

**Разработка серверной части приложения на Java для обмена  
событиями между пользователями с возможностью создавать  
мероприятия и участвовать в них  
«EventHub: Планируй и Присоединяйся»**

IT-специалист:  
Программист Java  
Кондрашов Б.В.

## Содержание:

1. Введение	
1.1. Актуальность темы	3
1.2. Цель и задачи исследования	5
1.3. Обзор приложений	6
2. Основная часть	
2.1. Разработка бэкенд части приложения "EventHub" на Java	9
2.1.1. Изучение основных технологий веб-разработки на Java	9
2.1.2. Проектирование структуры приложения	22
2.1.3. Создание базы данных с использованием PostgreSQL	23
2.1.4. Реализация серверной части сайта с помощью Spring Boot	26
2.2. Тестирование разработанного приложения	36
2.3. Процесс создания микросервисов и использование Docker	39
3. Основы разработки Ui/Ux дизайна	
3.1 Теоретические основы разработки Ui/Ux дизайна	42
3.2. Дизайн приложения	44
4. Список литературы	49
5. Приложение А	50

## **1. Введение**

### **1.1. Актуальность темы**

Современный мир становится все более цифровым, а жизнь людей все больше зависит от информационных технологий. С развитием интернета и мобильных устройств все больше людей общаются и взаимодействуют онлайн, включая планирование и посещение различных мероприятий и событий. В связи с этим, создание удобных и функциональных платформ для обмена информацией о событиях и участия в них становится все более актуальным.

Приложения для социальной активности играют значительную роль в современном общении и участии в различных событиях. Они способствуют расширению социальных связей и формированию сообществ, позволяя пользователям находить себе интересные мероприятия и находить единомышленников для общения и совместного времяпровождения. Благодаря возможности быстро создавать события и приглашать друзей принять участие, стимулируют активность пользователей в планировании и участии в различных событиях, что способствует развитию общественной активности и формированию привычки участвовать в жизни сообщества.

Java обеспечивает высокую производительность и масштабируемость не только веб-приложений, но и бэкенд-систем, что особенно важно для создания приложений, которые обрабатывают большие объемы данных и высокий трафик. Благодаря возможности разработки сложных функциональных решений с использованием широкого набора инструментов и библиотек, разработчики могут создавать мощные приложения, способные эффективно обрабатывать информацию и предоставлять пользовательский опыт высокого уровня. Кроме того, Java известен своей безопасностью и надежностью, что делает его одним из лучших выборов для создания веб-приложений, особенно в области обмена конфиденциальной информацией и данных. По совокупности этих качеств Java остается одним из самых популярных языков программирования для веб-

разработки в настоящее время, предоставляя разработчикам мощный инструментарий для реализации разнообразных проектов. Java обеспечивает высокую производительность и масштабируемость не только веб-приложений, но и бэкенд-систем, что особенно важно для создания приложений, которые обрабатывают большие объемы данных и высокий трафик. Благодаря возможности разработки сложных функциональных решений с использованием широкого набора инструментов и библиотек, разработчики могут создавать мощные приложения, способные эффективно обрабатывать информацию и предоставлять пользовательский опыт высокого уровня. Кроме того, Java известен своей безопасностью и надежностью, что делает его одним из лучших выборов для создания веб-приложений, особенно в области обмена конфиденциальной информацией и данных. По совокупности этих качеств Java остается одним из самых популярных языков программирования для веб-разработки в настоящее время, предоставляя разработчикам мощный инструментарий для реализации разнообразных проектов.

Приложение "EventHub" предлагает удобный и интуитивно понятный интерфейс для создания и поиска различных мероприятий, таких как концерты, встречи, фестивали и т. д. Пользователи могут легко создавать собственные события, устанавливать дату, время, местоположение, описание и другие параметры, а также приглашать других пользователей принять участие. Платформа позволяет легко просматривать актуальные события, фильтровать их по интересам и местоположению, а также удобно подписываться на обновления и уведомления о новых мероприятиях. Благодаря использованию Java для бэкенд-разработки, приложение "EventHub" обеспечивает стабильную работу, высокую производительность и безопасность, что делает его привлекательным для широкого круга пользователей.

Приложение "EventHub", позволяющее пользователям легко создавать и принимать участие в событиях, востребовано и может принести значительную пользу сообществу. Именно поэтому разработка бэкенд-части такого

приложения на Java и исследование современных методов веб-разработки является важной задачей сегодняшнего дня.

## **1.2. Цель и задачи исследования**

Цель исследования заключается в разработке бэкенд части приложения "EventHub" на Java с использованием современных технологий веб-разработки и проверки его работоспособности, изучение и применение принципов микросервисной архитектуры, разработки REST API для взаимодействия с фронтенд частью приложения и использование базы данных для хранения информации о событиях.

### **Задачи исследования:**

1. Изучить основные концепции веб-разработки на Java и выбрать подходящие технологии для создания бэкенд части приложения.
2. Разработать структуру приложения и создать базу данных для хранения информации о событиях и пользователях.
3. Реализовать серверную часть приложения с использованием фреймворка Spring Boot и обеспечить интеграцию с базой данных.
4. Протестировать API приложения с помощью специальных инструментов для проверки корректности работы эндпоинтов.
5. Оптимизировать работу разработанного приложения и обеспечить его эффективное функционирование.
6. Документировать процесс разработки, полученные результаты и выводы, а также предложить перспективы развития и улучшения приложения.

**Инструменты:** Postman, PostgreSQL, Spring Boot, REST API, Maven, микросервисы, Docker, JPA, Hibernate, Lombok.

### 1.3. Обзор приложений

На данный момент существует много популярных приложений для создания и поиска различных мероприятий, рассмотрим некоторые из них.

1. Eventbrite: является одной из крупнейших онлайн-платформ для создания, поиска и продажи билетов на различные мероприятия, начиная от концертов и фестивалей до конференций и встреч. Пользователи могут легко создавать события, продавать билеты, управлять регистрацией и привлекать аудиторию через онлайн-платформу Eventbrite.

2. Meetup: платформа для организации и участия в мероприятиях на различные темы и интересы. Пользователи могут создавать группы по интересам, планировать встречи, обсуждать идеи и находить единомышленников. Meetup предлагает широкий спектр различных мероприятий, от спортивных событий до искусства и культуры.

3. Facebook Events: это функция в социальной сети Facebook, которая позволяет пользователям создавать события, приглашать друзей, делиться новостями о мероприятиях и покупать билеты. Пользователи могут легко находить разнообразные мероприятия, подписываться на обновления и добавлять их в свой календарь.

4. Eventful: платформа для поиска и продвижения мероприятий, где пользователи могут найти информацию о концертах, фестивалях, выставках, спортивных событиях и прочих развлекательных мероприятиях. Пользователи могут также создавать собственные события и делиться ими с другими участниками.

5. Bandsintown: приложение, специализирующееся на поиске и отслеживании концертов и выступлений музыкальных исполнителей. Пользователи могут подписаться на своих любимых артистов, получать уведомления о ближайших концертах и приобретать билеты на мероприятия.

6. Songkick: Songkick — это приложение для поиска концертов и выступлений музыкальных исполнителей, которое также предлагает

персонализированные рекомендации для пользователей на основе их музыкальных предпочтений. Пользователи могут отслеживать гастроли артистов, добавлять концерты в свой календарь и покупать билеты.

7. Eventzilla: онлайн-платформа для создания, управления и продажи билетов на мероприятия различного масштаба. Пользователи могут легко настраивать страницы событий, проводить регистрацию участников, получать отчеты и аналитику о продажах билетов.

8. Ticketmaster: один из крупнейших сервисов по продаже билетов на различные мероприятия, включая концерты, спортивные состязания, театральные постановки и другие развлекательные мероприятия. Пользователи могут приобретать билеты онлайн, выбирать места и получать информацию о предстоящих мероприятиях.

9. Goldstar: платформа, специализирующаяся на предложениях по дешевым билетам на различные развлекательные мероприятия, включая театральные постановки, концерты, спектакли и экскурсии. Пользователи могут находить скидки на билеты и эксклюзивные предложения от партнеров сервиса.

10. Fever: приложение, предлагающее персонализированные рекомендации по различным развлекательным событиям в городе, таким как концерты, выставки, рестораны, клубы и другие мероприятия. Пользователи могут отслеживать популярные мероприятия, получать рекомендации и бронировать билеты онлайн.

Однако не надо расстраиваться, несмотря на большое количество приложений для создания и поиска различных мероприятий все равно возникает необходимость в создании новых:

1. Уникальность целевой аудитории: Новое приложение может быть направлено на узкую целевую аудиторию, которая не удовлетворена существующими решениями.

2. Новые функциональные возможности: Новые приложения могут предлагать инновационные функции или сервисы, которые отсутствуют в существующих решениях.

3. Локальный рынок: Некоторые приложения могут быть ориентированы на конкретные географические регионы или культурные особенности, что делает их более привлекательными для пользователей из этих областей.

4. Улучшение пользовательского опыта: Новые приложения могут быть разработаны с учетом современных тенденций в дизайне и удобства использования, что делает их более привлекательными для пользователей.

5. Конкурентное преимущество: Создание нового приложения может помочь компании выделиться на рынке и получить конкурентное преимущество перед другими игроками.

Таким образом, несмотря на широкий выбор существующих приложений для создания и поиска мероприятий, всегда есть потребность в разработке новых решений, отвечающих уникальным потребностям пользователей и рынка.



## **2. Основная часть**

### **2.1. Разработка бэкенд части приложения "EventHub" на Java**

#### **2.1.1. Изучение основных технологий веб-разработки на Java**

**2.1.1.1 Spring Framework** - это один из самых популярных и влиятельных фреймворков Java, который предоставляет широкий спектр инструментов и функций для разработки веб-приложений. Он был создан Rod Johnson в 2003 году и с тех пор стал одним из основных стандартов в мире Java-разработки.

Основные особенности и преимущества Spring Framework:

1. Inversion of Control (IoC): Одной из ключевых концепций Spring является принцип Inversion of Control, который позволяет разработчикам создавать слабосвязанные компоненты и делегировать управление зависимостями контейнеру Spring. Это упрощает тестирование и повышает гибкость приложений.

2. Dependency Injection (DI): Spring Framework обеспечивает механизм Dependency Injection, который позволяет внедрять зависимости в объекты без явного создания или управления ими. Это уменьшает связанность компонентов и делает код более читаемым и поддерживаемым.

3. Spring MVC: Spring Framework предоставляет модуль Spring MVC для создания веб-приложений на основе шаблона Model-View-Controller. Spring MVC облегчает разделение бизнес-логики, представления и управления запросами от клиентов.

4. Spring Boot: Spring Boot - это модуль Spring Framework, который позволяет разработчикам быстро создавать самостоятельные, готовые к запуску приложения без необходимости настройки сложной конфигурации. Он автоматизирует множество рутинных задач и упрощает развертывание приложений.

5. Spring Data: Spring Framework предоставляет модуль Spring Data для работы с различными хранилищами данных, такими как реляционные базы данных, NoSQL базы данных и другие. Spring Data упрощает взаимодействие с базами данных и предоставляет удобные средства для работы с данными.

6. AOP (Aspect-Oriented Programming): Spring поддерживает принцип AOP, который позволяет разработчикам выносить общие аспекты функциональности приложения (например, логирование, транзакции) в отдельные модули и применять их к различным компонентам приложения.

7. Security: Spring Security - это модуль Spring Framework, который обеспечивает мощные средства для обеспечения безопасности веб-приложений. Он поддерживает аутентификацию, авторизацию, защиту от CSRF атак и другие функции безопасности.

Spring Framework является мощным инструментом для создания современных веб-приложений на Java, который облегчает разработку, поддержку и масштабирование приложений. Он имеет активное сообщество разработчиков, обширную документацию и множество интеграций с другими технологиями Java.

**2.1.1.2 Hibernate** - это популярный фреймворк для объектно-реляционного отображения (ORM) в Java, который облегчает взаимодействие приложений с базами данных. Hibernate позволяет разработчикам работать с объектами Java, а не напрямую с запросами SQL, что упрощает разработку, повышает производительность и уменьшает количество кода.

Основные особенности и преимущества Hibernate:

1. ORM Mapping: Hibernate предоставляет механизм отображения объектов Java на таблицы в базе данных и наоборот. С помощью аннотаций или файлов маппинга (XML) разработчики могут указать соответствие между классами Java и таблицами в базе данных, а также между полями объектов и столбцами таблиц.

2. Lazy Loading: Hibernate поддерживает ленивую загрузку данных, что означает, что данные из базы данных загружаются только тогда, когда они

действительно нужны. Это помогает уменьшить количество запросов к базе данных и повысить производительность приложения.

3. Caching: Hibernate предоставляет возможности кэширования данных, что позволяет ускорить доступ к данным и снизить нагрузку на базу данных. Разработчики могут настроить различные уровни кэширования для оптимизации производительности приложения.

4. Transaction Management: Hibernate обеспечивает управление транзакциями, что позволяет разработчикам безопасно выполнять операции с базой данных в рамках одной или нескольких транзакций. Это гарантирует целостность данных и обеспечивает надежность приложения.

5. Query Language: Hibernate предоставляет свой собственный язык запросов HQL (Hibernate Query Language), который аналогичен SQL, но работает с объектами Java, а не с таблицами базы данных. HQL позволяет выполнять сложные запросы к данным и упрощает работу с объектами.

6. Integration with Spring: Hibernate хорошо интегрируется с фреймворком Spring, что позволяет использовать их вместе для создания надежных и масштабируемых приложений. Spring обеспечивает управление транзакциями, безопасность и другие функции, которые дополняют возможности Hibernate.

7. Community Support: Hibernate имеет активное сообщество разработчиков, обширную документацию и множество ресурсов для обучения. Это делает использование Hibernate удобным и эффективным для разработчиков.

Hibernate является мощным инструментом для работы с базами данных в Java, который упрощает разработку приложений, повышает производительность и обеспечивает надежность работы с данными. Он широко используется в индустрии и является одним из основных инструментов ORM в мире Java-разработки.

**2.1.1.3 JavaServer Faces (JSF)** - это фреймворк для разработки веб-приложений на языке Java, который предоставляет компонентную модель для создания пользовательских интерфейсов. JSF основан на шаблоне проектирования Model-View-Controller (MVC) и предоставляет инструменты для

управления состоянием пользовательского интерфейса, обработки событий и взаимодействия с бизнес-логикой приложения.

Описание основных компонентов JavaServer Faces:

1. Компоненты: JSF предоставляет богатый набор готовых компонентов интерфейса, таких как кнопки, поля ввода, таблицы, панели и другие. Компоненты могут быть использованы для быстрой разработки пользовательского интерфейса без необходимости писать HTML-код вручную.

2. Управление состоянием: JSF автоматически управляет состоянием пользовательского интерфейса, что позволяет сохранять данные между запросами и обновлениями страниц. Это облегчает разработку сложных форм и динамических интерфейсов.

3. Обработка событий: JSF предоставляет механизм обработки событий, таких как нажатие кнопки или выбор элемента списка. Разработчики могут определять методы обработки событий в управляемых бинах (managed beans) и связывать их с компонентами интерфейса.

4. Управляемые бины (Managed Beans): Управляемые бины являются Java-объектами, которые используются для хранения данных и бизнес-логики приложения в контексте JSF. Они могут быть связаны с компонентами интерфейса для обмена данными и выполнения операций.

5. Выражения Unified EL: JSF использует Unified Expression Language (EL) для связывания данных между компонентами интерфейса и управляемыми бинами. EL позволяет обращаться к свойствам объектов, вызывать методы и выполнять операции в выражениях.

6. Жизненный цикл запроса: JSF определяет жизненный цикл запроса, который включает этапы обработки запроса, валидации данных, обновления модели, отображения представления и отправки ответа клиенту. Это обеспечивает стандартизированный способ работы с запросами и ответами.

7. Шаблоны и композиции: JSF поддерживает использование шаблонов (templates) и композиций (composite components), что позволяет создавать

повторно используемые части интерфейса и упрощает разработку масштабируемых приложений.

8. Интеграция с другими технологиями Java: JSF легко интегрируется с другими технологиями Java, такими как Java EE (Enterprise Edition), CDI (Contexts and Dependency Injection), JPA (Java Persistence API) и другими. Это позволяет создавать полноценные веб-приложения с использованием различных инструментов и технологий.

JavaServer Faces является мощным фреймворком для разработки веб-приложений на Java, который предоставляет стандартизированный подход к созданию пользовательских интерфейсов. Он упрощает разработку веб-приложений, обеспечивает высокую производительность и удобство использования для разработчиков.

**2.1.1.4 Spring Boot** - это фреймворк для разработки веб-приложений на языке Java, который упрощает процесс создания, конфигурирования и развертывания приложений. Spring Boot основан на фреймворке Spring и предоставляет ряд инструментов и функциональности для быстрой разработки микросервисов, веб-приложений и RESTful API.

Основные характеристики и возможности Spring Boot:

1. Упрощенная конфигурация: Spring Boot предлагает концепцию "опинионированной конфигурации", которая позволяет разработчикам создавать приложения с минимальным количеством настроек. Фреймворк автоматически определяет и конфигурирует многие аспекты приложения, что упрощает процесс разработки.

2. Встроенные зависимости: Spring Boot включает в себя большое количество встроенных зависимостей (starter dependencies), которые предоставляют готовые библиотеки и инструменты для работы с различными технологиями, такими как базы данных, веб-серверы, шаблонизаторы и многое другое. Это позволяет быстро настраивать и расширять функциональность приложения.

3. Встроенный веб-сервер: Spring Boot поставляется с встроенными веб-серверами, такими как Tomcat, Jetty или Undertow, что позволяет запускать приложение без необходимости установки дополнительных серверов приложений. Это упрощает развертывание и тестирование приложений.

4. Управление зависимостями: Spring Boot использует систему управления зависимостями Maven или Gradle для управления зависимостями и сборки проекта. Фреймворк автоматически обрабатывает зависимости и их версии, что упрощает управление зависимостями.

5. Автоматическая конфигурация: Spring Boot предлагает автоматическую конфигурацию, которая позволяет определить настройки приложения на основе классов конфигурации, аннотаций и свойств. Это позволяет быстро настраивать различные аспекты приложения без необходимости явного указания всех параметров.

6. Мониторинг и управление приложением: Spring Boot предоставляет инструменты для мониторинга и управления приложением, такие как Actuator, которые позволяют получать информацию о состоянии приложения, метриках производительности, журналах и других аспектах работы приложения.

7. Интеграция с другими технологиями: Spring Boot легко интегрируется с другими технологиями Java, такими как Spring Framework, Spring Data, Spring Security и другими. Это обеспечивает возможность использовать различные компоненты и инструменты для создания полноценных веб-приложений.

Spring Boot является популярным фреймворком для разработки веб-приложений на Java благодаря своей простоте использования, гибкости и мощным функциональностям. Он позволяет быстро создавать высокопроизводительные и масштабируемые приложения с минимальными усилиями разработчика.

**2.1.1.5 Apache Struts** - это открытый фреймворк для создания веб-приложений на языке Java. Он был разработан для облегчения разработки веб-приложений с применением шаблона проектирования Model-View-Controller

(MVC). Фреймворк предоставляет инструменты для разделения бизнес-логики, пользовательского интерфейса и управления потоком приложения.

Apache Struts обеспечивает следующие основные компоненты:

1. Контроллер (Controller): в Struts он представлен классом `ActionServlet`, который обрабатывает запросы от клиента и управляет потоком выполнения приложения.

2. Модель (Model): представляет собой бизнес-логику приложения, например, классы `JavaBeans`, сервисы и т. д.

3. Представление (View): обычно в Struts используется технология `JavaServer Pages (JSP)` для создания пользовательского интерфейса.

Apache Struts имеет множество полезных функций, включая валидацию данных, управление сессиями, поддержку многоязычности, обработку форм, обработку ошибок и многое другое. Фреймворк также облегчает интеграцию с другими технологиями `Java EE`.

Одной из основных особенностей Apache Struts является легкость расширения с помощью собственных компонентов и плагинов. Это позволяет разработчикам создавать мощные и гибкие веб-приложения, соответствующие требованиям их проекта.

Несмотря на то, что Apache Struts является популярным и мощным фреймворком, его использование сегодня снизилось в связи с появлением более современных альтернатив, таких как `Spring Framework` и `Spring MVC`. Тем не менее, Apache Struts остается важным инструментом для многих `Java`-разработчиков и предлагает надежное средство для быстрой и эффективной разработки веб-приложений.

**2.1.1.6 Apache Wicket** - это фреймворк для разработки веб-приложений на языке `Java`, который использует компонентный подход и шаблон проектирования `Model-View-Controller (MVC)`. Он позволяет разработчикам создавать веб-приложения, используя объектно-ориентированные принципы и повторное использование компонентов.

### Основные принципы Apache Wicket:

1. Компонентный подход: в Wicket вся веб-страница состоит из компонентов, которые могут быть повторно использованы, расширены и модифицированы. Каждый компонент является объектом Java, что облегчает работу с ним и повышает гибкость приложения.

2. MVC: фреймворк разделяет бизнес-логику (Model), пользовательский интерфейс (View) и управление потоком выполнения (Controller). Это позволяет сделать код более легко читаемым, тестируемым и поддерживаемым.

3. Statefulness: Wicket поддерживает состояние компонентов, что означает, что он автоматически управляет состоянием форм и других веб-элементов без необходимости использования скрытых полей или сеансов.

### Другие особенности Apache Wicket:

- Простота и чистота кода: Wicket позволяет разрабатывать веб-приложения, используя Java и HTML без специальных шаблонов или генерации кода.

- Встроенная валидация данных: Wicket предоставляет механизмы для проверки входных данных, что позволяет разработчикам обеспечить безопасность и целостность данных.

- Поддержка AJAX: фреймворк предоставляет возможность использовать AJAX для создания динамических и отзывчивых веб-приложений.

Apache Wicket имеет активное сообщество разработчиков, поэтому существует множество расширений, плагинов и интеграций с другими технологиями. Фреймворк остается популярным среди Java-разработчиков, которые ценят его гибкость, простоту и мощный набор функций для создания качественных веб-приложений.

**2.1.1.7 Apache Tapestry** - это фреймворк для разработки веб-приложений на языке Java, который предлагает компонентный подход к созданию веб-приложений и обеспечивает эффективную и гибкую разработку веб-проектов. Он основан на шаблоне проектирования Model-View-Controller (MVC) и



предоставляет мощные инструменты и возможности для разработки современных веб-приложений.

Основные черты Apache Tapestry:

1. Компонентный подход: в Tapestry веб-страницы строятся из множества маленьких компонентов, которые могут быть повторно использованы, настроены и объединены. Это позволяет разработчикам создавать модульные и легко поддерживаемые веб-приложения.

2. Отсутствие XML-конфигураций: в отличие от некоторых других фреймворков, Apache Tapestry не требует большого количества XML-конфигураций, благодаря чему разработка веб-приложений становится более простой и прозрачной.

3. Полная интеграция с Java: Tapestry ориентирован на использование возможностей языка Java, что делает разработку веб-приложений более естественной для Java-разработчиков.

4. Поддержка AJAX: фреймворк обеспечивает поддержку AJAX из коробки, что позволяет создавать динамические и интерактивные веб-приложения с использованием асинхронного обмена данными.

5. Встроенная валидация данных: Tapestry предоставляет инструменты для валидации данных на стороне сервера и на стороне клиента, обеспечивая безопасность и целостность информации.

Apache Tapestry также предлагает множество расширений, библиотек и интеграций со сторонними технологиями, что делает его мощным инструментом для создания разнообразных веб-приложений. Фреймворк имеет свое сообщество разработчиков, которые активно поддерживают его и расширяют его функциональность.

**2.1.1.8 Java API for RESTful Web Services (JAX-RS)** - это стандартная спецификация Java, которая облегчает создание веб-сервисов в стиле REST (Representational State Transfer) на языке Java. JAX-RS предоставляет разработчикам инструменты и возможности для создания надежных,

масштабируемых и гибких веб-сервисов, которые могут быть использованы для обмена данными между различными системами и приложениями.

Основные особенности JAX-RS:

1. Аннотированные классы и методы: для создания веб-сервисов в JAX-RS используются аннотации, которые позволяют определить пути URL, тип HTTP-метода (GET, POST, PUT, DELETE и др.) и другие характеристики сервиса.

2. Разбор и генерация данных: JAX-RS предоставляет интеграцию с различными форматами данных, такими как JSON и XML, и позволяет легко преобразовывать объекты Java в эти форматы для обмена информацией с клиентскими приложениями.

3. Клиентские и серверные компоненты: JAX-RS поддерживает как создание RESTful веб-сервисов (серверная сторона), так и разработку клиентских приложений, которые могут общаться с этими сервисами.

4. Интеграция с другими Java EE стандартами: JAX-RS хорошо интегрируется с другими стандартами Java Enterprise Edition (Java EE), такими как Servlet API, EJB (Enterprise JavaBeans) и CDI (Contexts and Dependency Injection), обеспечивая единое и согласованное решение для разработки веб-приложений.

5. Поддержка асинхронности: JAX-RS предоставляет возможность создания асинхронных HTTP-запросов и ответов, что позволяет улучшить производительность веб-сервисов при работе с большим объемом данных и запросов.

JAX-RS является широко используемым и популярным стандартом для разработки RESTful веб-сервисов на языке Java. Он обеспечивает удобные и эффективные средства для реализации REST-интерфейсов и упрощает процесс создания веб-сервисов, что делает его предпочтительным выбором для многих разработчиков в данной области.

**2.1.1.9 Thymeleaf** - это Java-шаблонизатор, который используется для создания динамических веб-приложений. Thymeleaf обеспечивает интеграцию с Java-фреймворками, такими как Spring Framework, и позволяет разработчикам

создавать веб-приложения с использованием HTML шаблонов, в которых можно встраивать динамические данные и логику.

Основные особенности Thymeleaf:

1. Естественный синтаксис: Thymeleaf использует естественный синтаксис, который делает шаблоны легкими для чтения и понимания. Обычный HTML-код можно использовать без изменений, добавляя специальные атрибуты, начинающиеся с "th:", для встраивания динамических данных.

2. Интеграция с Java: Thymeleaf хорошо интегрируется с Java-кодом, что позволяет передавать данные из Java-контроллеров в HTML-шаблоны. Данные могут передаваться как модели, коллекции, объекты и т.д.

3. Поддержка шаблонизации: Thymeleaf поддерживает создание шаблонов, которые могут использоваться повторно для отображения контента на разных страницах приложения. Это облегчает разработку, поддержку и изменение внешнего вида приложения.

4. Визуализация: Thymeleaf обладает мощными средствами для визуализации данных, такими как итерации, условия, выражения и т.д. Это позволяет разработчикам создавать интерактивные и динамические веб-приложения с использованием шаблонов Thymeleaf.

5. Поддержка международизации: Thymeleaf обеспечивает поддержку международизации, позволяя локализовать веб-приложения на разные языки и культуры с помощью файлов свойств и специальных атрибутов.

Thymeleaf является популярным инструментом для создания динамических веб-приложений на платформе Java. Он обладает простым и интуитивно понятным синтаксисом, хорошо интегрируется с Java-фреймворками и предоставляет разработчикам мощные средства для создания красивых и функциональных веб-интерфейсов.

**2.1.1.10 Apache CXF** — это открытая библиотека для разработки и обеспечения веб-сервисов на основе Java. CXF поддерживает различные стандарты веб-сервисов, такие как SOAP (Simple Object Access Protocol), REST

(Representational State Transfer), WSDL (Web Services Description Language) и другие.

Ключевые особенности Apache CXF:

1. Поддержка различных стандартов: CXF обеспечивает поддержку различных стандартов веб-сервисов, что делает его универсальным инструментом для разработки как SOAP, так и RESTful веб-сервисов.

2. Интеграция с другими фреймворками: Apache CXF легко интегрируется с другими инструментами и фреймворками Java, такими как Spring Framework, что обеспечивает удобство использования и расширяемость.

3. Гибкость и расширяемость: CXF предоставляет множество расширяемых точек, что позволяет разработчикам легко настраивать поведение и функциональность веб-сервисов в соответствии с их требованиями.

4. Богатая функциональность: Библиотека предоставляет широкий набор возможностей, таких как поддержка различных протоколов безопасности (например, WS-Security), обработка ошибок, логирование запросов и многое другое.

5. Простота в использовании: Apache CXF предоставляет простой API для создания и настройки веб-сервисов, что позволяет быстро создавать и развертывать сервисы.

Использование Apache CXF помогает разработчикам быстро создавать надежные и мощные веб-сервисы на платформе Java, что делает его одним из популярных инструментов в области веб-разработки.

**Вывод** - перечисленные технологии представляют собой различные инструменты и фреймворки, используемые в разработке приложений на Java. Каждая из них имеет свои уникальные особенности и предназначена для решения определенных задач. Например, Spring Framework обеспечивает инверсию управления и поддержку тестирования, Hibernate упрощает взаимодействие с базами данных, JSF упрощает создание пользовательских интерфейсов, а JAX-RS позволяет создавать RESTful веб-сервисы.

Apache Struts, Apache Wicket и Apache Tapestry предлагают фреймворки для создания веб-приложений на Java с различными подходами к разработке. Spring Boot ускоряет процесс создания автономных приложений с минимальной конфигурацией. Thymeleaf облегчает работу с шаблонами веб-страниц, а Apache CXF предоставляет библиотеку для создания и обеспечения веб-сервисов.

В целом, эти технологии являются популярными инструментами в мире Java-разработки и предлагают широкий набор функциональности для создания современных приложений. Выбор конкретной технологии зависит от требований проекта, опыта разработчиков и особенностей задачи, которую необходимо решить.

В разрабатываемом приложении использованы: **Postman** - это инструмент для тестирования API, который позволяет разработчикам создавать, тестировать и документировать API запросы. Postman предоставляет удобный и интуитивно понятный интерфейс для взаимодействия с API. **PostgreSQL** - мощная система управления базами данных с открытым исходным кодом. Она известна своей надежностью, расширенными возможностями и поддержкой стандартов ANSI SQL. PostgreSQL широко используется в различных типах приложений и веб-сервисов. **Spring Boot** - это фреймворк для разработки приложений на языке Java. Он упрощает создание и развертывание Java-приложений, предоставляя набор инструментов и шаблонов для быстрой разработки. Spring Boot также обеспечивает интеграцию с другими проектами Spring Framework. **REST API** (Representational State Transfer) - это архитектурный стиль для создания веб-сервисов. REST API основан на стандартных методах HTTP и позволяет взаимодействовать с сервером посредством унифицированных запросов. **Maven** - инструмент для автоматизации сборки и управления проектами на Java. Он обеспечивает эффективное управление зависимостями, сборку проекта и управление жизненным циклом приложения. **Микросервисы** - это подход к разработке программного обеспечения, при котором приложение разбивается на небольшие и независимые службы. Каждый микросервис выполняет отдельную функцию и взаимодействует с другими микросервисами через API. **Docker** - это

платформа для контейнеризации приложений. Она позволяет упаковывать приложения и их зависимости в контейнеры, что облегчает развертывание, масштабирование и управление приложениями. **JPA** (Java Persistence API): JPA - стандартный интерфейс для работы с реляционными данными в приложениях на Java. Он предоставляет возможность управлять объектами Java и их отображением на таблицы баз данных. **Hibernate** - это фреймворк для объектно-реляционного отображения в Java. Он облегчает взаимодействие с базами данных, позволяя разработчикам работать с объектами Java, не беспокоясь о деталях связей с БД. **Lombok** - это библиотека для упрощения разработки на Java. Она позволяет генерировать методы доступа, конструкторы и другие стандартные структуры кода автоматически, улучшая производительность разработчика.

### 2.1.2. Проектирование структуры приложения

Проектирование структуры приложения - это процесс определения архитектуры и организации компонентов приложения для обеспечения его эффективной работы. В рамках этого процесса необходимо определить основные модули и функциональные блоки приложения, их взаимосвязи и взаимодействие, а также спецификацию данных, используемых приложением.

При проектировании структуры приложения важно учитывать требования к производительности, масштабируемости, безопасности и удобству использования. Также необходимо учитывать возможность будущего расширения и изменения функционала приложения.

В процессе проектирования структуры приложения часто используются различные методы и инструменты, такие как диаграммы UML, диаграммы потоков данных, диаграммы классов и другие. Эти инструменты помогают визуализировать структуру приложения и облегчают понимание его работы и взаимосвязей между компонентами.

При проектировании структуры приложения также важно учитывать принципы SOLID (Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, Dependency Inversion), которые помогают создавать гибкие, расширяемые и поддерживаемые приложения. Также полезно применять шаблоны проектирования, такие как MVC (Model-View-Controller), MVVM (Model-View-ViewModel), и другие, чтобы облегчить разделение ответственностей и улучшить переиспользуемость кода.

При проектировании структуры приложения важно также учитывать требования к хранению и обработке данных, выбор технологий и инструментов разработки, а также организацию работы команды разработчиков.

В целом, проектирование структуры приложения - это комплексный процесс, требующий внимательного анализа требований, планирования и принятия решений на различных этапах разработки.

### **2.1.3. Создание базы данных с использованием PostgreSQL**

PostgreSQL — самая популярная система управления базами данных в мире. Об этом говорят не только результаты опросов разработчиков и предпринимателей, но и количество вакансий, в которых владение ею указывается среди необходимых навыков.

PostgreSQL — это свободно распространяемая объектно-реляционная система управления базами данных (СУБД) с открытым исходным кодом, написанном на языке C. «Объектно-реляционная» означает, что PostgreSQL поддерживает концепции, присущие как реляционным базам данных, так и объектно-ориентированным языкам программирования (объекты, классы, наследование и другие).

В своих продуктах PostgreSQL используют такие IT-гиганты, как Huawei, Alibaba, «Инстаграм»\* и Yahoo. Согласно исследованию Stack Overflow, в котором приняло участие 48 тысяч профессиональных разработчиков, в 2022 году PostgreSQL была самой популярной СУБД.



Рисунок – Опрос разработчиков исследователями Stack Overflow

Достоинства базы данных:

1. Расширяемость и богатый набор типов данных. Помимо стандартных, в PostgreSQL есть типы для геометрических расчётов, сетевых адресов и полнотекстового поиска. Мощная система расширений позволяет добавлять новые возможности и типы данных. Кроме того, администратор может писать свои функции и процедуры на Python, PHP, Java, Ruby и многих других языках программирования, а также загружать модули на языке C из центрального репозитория PGXN.

2. Масштабируемость. Для повышения производительности и масштабируемости в PostgreSQL используются разные виды блокировок на уровне таблиц и строк; шесть видов индексов, среди которых B-дерево и обобщённое дерево поиска (GiST) для полнотекстового поиска; наследование



таблиц — для быстрого создания таблиц на основе имеющейся структуры. Вы также можете анализировать скорость выполнения запросов с помощью команды `explain`, очищать диск от мусора командой `vacuum` и собирать статистику по таблицам с `analyze`.

3. Кросс-платформенность. PostgreSQL поддерживается всеми популярными операционными системами, среди которых различные дистрибутивы Linux и BSD, macOS, Windows, Solaris и другие. Интерфейсы для этой СУБД реализованы практически во всех языках программирования.

4. Безопасность. В PostgreSQL есть множество инструментов для защиты данных от злоумышленников: пароль, Kerberos, LDAP, GSSAPI, SSPI, PAM и другие. Она позволяет управлять доступом к объектам БД на нескольких уровнях — от базы данных до отдельных столбцов, а также шифровать данные на аппаратном уровне.

5. Возможности NoSQL. Помимо стандартных форматов, PostgreSQL поддерживает XML, а с девятой версии — JSON и JSONB. Последний позволяет не разбирать JSON-документ перед записью в базу данных, что существенно ускоряет его сохранение в БД.

6. Свободное распространение и открытый код. Проект распространяется под лицензией BSD, что позволяет бесплатно его использовать, модифицировать и распространять. Исходный код можно посмотреть на зеркале официального Git-репозитория.

7. Живое сообщество и исчерпывающее официальное руководство. PostgreSQL поддерживается большим активным сообществом пользователей и разработчиков, которые получают новости о проекте через еженедельные рассылки, обсуждают вопросы администрирования и ведут дискуссии. Официальная документация регулярно обновляется и на момент публикации содержит около 3000 страниц подробных и понятных инструкций.

Недостатки базы данных:

1. Сложность настройки. Наверняка вы уже оценили обилие возможностей, которые даёт Postgres. Очевидно, что оно влечёт за собой разнообразие

конфигураций, что может создавать сложности для начинающих пользователей. Настройка базы данных требует глубокого понимания архитектуры и параметров.

2. Высокое потребление ресурсов. PostgreSQL может потреблять больше ресурсов (памяти и процессорного времени) по сравнению с некоторыми другими СУБД. Особенно это заметно при работе с большими объёмами данных и сложными запросами.

3. Отсутствие некоторых функций. В сравнении с некоторыми коммерческими СУБД PostgreSQL может слегка отставать в функциональности.

Недостатки в основном относятся к конкретным сценариям использования. В целом PostgreSQL остаётся одной из наиболее мощных и широко используемых открытых СУБД.принятия решений на различных этапах разработки.

#### 2.1.4. Реализация серверной части сайта с помощью Spring Boot

Приложение разработано на самом распространённом и востребованном Java-фреймворке – Spring Boot.

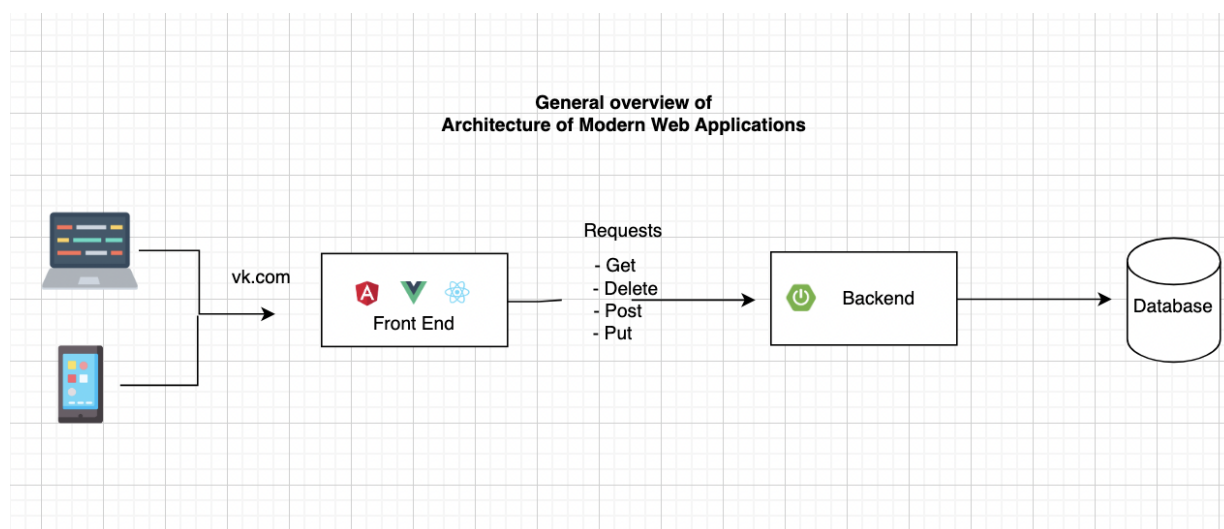


Рисунок - Архитектура современных web-приложений

Архитектура современных приложений состоит из отдельных модулей, как показано на рисунке выше. Эти модули часто называют Frontend и Backend. Frontend – это модуль, который отвечает за юзер-интерфейс и логику, которые предоставляется приложением при использовании. При заходе в соцсети через браузер взаимодействие происходит с FrontEnd-модулем приложения. То, как отображаются наши посты в виде сторисов или карточек, сообщения и другие активности реализуются именно в FrontEnd-модуле. А все данные, которые мы видим, хранятся и обрабатываются в Backend или серверной части приложения. Эти модули обмениваются между собой посредством разных архитектурных стилей: REST, GRPC и форматов сообщений – JSON и XML.

Рассмотрим примитивную серверную часть приложения с использованием Spring Boot, запустим свой сервер, рассмотрим разные типы HTTP запросов и их применение.

Для создания Spring Boot проекта, перейдем на страницу <https://start.spring.io/> и выберем необходимые зависимости: в нашем случае Spring Web. Чтобы запустить проект, необходима минимальная версия Java 17. Скачиваем проект и открываем в любом IDE (например – IntelliJ Idea)

**Project**

☐ Gradle - Groovy ☒ Gradle - Kotlin ☒ Maven

**Language**

☒ Java ☐ Kotlin ☐ Groovy

**Spring Boot**

☐ 3.0.2 (SNAPSHOT) ☒ 3.0.1 ☐ 2.7.8 (SNAPSHOT) ☐ 2.7.7

**Project Metadata**

Group

Artifact

Name

Description

Package name

**Packaging**

☒ Jar ☐ War

**Java**

☐ 19 ☒ 17 ☐ 11 ☐ 8

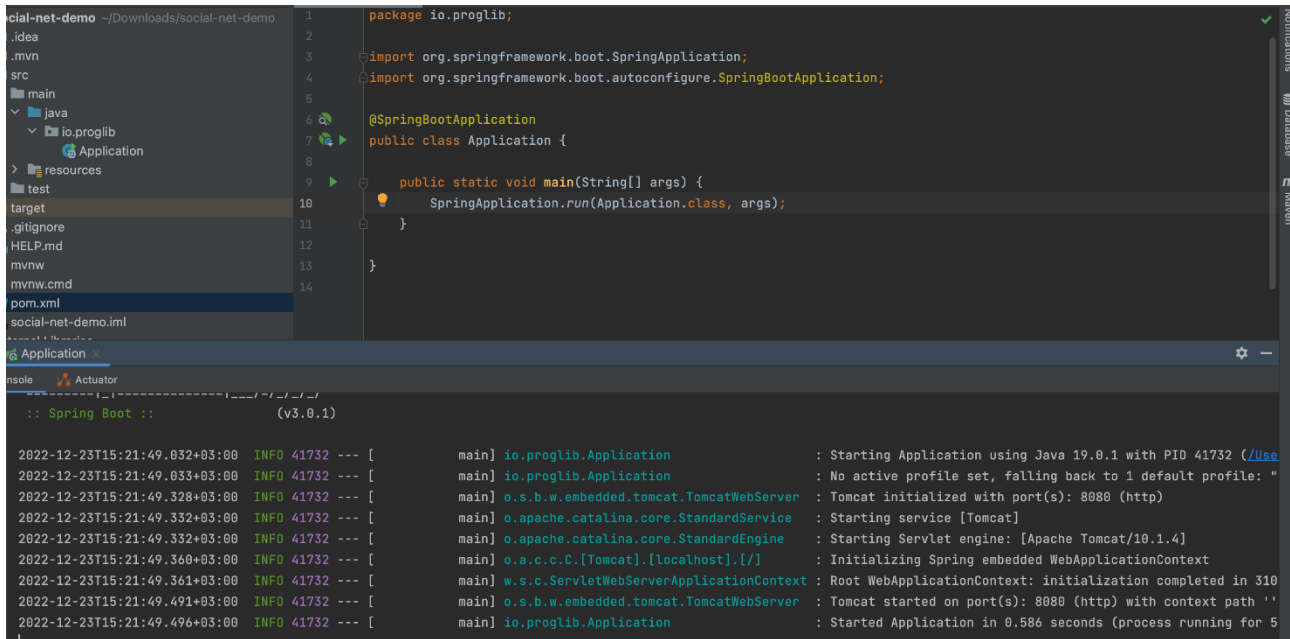
**Dependencies**

**Spring Web** WEB

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Рисунок - Инициализация проекта

Spring Web – зависимость, которая предоставляет контейнер сервлетов Apache Tomcat (является дефолтным веб-сервером). Проще говоря, сервлеты – это классы, которые обрабатывают все входящие запросы. Открываем проект и запускаем.



The screenshot shows an IDE with a project named 'social-net-demo'. The source code in the editor is as follows:

```
1 package io.proglib;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class Application {
8
9     public static void main(String[] args) {
10         SpringApplication.run(Application.class, args);
11     }
12
13 }
14
```

The console output at the bottom shows the application starting successfully on port 8080:

```
2022-12-23T15:21:49.032+03:00 INFO 41732 --- [main] io.proglib.Application : Starting Application using Java 19.0.1 with PID 41732 (/Use
2022-12-23T15:21:49.033+03:00 INFO 41732 --- [main] io.proglib.Application : No active profile set, falling back to 1 default profile: "
2022-12-23T15:21:49.328+03:00 INFO 41732 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2022-12-23T15:21:49.332+03:00 INFO 41732 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2022-12-23T15:21:49.332+03:00 INFO 41732 --- [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.4]
2022-12-23T15:21:49.360+03:00 INFO 41732 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2022-12-23T15:21:49.361+03:00 INFO 41732 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 310
2022-12-23T15:21:49.491+03:00 INFO 41732 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2022-12-23T15:21:49.496+03:00 INFO 41732 --- [main] io.proglib.Application : Started Application in 0.586 seconds (process running for 5
```

Рисунок - Запуск проекта

Проект запустился и готов обрабатывать запросы на порту 8080 – Tomcat started on port(s): 8080 (http). Создадим класс – GreetingController. Controller-классы ответственны за обработку входящих запросов и возвращают ответ.

```
package io.proglib;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/greet")
public class GreetingController {

    @GetMapping
    public String greet() {
        return "Hello";
    }
}
```

Рисунок - Greeting Controller

Чтобы сделать класс Controller, достаточно прописать аннотацию `@RestController`. `@RequestMapping` указывает, по какому пути будет находиться определённый ресурс или выполняться логика.

После перезапуска сервер готов уже обрабатывать запросы. Открываем браузер по адресу `http://localhost:8080/greet` и получаем следующий вывод.

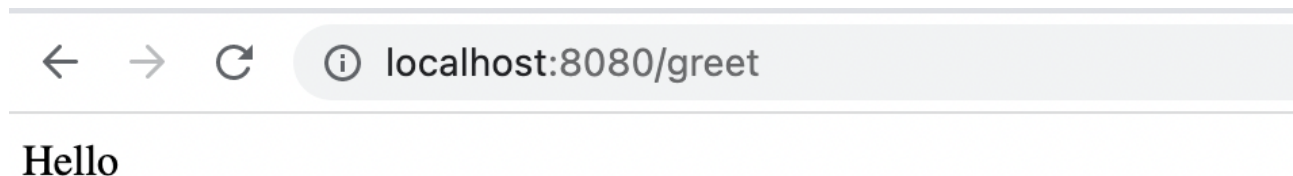


Рисунок - Ответ сервера

При отправке запроса по адресу `http://localhost:8080/`, мы получим ошибку, т. к. по этому пути не определены логика обработки запроса и ресурсы. При отправке запросов часто используются переменные в запросе, чтобы передавать дополнительную информацию или же делать запросы гибкими. Параметр в запросе передаётся в конце адреса (=url) сервера и указывается после вопросительного знака (=?). Например, `http://localhost:8080/greet?name=Alice`. Параметр запроса является `name` со значением `Alice`.

Для обработки переменной запроса, используется аннотация `@RequestParam`. Параметры запроса могут быть опциональными или же обязательными. `@RequestParam("name")` означает следующее: взять ту переменную из запроса, название которого равно `name`.

Запрос может содержать несколько параметров. Например, `http://localhost:8080/greet/full?name=John&surname=Smith`. Параметры выделяются знаком `&`. В этом запросе два параметра: `name=John` и `surname=Smith`. Чтобы обработать каждый параметр запроса, нужно пометить каждую переменную `@RequestParam`.

```

@RestController
@RequestMapping("/greet")
public class GreetingController {

    @GetMapping
    public String greet(@RequestParam("name") String name) {
        return "Hello, " + name;
    }
}

```

Рисунок - Метод с параметризованным запросом

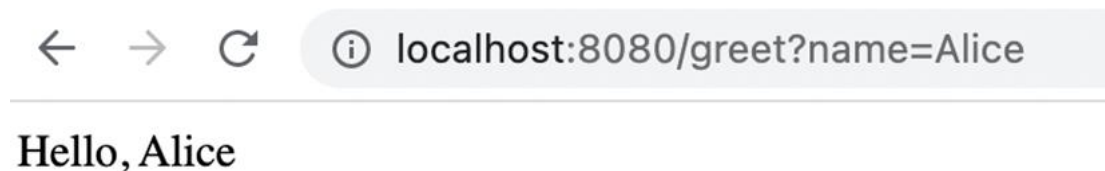


Рисунок - Ответ сервера

```

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/greet")
public class GreetingController {

    @GetMapping
    public String greet(@RequestParam("name") String name) {
        return "Hello, " + name;
    }

    @GetMapping("/full")
    public String fullGreeting(@RequestParam("name") String name,
                               @RequestParam("surname") String surname) {
        return "Nice to meet you, " + name + " " + surname;
    }
}

```

Рисунок - Параметризованный запрос с двумя параметрами



Nice to meet you, John Smith

Рисунок - Ответ сервера

PathVariable по применению похож на @RequestParam. @PathVariable также является параметром запроса, но используются внутри адреса запроса.

Например,

RequestParam – <http://localhost:8080/greet/full?name=John&surname=Smith>

PathVariable – <http://localhost:8080/greet/John>.

В этом случае John является PathVariable. В запросе можно указывать несколько PathVariable, как и в случае RequestParam

```
package io.proglib;

import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/greet")
public class GreetingController {

    @GetMapping
    public String greet(@RequestParam("name") String name) {
        return "Hello, " + name;
    }

    @GetMapping("/full")
    public String fullGreeting(@RequestParam("name") String name, @RequestParam("surname") String surname) {
        return "Nice to meet you, " + name + " " + surname;
    }

    @GetMapping("/{name}")
    public String greetWithPathVariable(@PathVariable("name") String name) {
        return "Hello, " + name;
    }
}
```

Рисунок - Запрос с двумя параметризованными PathVariable.

Для тестирования, открываем браузер и переходим по адресам:

<http://localhost:8080/greet/John/Smith>

<http://localhost:8080/greet/John>

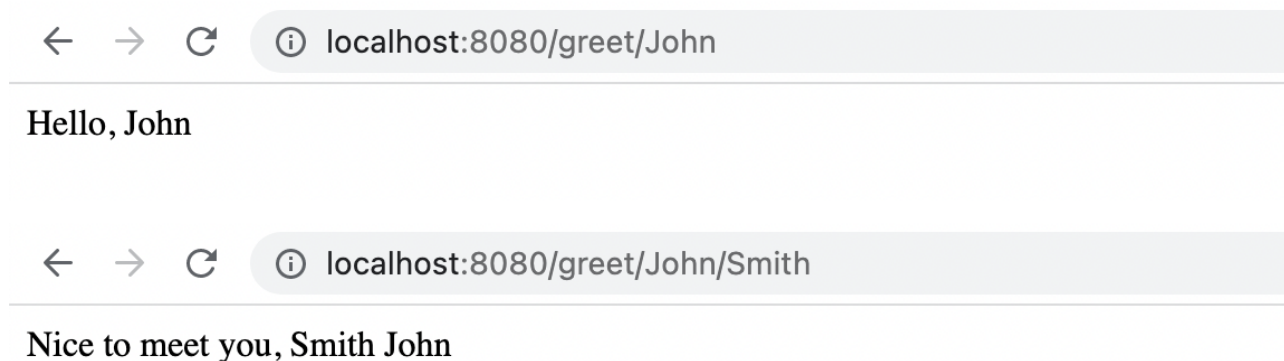


Рисунок - Ответы сервера

Каждый запрос представляет собой HTTP-метод. При переходе в браузере по адресу <http://localhost:8080/greet/John/Smith>, браузер отправляет GET-запрос на сервер. Большая часть информационных систем обмениваются данными посредством HTTP-методов. Основными HTTP-методами являются – POST, GET, PUT, DELETE. Эти четыре запроса также называют CRUD-запросами.

1. POST-метод – используется при создании новых ресурсов или данных. Например, когда мы загружаем новые посты в соцсетях, чаще всего используется POST-запросы. POST-запрос может иметь тело запроса.

2 GET-метод – используется при получении данных. Например, при открытии любого веб-приложения, отправляется именно GET-запрос для получения данных и отображения их на странице. GET-запрос не имеет тела запроса.

3 PUT-метод – используется для обновления данных, а также может иметь тело запроса, как и POST.

4 DELETE-метод – используется для удаления данных.



Чтобы сервер принимал все четыре запроса создадим сущности User и Post, и будем проводить операции над ними. Для простоты User имеет только два поля: username и список постов posts, а сущность Post имеет поле description и imageUrl.

```
package io.proglib;

import java.util.ArrayList;
import java.util.List;

public class User {
    private String username;
    private List<Post> posts;

    public User() {
        posts = new ArrayList<>();
    }

    public User(String username, List<Post> posts) {
        this.username = username;
        this.posts = posts == null ? new ArrayList<>() : posts;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public List<Post> getPosts() {
        return posts;
    }

    public void setPosts(List<Post> posts) {
        this.posts = posts;
    }
}
```

Рисунок - Сущность User

```
package io.proglib;

public record Post(
    String description,
    String imageUrl
) {
}
```

Рисунок - Сущность Post

Создадим новый класс контроллер – `UserActivityController`, который будет обрабатывать наши запросы – POST, GET, PUT, DELETE. Список – `List<User> users` используем в качестве локальной базы данных, где будем хранить все наши данные.

```
package io.proglib;

import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/users")
public class UserActivityController {

    private final static List<User> users = new ArrayList<>();

}
```

Рисунок - `UserActivityController`

Чтобы указать, что метод принимает POST-запросы используем аннотацию – `@PostMapping`. Так как запрос имеет тело запроса, где мы передаем пользователя, нужно пометить переменную `user` аннотацией `@RequestBody`.

```

@PostMapping("")
public User addUser(@RequestBody User user) {
    users.add(user);
    return user;
}

```

Рисунок - POST-запрос

```

@GetMapping("")
public List<User> getUsers() {
    return users;
}

@GetMapping("/{username}")
public User getUserByUsername(@PathVariable("username") String username) {
    return users.stream().filter(user -> user.getUsername().equals(username))
        .findFirst().get();
}

```

Рисунок – Get-запрос

@GetMapping("") указывает, что методы обрабатывают GET-запросы. Значение, которое передаётся внутри аннотации, является частью url или адреса. Запрос `http://localhost:8080/users/baggio` обработается методом `getUserByUsername()`, а запрос `http://localhost:8080/users/` обработается методом <http://localhost:8080/users>.

```

@PutMapping("/{username}")
public Post update(@PathVariable("username") String username, @RequestBody Post post) {
    users.stream().filter(user ->
        user.getUsername().equals(username))
        .findAny()
        .ifPresent(user -> user.getPosts().add(post));
    return post;
}

```

Рисунок - PUT-запрос

@PutMapping("/{username}") указывает, что метод принимает PUT-запросы. В нашем примере в запросе мы передаем параметр запроса, а также тело запроса – post. Метод принимает – username, ищет юзера из списка с таким username и добавляем новый пост к его списку постов.

```
@DeleteMapping("/{username}")
public String deleteUser(@PathVariable("username") String username) {
    users.stream().filter(user ->
        user.getUsername().equals(username))
        .findAny()
        .ifPresent(users::remove);
    return "User with username: " + username + " has been deleted";
}
```

Рисунок - Delete-запрос: удаление данных

@DeleteMapping("/{username}") – указывает, что метод принимает DELETE-запросы. Данный метод получает параметр username из запроса и удаляет из списка пользователя с таким username.

## 2.2. Тестирование разработанного приложения

Для проверки работоспособности отправим каждый вид запроса и протестируем. Для этого нам необходим API-client, который может посылать запросы. В качестве него использую POSTMAN.

Тело запроса отправляется в виде JSON.

POST http://localhost:8080/users

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings

● none ● form-data ● x-www-form-urlencoded ● **raw** ● binary ● GraphQL **JSON** ▼

```
1 {
2   "username": "baggio"
3 }
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON ▼

```
1 {
2   "username": "baggio",
3   "posts": []
4 }
```

Рисунок - Post-запрос: создание нового пользователя

GET http://localhost:8080/users

Params Authorization Headers (6) **Body** Pre-request Script Tests Settings

Query Params

KEY	DESCRIPTION
Key	Description

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON ▼

```
1 [
2   {
3     "username": "baggio",
4     "posts": []
5   },
6   {
7     "username": "artur",
8     "posts": []
9   }
10 ]
```

Рисунок - GET-запрос: получение пользователей

PUT http://localhost:8080/users/baggio

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL **JSON**

```
1 {
2   "description": "horse",
3   "imageUrl": "https://unsplash.com/photos/ON8BL319p4g"
4 }
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "description": "horse",
3   "imageUrl": "https://unsplash.com/photos/ON8BL319p4g"
4 }
```

Рисунок - PUT-запрос: обновление списка постов пользователя

DELETE http://localhost:8080/users/artur

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY	DESCRIPTION
Key	Description

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize Text

```
1 User with username: artur has been deleted
```

Рисунок - DELETE-запрос: удаление пользователя по username

## 2.3. Процесс создания микросервисов и использование Docker

Рассмотрим создание микросервисов для примитивного приложения, в котором пользователи могут предлагать события и проводить ивенты. Мы создадим два микросервиса: один для управления событиями (Event Service) и другой для управления пользователями (User Service). Оба микросервиса будут использовать Docker для контейнеризации.

Шаги создания микросервисов и использования Docker:

### 1. Создание Event Service:

- Создать Spring Boot приложение для Event Service.
- Определить сущности, репозитории, контроллеры и настройки базы данных для управления событиями.
- Создать Dockerfile для Event Service:

```
FROM openjdk:11
```

```
ADD target/event-service.jar event-service.jar
```

```
EXPOSE 8080
```

```
ENTRYPOINT ["java", "-jar", "event-service.jar"]
```

### 2. Создание User Service:

- Создать Spring Boot приложение для User Service.
- Определить сущности, репозитории, контроллеры и настройки базы данных для управления пользователями.
- Создай Dockerfile для User Service:

```
FROM openjdk:11
```

```
ADD target/user-service.jar user-service.jar
```

```
EXPOSE 8081
```

```
ENTRYPOINT ["java", "-jar", "user-service.jar"]
```

### 3. Создание Docker-композиции для запуска обоих сервисов:

- Создать файл docker-compose.yml:

```
version: '3'  
services:  
  event-service:  
    build: ./event-service  
    ports:  
      - "8080:8080"  
    depends_on:  
      - db  
  user-service:  
    build: ./user-service  
    ports:  
      - "8081:8081"  
    depends_on:  
      - db  
  db:  
    image: postgres  
    ports:  
      - "5432:5432"
```

### 4. Создание сети для взаимодействия сервисов:

- Указать адрес базы данных как db в настройках каждого микросервиса.

### 5. Сборка и запуск микросервисов с помощью Docker:

- Собрать оба микросервиса с помощью команды docker-compose build.
- Запустить микросервисы с помощью команды docker-compose up.

В результате будет два микросервиса (Event Service и User Service), которые работают в контейнерах Docker. Они могут взаимодействовать друг с



другом через сеть и обеспечивают изоляцию и гибкость в управлении приложением.

### **3. Основы разработки Ui/Ux дизайна**

#### **3.1 Теоретические основы разработки Ui/Ux дизайна**

Разработка интерфейсов приложений, сайтов или игр - сложный процесс, требующий знаний из разных областей, таких как инженерия, психология и дизайн. Для создания удобного и привлекательного пользовательского интерфейса необходимо учитывать множество аспектов дизайна, включая колористику, типографику и управление объектами в пространстве. Дизайнер интерфейса отвечает за эмоции, которые вызывает использование продукта. Для достижения этой цели ему необходимо не только создавать привлекательный визуальный облик, но и учитывать потребности пользователей через исследования, разрабатывать навигационную модель и проектировать интерфейс с учетом иерархии компонентов.

Это является основой UI и UX дизайна. UI или User Interface Design фокусируется на способе представления функциональности сайта или приложения (например, поиск, вкладки, меню) и деталях взаимодействия клиента с интерфейсом. Цель UI-дизайна - создать эстетически приемлемый и современный дизайн продукта. UX расшифровывается как User Experience, что в переводе значит «пользовательский опыт».

UX-дизайн сосредоточен на удобстве и понимании интерфейса потенциальным пользователем. Для этого проводятся исследования, опросы и тестирования концепций, которые становятся основой для создания дизайна. Основной задачей UX-дизайнера является работа над структурой, содержанием, навигацией и взаимодействием пользователя с элементами интерфейса. UX-дизайнер создает карты сайтов, прототипы и базовую структуру программного обеспечения. Целью UX-дизайна является обеспечить пользователю быстрый и безболезненный доступ к необходимой информации на сайте или в приложении. UX/UI дизайн объединяет эти два аспекта, обеспечивая проектирование взаимодействия пользователя с интерфейсом и разработку удобного и

эстетичного внешнего вида. Важно понимать, что дизайн включает не только работу с графическими инструментами, но и глубокое понимание функциональности продукта и создание полезных продуктов для пользователей.

Этапы создания дизайна интерфейсов мобильных приложений включают в себя как UI, так и UX аспекты, объединяя их для достижения оптимального результата:

### 1. Исследование (анализ)

Первым шагом к успешному UI/UX дизайну является проведение исследования рынка. В этот период проводятся маркетинговые исследования, изучаются аналогичные продукты конкурентов. Это помогает создавать решения, которые опережают других участников рынка UI/UX дизайна. Кроме того, на этом этапе происходит знакомство с бизнес-логикой продукта и требованиями клиентов, что позволяет лучше понять потребности пользователей.

2. Вайрфреймы На этом этапе создания дизайна мобильного приложения разрабатываются вайрфреймы, которые показывают, как все экраны будут соединены и какие элементы они будут отображать: от кнопок и всплывающих окон до визуала и текста. Вайрфреймы представляют собой черновой вариант интерфейса без контента, таких как фотографии, видео, цвета и шрифты, которые будут добавлены на более поздних этапах. Они несут в себе логику продукта, показывая планируемый путь использования приложения. Вайрфреймы обычно создаются в черно-белом цвете, чтобы не отвлекать от основного функционала и структуры приложения. Этот этап можно описать как создание чернового скелета приложения, который затем будет заполнен контентом и деталями дизайна.

3. Дизайн-концепт На этом этапе процесса создания дизайна мобильного приложения происходит анализ всей имеющейся информации, на основе которой разрабатывается 1-2 визуальных экрана. Эти экраны представляют собой концепцию стиля и пользовательского опыта будущего продукта. Они отражают основные элементы дизайна, их расположение и визуальное

оформление. Дизайн-концепт является важным инструментом для презентации идей пользователю, позволяя ему получить представление о том, как будет выглядеть и работать приложение. Визуальные экраны помогают пользователям лучше понять функционал продукта и оценить его удобство использования. Кроме того, дизайн-концепт служит основой для последующего развития дизайна и создания более детальных макетов приложения.

4. Визуальный дизайн На этом этапе важно доработать дизайн приложения, добавив анимации, логотипы, цветовую гамму и пользовательские шрифты. Это поможет придать продукту уникальный стиль и создать привлекательный внешний вид. В результате получается красивый и кликабельный прототип, который соответствует ожиданиям пользователей и обеспечивает им удобство использования. Визуальный дизайн играет важную роль в формировании первого впечатления о приложении и способствует узнаваемости бренда. Грамотно выполненный визуальный дизайн помогает привлечь внимание пользователей и сделать продукт более привлекательным для целевой аудитории.

### **3.2. Дизайн приложения**

Визуальный дизайн приложения "EventHub: Планируй и Присоединяйся" будет вдохновлен энергичным настроением. Основной цветовой палитрой будут приглушенно синий и насыщенный красный, символизирующие общительность, активность, энергию, свободу и неформальный стиль.

Интерфейс приложения будет интуитивно понятным и удобным для использования. Главный экран будет содержать самые популярные события в твоём городе. Карточки с информацией о событии будут оформлены в ярком и привлекательном стиле, чтобы мотивировать пользователей к участию в событии.

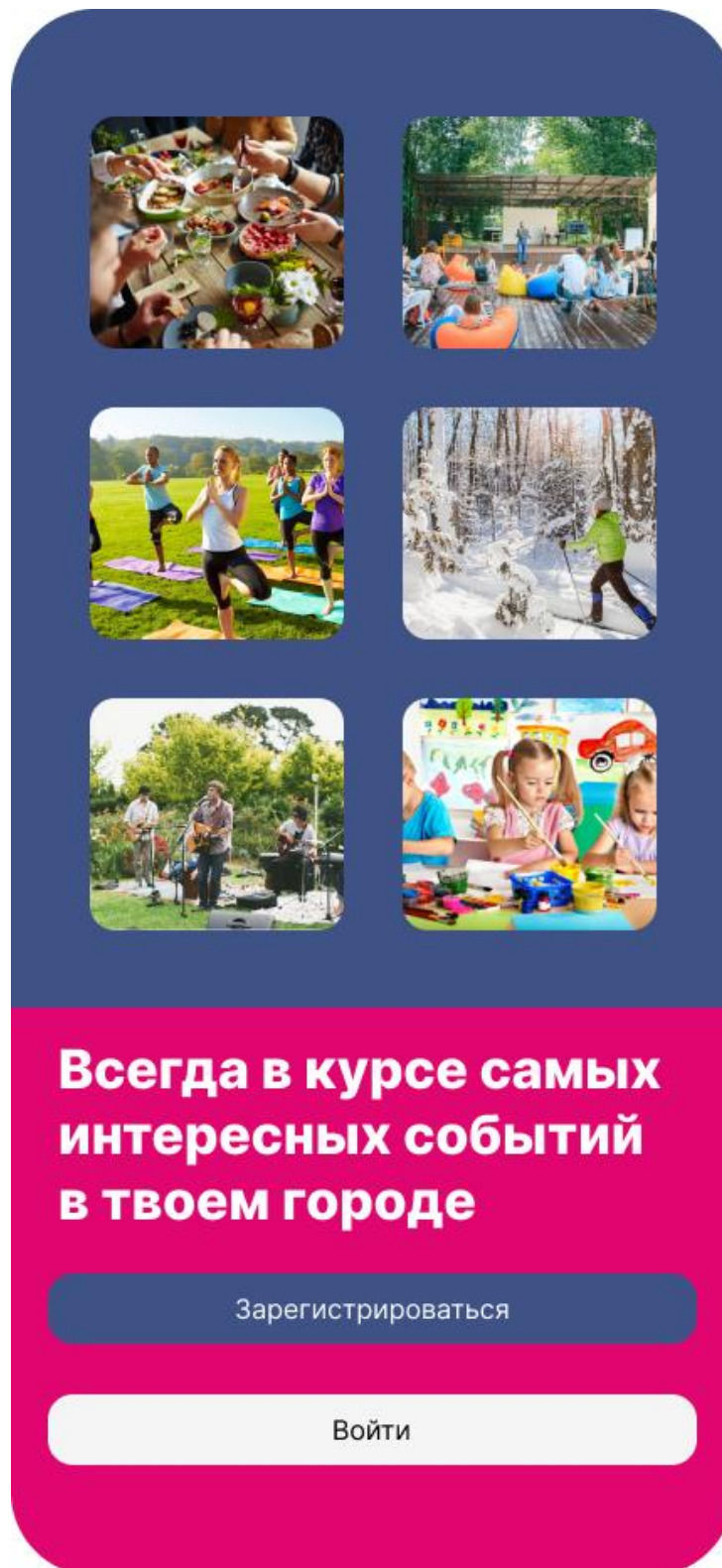



Рисунок – Главный экран с регистрацией

Красивые фотографии мероприятий, здоровых блюд и мотивационные цитаты будут использоваться для создания позитивной атмосферы в приложении.



**Поход по зимнему  
Верх-Исетскому  
пруду на лыжах**

Бесплатно

**15.02.2024**

Встреча в 10.00 переулок Конный 5

Лыжный маршрут проходит по живописным местам вдоль островов  
Длительность 2 часа

Рисунок – Карточка с информацией о событии

Пользовательские шрифты с четкими и читаемыми буквами помогут сделать текстовую информацию легкой для восприятия.



Рисунок – Выбор категорий

Также возможно будет выбрать события по наиболее интересным категориям.

В целом, визуальный дизайн приложения "EventHub: Планируй и Присоединяйся" будет сочетать в себе функциональность, стиль и мотивацию, чтобы помочь пользователям достичь своих целей с удовольствием и энтузиазмом.



## Список литературы

1. Гери Д., Хорстманн К. JavaServer Faces. Библиотека профессионала, 3-е изд.: Пер. с англ. – М.: ООО «И. Д. Вильямс», 2011. – 544 с.: ил. – Парал. тит.англ.
2. Емельянова, Н.З. Проектирование информационных систем: учебное пособие / Н.З. Емельянова, Т.Л. Партыка, И.И. Попов. - М.: Форум, 2014. - 432 с.
3. Заботина, Н. Н. Проектирование информационных систем – М.: ДРОФА, 2013. – 336 с
4. Машнин Т. С. Технология Web-сервисов платформы Java. — БХВПетербург, 2012. — С. 115. — 560 с.:ил.
5. Монахов В. В. Язык программирования Java и среда NetBeans. – 3-тье изд., перераб. и доп. – СПб.: БХВ-Петербург, 2012. – 704 с.: ил.
6. Фримен Э., Фримен Э., Сьерра К., Бейтс Б.. Паттерны проектирования. – СПб.: Питер, 2011. – 656 с.: ил. – М.: Юрайт, 2016. – 260 с.
7. Эккель Б. Философия Java. Библиотека программиста.4-е изд. – СПб.: Питер, 2015. – 640 с.: ил
8. Крейг Уоллс, Роб Харроп. Спринг в действии . - Питер, 2018. - 608 стр.
9. Java Persistence с Hibernate / Гэвин Кинг, Кристиан Бауэр. - ДМК Пресс, 2017. - 400 стр.
10. Билл Бёрк. RESTful Java с JAX-RS 2.1 /. - Питер, 2019. - 288 стр.
11. Роберт Мартин. Чистый код. - Питер, 2010. - 464 стр.
12. Арун Гупта. Docker для Java-разработчиков. - Питер, 2018. - 352 стр.
13. Джошуа Блох. Эффективная Java. - Питер, 2019. - 432 стр.
14. Пишем свой первый сервер на Java и Spring Boot. – URL: <https://proglib.io/p/pishem-svoy-pervyy-server-na-java-i-spring-boot-2023-01-10> (дата обращения: 14.03.2024)

## Приложение А

### ERD – диаграмма приложения

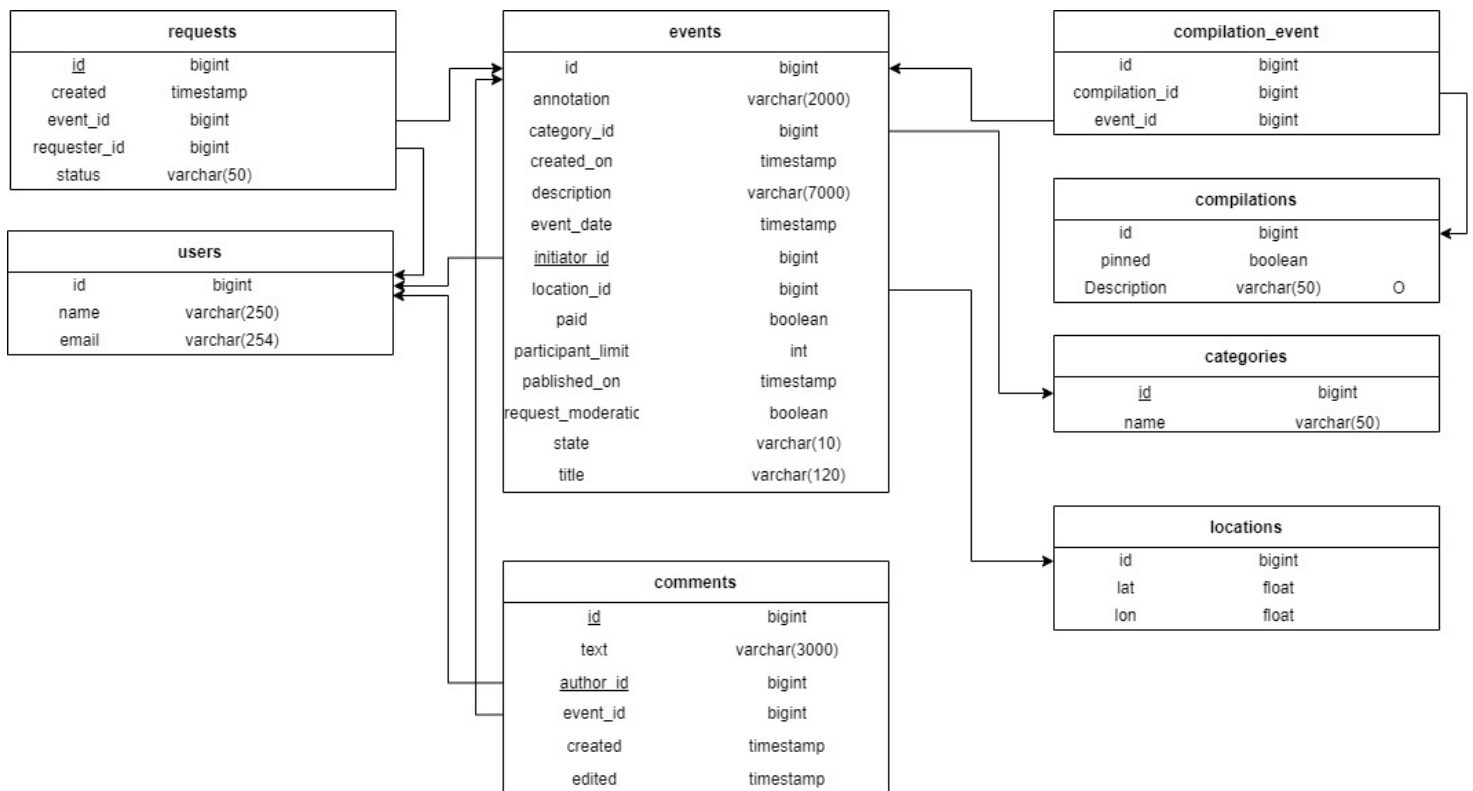


Рисунок - ERD – диаграмма приложения