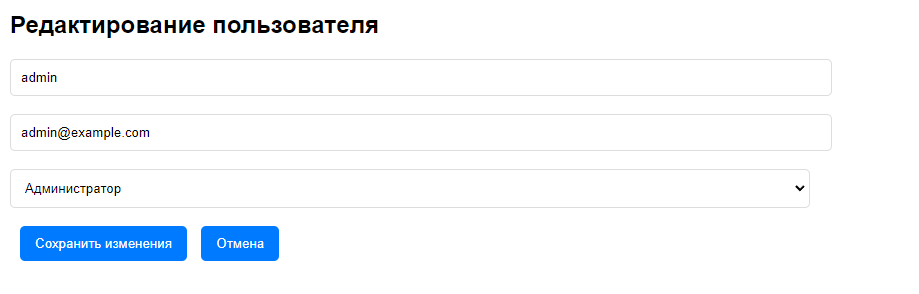
Управление проектами: Приложение позволяет создавать, редактировать, делать поиск и удалять проекты. Приложение позволяет просматривать список всех проектов и получать подробную информацию о каждом из них.



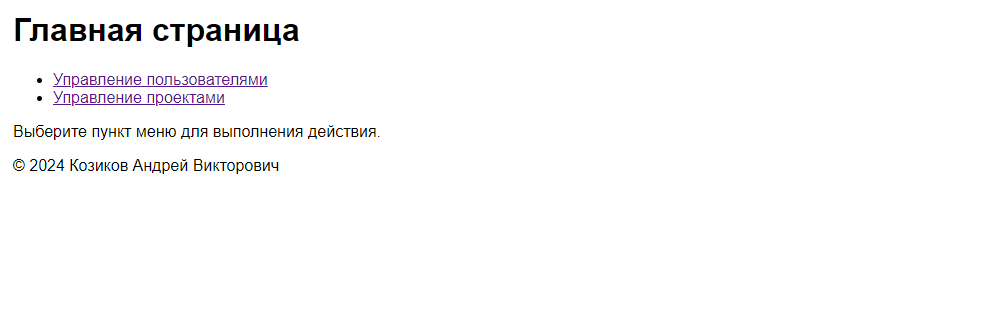


Управление пользователями: Администраторы могут управлять учетными записями пользователей, включая добавление новых пользователей, редактирование их данных, поиск и удаление учетных записей.





Интуитивно понятный интерфейс: Веб-интерфейс разработан с акцентом на удобство использования, что позволяет пользователям легко находить необходимые функции и выполнять действия без лишних усилий.

На главной странице приложения пользователи видят панель навигации, позволяющую быстро переходить между различными разделами, такими как управление проектами и пользователями.

# 4 Тестирование приложения

Тестирование — неотъемлемая часть разработки программного обеспечения, обеспечивающая высокое качество и надежность кода. В данном проекте реализовано два типа тестов: интеграционные и модульные. Оба типа тестов направлены на проверку корректности работы компонентов системы, их взаимодействия и обработки данных.

4.1 Интеграционные тесты

Интеграционные тесты направлены на проверку взаимодействия различных компонентов приложения и их взаимодействия с базой данных. В данном проекте был написан класс ProjectServiceIntegrationTest, который выполняет следующие задачи:

Проверка добавления нового проекта: Тест проверяет, что проект корректно сохраняется в базе данных и данные проекта возвращаются корректно после добавления.

Проверка получения всех проектов: Тест проверяет метод getAllProjects, убедившись, что он возвращает все проекты, сохраненные в базе данных.

Поиск проекта по ID: Тест проверяет, что метод findProjectById корректно возвращает проект по его идентификатору.

Поиск проектов по имени или описанию: Тест проверяет метод findProjectByNameOrDescription, который должен находить проекты по заданной строке в имени или описании.

Обновление проекта по ID: Тест проверяет метод updateProjectById, который обновляет данные существующего проекта в базе данных.

Удаление проекта по ID: Тест проверяет метод deleteProjectById, который удаляет проект из базы данных.

Класс ProjectServiceIntegrationTest использует аннотацию @SpringBootTest, что позволяет запускать тесты в контексте Spring, обеспечивая полное взаимодействие с репозиториями и сервисами приложения. Для тестов используются аннотации @Transactional, что позволяет выполнять тесты в транзакции, которая откатывается после каждого теста, обеспечивая чистое состояние базы данных.

4.2 Модульные тесты

Модульные тесты предназначены для проверки отдельных компонентов приложения без взаимодействия с реальной базой данных. В проекте реализован класс ProjectServiceUnitTest, который тестирует метод updateProjectById сервиса ProjectService.

Класс ProjectServiceUnitTest использует аннотацию @ExtendWith(MockitoExtension.class) для интеграции с библиотекой Mockito, которая используется для создания мок-объектов. В данном случае:

Мок-объект ProjectRepository используется для имитации взаимодействия с базой данных, что позволяет изолировать тестируемый метод от реального хранилища данных.

Метод updateProjectById проверяется на корректное обновление данных проекта. Метод given() используется для настройки возвращаемого значения при вызове findById(), а метод verify() проверяет вызовы репозитория.

# 5 Интеграция мониторинга с Prometheus и Grafana

В рамках данного проекта для управления проектами была реализована интеграция инструментов мониторинга Prometheus и Grafana. Эти мощные инструменты позволяют отслеживать состояние приложения, собирать метрики и визуализировать данные, что способствует повышению доступности и стабильности работы системы. В этом разделе мы рассмотрим, как Prometheus и Grafana используются в проекте, их ключевые особенности и преимущества.

Что такое Prometheus?

Prometheus — это система мониторинга и сбора метрик с открытым исходным кодом, предназначенная для надежного сбора и хранения временных рядов данных. В контексте нашего веб-приложения Prometheus используется для мониторинга различных аспектов системы, таких как производительность API, время отклика сервисов и состояние базы данных.

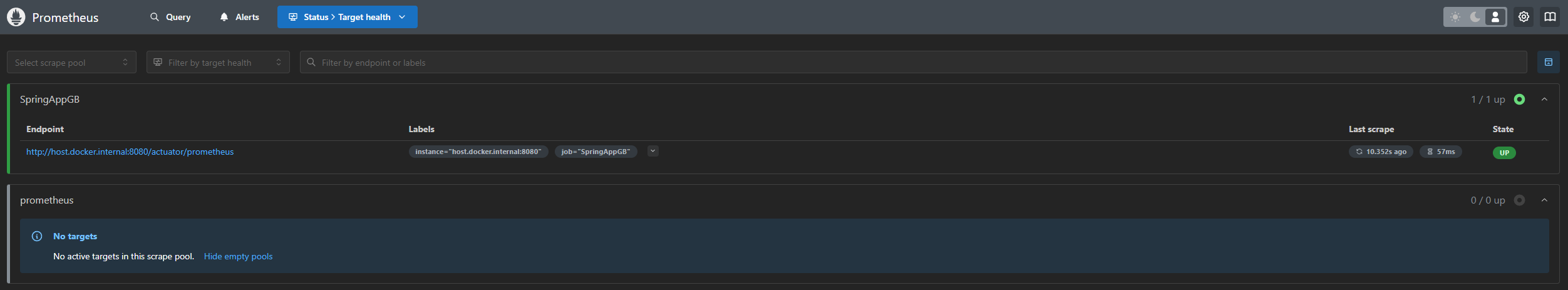
Ключевые особенности Prometheus:

Сбор данных: Prometheus собирает метрики с различных источников, используя метод pull (запрос метрик). В нашем проекте это позволяет получать данные о производительности приложения в реальном времени.

Язык запросов: PromQL — мощный язык запросов для анализа данных, который позволяет фильтровать и агрегировать метрики. Это дает возможность глубже анализировать работу приложения и выявлять узкие места.

Долговременное хранение данных: Prometheus может хранить метрики в своей базе данных с возможностью настройки хранения, что позволяет сохранять исторические данные для дальнейшего анализа.

Экспортеры: Для получения данных от приложений и систем используются экспортёры. В нашем проекте настроен Spring Actuator, который предоставляет необходимые метрики для мониторинга состояния приложения.



Что такое Grafana?

Grafana — это платформа с открытым исходным кодом для визуализации данных, которая позволяет создавать интерактивные панели мониторинга. В нашем проекте Grafana используется для отображения данных, собранных Prometheus, в виде графиков и диаграмм.

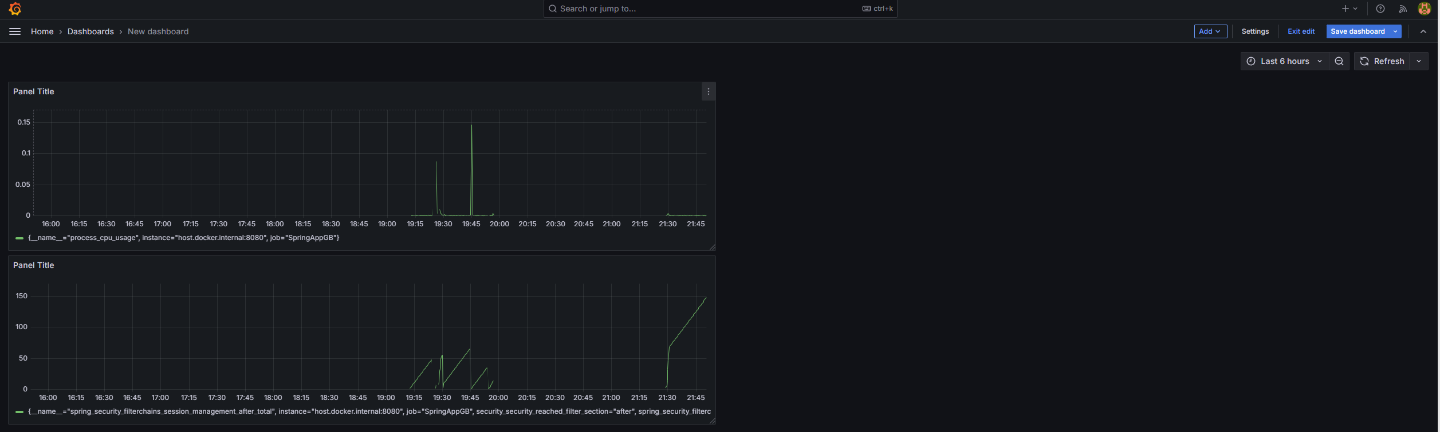
Ключевые особенности Grafana:

Гибкость в визуализации: Grafana поддерживает создание пользовательских панелей с множеством типов визуализаций, что позволяет адаптировать представление данных под конкретные нужды пользователей.

Интеграция с множеством источников данных: Grafana может подключаться к различным источникам, включая Prometheus, что делает её идеальным инструментом для визуализации собранных метрик.

Алерты и уведомления: Возможность создания алертов позволяет информировать пользователей о важных событиях в системе, таких как превышение пороговых значений производительности.

Плагины и расширения: Система плагинов позволяет добавлять новые функции и интеграции, что делает Grafana еще более мощным инструментом для мониторинга.



Интеграция Prometheus и Grafana

Prometheus и Grafana часто используются вместе для создания мощных систем мониторинга. В нашем проекте интеграция этих инструментов позволяет эффективно собирать и визуализировать метрики:

Сбор метрик: Prometheus собирает метрики из нашего веб-приложения через Spring Actuator.

Визуализация данных: Grafana подключается к Prometheus как источник данных и строит дашборды на основе собранной информации. Пользователи могут настраивать графики и диаграммы для отображения актуальной информации о состоянии системы.

Преимущества использования Grafana и Prometheus

Использование Grafana и Prometheus в нашем проекте предоставляет множество преимуществ:

Гибкость: Легкость настройки дашбордов под конкретные нужды команды.

Масштабируемость: Подходит для систем с большим объемом данных, обеспечивая эффективный сбор и анализ информации.

Обширные возможности визуализации: Grafana предлагает разнообразные форматы визуализации, что позволяет лучше понять производительность приложения.

Поддержка сообщества: Оба инструмента имеют активное сообщество разработчиков и обширную документацию, что упрощает обучение и интеграцию.

# 6 Документация с использованием Javadoc

В рамках разработки веб-приложения для управления проектами особое внимание уделено качеству кода и его документированию. Для этого использовалась система Javadoc, которая позволяет генерировать документацию на основе комментариев в коде. Javadoc является стандартным инструментом для документирования кода на Java, обеспечивая ясность и доступность информации о классах, методах и их назначении.

Основные аспекты использования Javadoc в проекте:

Стандарты документирования:

В проекте были установлены четкие стандарты для написания комментариев Javadoc. Каждый класс, метод и интерфейс снабжен описанием, которое включает информацию о его назначении, параметрах и возвращаемых значениях. Это упрощает понимание структуры приложения и помогает разработчикам быстрее ориентироваться в коде.

Генерация документации:

С помощью инструмента Javadoc была сгенерирована полная документация, которая включает все классы и методы проекта. Эта документация доступна в формате HTML, что делает её удобной для просмотра и использования как для текущих разработчиков, так и для новых участников команды.

Поддержка командной работы:

Наличие хорошо документированного кода облегчает совместную работу над проектом. Новые члены команды могут быстрее погрузиться в проект, а существующие разработчики могут легко находить нужную информацию без необходимости изучать каждый элемент кода.

Улучшение качества кода:

Регулярное использование Javadoc способствует более тщательному анализу кода во время его написания. Разработчики задумываются о том, как лучше описать функциональность своих методов и классов, что приводит к более чистому и понятному коду.

# 7 Использование Spring Boot Starter Validation

В рамках разработки веб-приложения для управления проектами была внедрена библиотека Spring Boot Starter Validation, которая обеспечивает поддержку валидации данных, поступающих от пользователей. Это решение позволяет гарантировать, что данные, отправляемые в приложение, соответствуют заданным критериям и правилам, что способствует повышению надежности и безопасности системы.

Основные аспекты использования Spring Boot Starter Validation:

Обеспечение целостности данных:

Валидация данных позволяет предотвратить ввод некорректной информации, такой как пустые поля, неверные форматы данных (например, неправильный адрес электронной почты или номер телефона) и другие ошибки, которые могут возникнуть при взаимодействии с пользователем. Это особенно важно для поддержания целостности данных в базе данных.

Аннотации для валидации:

Spring Boot Starter Validation предоставляет набор аннотаций, таких как @NotNull, @Size, @Email, @Min, @Max и другие, которые можно использовать для определения правил валидации непосредственно в моделях данных (DTO).

public class Project {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY) // Автоинкремент

private Long id;

@NotBlank(message = "The project name cannot be blank")

@Size(max = 255, message = "The name of the project should not exceed 255 characters")

@Column

private String name;

@Size(max = 500, message = "The project description should not exceed 500 characters")

@Column

private String description;

@PastOrPresent(message = "Incorrect project creation date")

@JsonFormat(shape = JsonFormat.Shape.STRING, pattern = "yyyy-MM-dd")

@Column(name="created\_date")

private LocalDate createdDate;

@OneToMany(mappedBy = "project", cascade = CascadeType.ALL, orphanRemoval = true)

@JsonIgnore

private List<UsersProject> usersProjects = new ArrayList<>();

}

Автоматическая обработка ошибок:

При использовании Spring Boot Starter Validation ошибки валидации автоматически обрабатываются и возвращаются пользователю в удобном формате. Это позволяет разработчикам сосредоточиться на бизнес-логике приложения, не тратя время на написание кода для обработки ошибок валидации.

Интеграция с контроллерами:

Валидация легко интегрируется с контроллерами Spring. При получении данных от пользователя можно использовать объект BindingResult для проверки наличия ошибок валидации:

public ResponseEntity<?> addUser(@Valid @RequestBody UserDTO userDTO, BindingResult result) {

if (result.hasErrors()) {

return ResponseEntity.badRequest().body(result.getAllErrors());

}

// Логика добавления пользователя

}

Улучшение пользовательского опыта:

Четкие сообщения об ошибках помогают пользователям быстрее исправить введенные данные и повторно отправить форму. Это улучшает общий пользовательский опыт и снижает вероятность возникновения проблем при работе с приложением.

# 8 Использование Spring AOP

В рамках разработки веб-приложения для управления проектами была внедрена технология Aspect-Oriented Programming (AOP) с использованием Spring AOP. Этот подход позволяет разделять кросс-функциональные аспекты приложения, такие как логирование, управление транзакциями и обработка ошибок, от основной бизнес-логики. Это способствует улучшению структуры кода и его читаемости.

Основные аспекты использования Spring AOP:

Логирование:

В проекте был создан класс GBAspect, в котором реализованы аспекты для логирования различных операций, таких как добавление, обновление и удаление проектов и пользователей. Это позволяет отслеживать действия в приложении и упрощает диагностику проблем.

Определение аспектов:

Аспекты определяются с помощью аннотаций, таких как @Aspect, что позволяет легко управлять поведением методов в зависимости от заданных условий. Например, можно настроить логирование для всех методов сервиса или только для определённых операций.

Управление транзакциями:

Spring AOP также используется для управления транзакциями в приложении. Это позволяет гарантировать, что операции с базой данных выполняются атомарно, что критически важно для поддержания целостности данных.

Интерфейсы для аспектов:

В проекте были созданы интерфейсы AuthInterface, AuthServiceInterface и UserServiceInterface, которые помогают определить контракты для сервисов и позволяют легко интегрировать аспекты в бизнес-логику.

Упрощение кода:

Использование AOP позволяет избежать дублирования кода при реализации кросс-функциональных задач. Вместо того чтобы добавлять логирование или обработку ошибок в каждый метод, разработчики могут сосредоточиться на бизнес-логике, а аспекты будут автоматически применяться.

# Заключение

В результате разработки веб-приложения для управления проектами был создан эффективный инструмент, который значительно упрощает взаимодействие между участниками команды и оптимизирует процессы управления проектами. Использование современных технологий, таких как Spring, H2, Thymeleaf, JUnit и Mockito, обеспечило высокую производительность, надежность и удобство в разработке и тестировании приложения.

Основные достижения проекта:

Эффективное управление проектами: Реализованы функции создания, редактирования и удаления проектов, а также управления пользователями и их связями с проектами. Это позволяет командам более эффективно организовывать свою работу и отслеживать прогресс выполнения задач.

Безопасность данных: Внедрение системы аутентификации с использованием JWT обеспечивает защиту данных пользователей и предотвращает несанкционированный доступ к приложению. Это критически важно для обеспечения конфиденциальности и безопасности информации.

Мониторинг и визуализация: Интеграция с Prometheus и Grafana позволяет отслеживать производительность приложения в реальном времени. Система мониторинга предоставляет возможность визуализировать ключевые метрики, что помогает быстро реагировать на возможные проблемы и улучшать качество обслуживания.

Тестирование и качество кода: Использование JUnit и Mockito для написания тестов гарантирует высокое качество кода и надежность приложения. Это позволяет выявлять ошибки на ранних стадиях разработки и обеспечивает уверенность в стабильной работе системы.

Интуитивно понятный интерфейс: Разработка пользовательского интерфейса с использованием Thymeleaf сделала приложение доступным для пользователей с различным уровнем технической подготовки. Это способствует более легкому освоению системы и повышает общую продуктивность команды.

Перспективы развития

Данный проект имеет значительный потенциал для дальнейшего развития. В будущем можно рассмотреть возможность интеграции дополнительных функций, таких как автоматизация процессов управления задачами, расширение возможностей отчетности или интеграция с другими системами управления.

В заключение, разработанное веб-приложение для управления проектами представляет собой эффективное решение для командной работы, которое сочетает в себе современные технологии, безопасность данных и удобство использования. Оно отвечает актуальным требованиям бизнеса и может быть адаптировано под различные сценарии использования в будущем.

# Список использованной литературы

Брюс Эккель. "Философия Java". — СПб.: Питер, 2016.

Годфри Бланшард. "Spring в действии". — М.: Вильямс, 2017.

Стивен К. Шварц. "Spring Boot: быстрое создание приложений на Java". — М.: БХВ-Петербург, 2018.

Кирилл Сидоров. "Java EE 8: Разработка корпоративных приложений". — М.: Вильямс, 2019.

Игорь Сысоев. "Java для профессионалов". — М.: ДМК Пресс, 2020.

Алексей Шевченко. "Тестирование программного обеспечения на Java". — М.: Питер, 2021.

Григорий Коваленко. "Thymeleaf в действии". — М.: БХВ-Петербург, 2020.

Дмитрий Шевченко. "Основы мониторинга приложений". — М.: Вильямс, 2022.