Geekbrains

**"Базовый функционал социальной сети"**

IT-специалист:

Web разработка на Java

Зотов Д.А.

Москва

2024

**Содержание**

[**Введение** 3](#_Toc167039557)

[**Глава 1. Разработка концепции модели социальной сети** 5](#_Toc167039558)

[1.1. Выбор платформы и средств разработки 5](#_Toc167039559)

[1.2. Определение функциональных требований 2](#_Toc167039560)7

[1.3. Архитектура программы 28](#_Toc167039561)

[**Глава 2. Реализация модели социальной сети** 3](#_Toc167039562)2

[2.1. Описание конфигурационных файлов проекта 32](#_Toc167039563)

[2.2. Описание разработанных компонентов и функциональности 37](#_Toc167039564)

[2.3. Проведение функционального и нагрузочного тестирования 45](#_Toc167039565)

[**Заключение** 50](#_Toc167039567)

[**Список использованных источников** 52](#_Toc167039568)

**Введение**

В современном мире, где скорость и доступность информации играют ключевую роль, социальные сети стали неотъемлемой частью нашей жизни. Они предоставляют платформу не только для общения, но и обмена информацией и формирования сообществ как среди двух людей, так и внутри крупных предприятий. В виду того, что оперативная передача информации играет ключевую роль в современном мире, обеспечивая эффективность и продуктивность во всех сферах деятельности. Она способствует ускорению принятия решений, так как своевременная информация позволяет анализировать ситуацию, принимать взвешенные решения и действовать быстро, что критически важно в динамичных условиях. Также благодаря оперативному обмену информацией повышается эффективность, в виду того, что быстрый доступ к актуальным данным оптимизирует рабочие процессы, минимизирует ошибки и повышает производительность труда. Сокращение рисков – также одно из преимуществ - своевременное получение информации о потенциальных угрозах позволяет предпринять профилактические меры и избежать серьезных последствий. Оперативная передача информации способствует более эффективному взаимодействию между людьми, подразделениями и организациями, а также обеспечивает доступ к актуальным данным позволяет выявлять новые возможности, создавать новые продукты и услуги и быть в авангарде своего рынка.

В данном проекте представлена базовая модель социальной сети, которая демонстрирует ключевые принципы их функционирования, акцентируя внимание на важности обмена информацией, позволяет взаимодействовать с профилями и предоставляет возможность осуществлять поиск, как среди профилей, так и среди сообщений с помощью специально разработанных фильтров, которые используют те или иные критерии.

Целью данной работы является реализация упрощенной работы социальной сети, которая демонстрирует основные принципы работы таких систем и фокусируется на важности передачи информации.

Задачи работы.

1. Произвести анализ состояния проблемы и подходов к ее решению.
2. Произвести анализ и обосновать выбор технологий реализации и необходимых программных платформ.
3. В соответствии с техническим заданием провести разработку проекта характеризующего
4. Подготовить техническую и сопроводительную документацию.
5. Реализовать бота – ассистента с последующей расширяемостью и с использованием современных технологий.

Для реализации поставленных задач, были выбраны следующие технологии:

1. Java 17;
2. Intellij IDEA;
3. Spring Framework;
4. Hibernate;
5. PostgreSQL;
6. Docker.

**Глава 1. Разработка концепции модели социальной сети.**

## Выбор технологий и платформы для разработки

Выбор языка программирования: языки программирования - это а мощные инструменты, позволяющие людям взаимодействовать с компьютерами и создавать цифровые решения. В современном мире, где технологии проникают во все сферы жизни, языки программирования играют решающую роль, являясь основой для создания программного обеспечения, которое управляет, автоматизирует и преобразовывает нашу реальность. От веб-сайтов до искусственного интеллекта:

Языки программирования служат инструментом для реализации инноваций в различных областях, таких как: Разработка веб-сайтов и приложений: языки, такие как HTML, CSS, JavaScript, Python, PHP, позволяют создавать интерактивные веб-ресурсы, мобильные приложения и платформы, предоставляя пользователям доступ к информации, развлечениям, онлайн-сервисам. Анализ и исследование данных: языки, такие как Python, R, SQL, дают возможность анализировать огромные массивы данных, выявлять закономерности, прогнозировать тренды, создавать интеллектуальные системы, используемые в маркетинге, научных исследованиях, финансах. Искусственный интеллект: зыки, такие как Python, Java, C++, используются для создания алгоритмов машинного обучения, систем распознавания образов, виртуальных ассистентов, автономных роботов, открывая новые возможности для автоматизации задач, повышения эффективности и улучшения жизни людей.

Выбор правильного языка: Каждый язык программирования обладает своими уникальными особенностями и предназначением, поэтому выбор зависит от конкретной задачи и требований проекта. Значимость языков программирования: Постоянное развитие технологий: Рынок постоянно эволюционирует, новые технологии требуют новых решений, и языки программирования должны быть гибкими, адаптивными, обеспечивать высокую производительность, безопасность и защиту данных. Поддержка и сообщество: Успешные языки имеют широкое сообщество разработчиков, делящихся знаниями и опытом, предлагающих документацию, курсы, библиотеки, инструменты, что сделает изучение языка более простым и эффективным. Кроссплатформенность: Современные разработчики должны учитывать разнообразие устройств и операционных систем. Язык программирования должен быть достаточно гибким, чтобы работать на различных платформах и эффективно взаимодействовать с другими технологиями. Языки программирования - это фундамент современного мира: Они являются необходимым инструментом для создания качественного и эффективного программного обеспечения, которое обеспечивает успешную реализацию проектов, решает важные задачи, улучшает жизнь людей. Использование современных и актуальных языков программирования позволяет разработчикам идти в ногу с меняющимися требованиями и создавать решения, отвечающие требованиям будущего.

Язык Java.

Изначально язык был создан для программирования бытовых электронных устройств, но очень скоро стало ясно, что его возможности гораздо шире. Язык взяли на вооружение разработчики серверного ПО и клиентских приложений. Так Java начал покорение мира. Очень скоро он вытеснил с рынка очень распространённый в начале 90-х C++.

С 2010 года все права на Java перешли к корпорации Oracle, поглотившей Sun Microsystems. А сам язык вышел на пик своей популярности. И, надо сказать, вполне заслуженно.

Востребованность, рейтинги, области применения.

Java – универсальный объектно-ориентированный язык со строгой типизацией. В нём реализован принцип WORA (от английского: write once, run anywhere). Это позволяет запускать приложения везде, где есть среда исполнения JRE (от английского: Java Runtime Environment). Механизм работы программ следующий. Исходный материал транслируется в байт-код, который обрабатывается виртуальной машиной Java (JVM). При этом не имеет значения, какая операционная система установлена на устройстве.

Благодаря кроссплатформенности язык получил широкое распространение, регулярно занимает ведущие позиции в Топ-5 авторитетных международных рейтингов.

PYPL (PopularitY of Programming Language) – специализированный рейтинг, основанный на данных Google SE. Взглянем на его первую пятёрку. Java – на почётном втором месте (Рис. 1).



Рис.1 – статистика PYPL

Stack Overflow (раздел технологий) – расширенный рейтинг, основан на количестве вопросов, касающихся языков программирования и скриптов. Java – в первой пятёрке (Рис. 2).

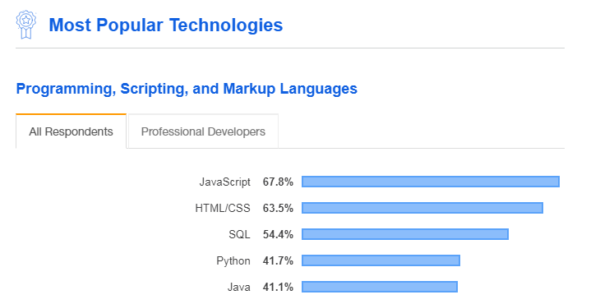


Рис.2 – статистика Stack Overflow

В международных рейтингах IEEE Spectrum и TIOBE (также как в PYPL) Java занимает вторые места в ТОП-5 (за 2019/2020 год). Это однозначно свидетельствует о лидирующих позициях языка в мировой IT-индустрии.

Java был протестирован программистами в самых разных сферах: от карманных компьютеров до интерактивного телевидения. Сейчас он наиболее востребован в таких направлениях, как

* веб-разработка (масштабные бизнес-проекты);
* программы для ПК (десктопный софт);
* компьютерные игры (например, Minecraft);
* приложения для мобильных устройств (ОС Android);
* научные исследования и разработки;
* промышленный программинг.

Компания ORACLE утверждает, что общее количество всех устройств, работающих с Java, достигло 3-х миллиардов. Это сообщение стало маркетинговым аргументом. Даже если цифра слегка преувеличена, популярность языка не вызывает сомнений.

Технические особенности и достоинства

Изначально перед создателями Java стояла задача сделать синтаксис более простым и понятным в сравнении с С/С++. В итоге так и получилось. Сходство между этими языками есть, но освоить Java и кодировать на нём гораздо легче.

У Java масса достоинств, благодаря которым многие разработчики выбирают именно его:

* Простота – первое техническое достоинство Java. У него чёткие синтаксические правила и понятная семантика. Рациональность и краткость очень полезны для обработки кода машинами с ограниченным объёмом ресурсов. Для встроенных устройств создана специальная платформа Java Micro Edition.
* Объектно-ориентированный подход. За 3 десятилетия он доказал свою эффективность. Суть состоит в том, что в центре внимания находятся данные (объекты), интерфейсы и алгоритмы вторичны. Другими словами, мы отталкиваемся от результата при выборе инструментов, способов их применения.
* Безопасность. Важнейший критерий, учитывая применение языка в сетевых / распределенных средах. Разработчики проделали большую работу по защите платформы Java. И она продолжается. Обойти или взломать механизмы защиты крайне сложно. Пример: использование классов, имеющих цифровую подпись. Полные права предоставляются только при полном доверии автору класса.
* Производительность. Изначально она вызывала вопросы. Новые версии динамических компиляторов Java не уступают традиционным из других платформ. Мощный прирост скорости обработки даёт оптимизация тех фрагментов кода, которая исполняется чаще. При необходимости те или иные приёмы оптимизации включаются или отменяются JIT-компилятором.
* Надёжность – одно из важнейших преимуществ. Программы на Java стабильно работают в любых условиях. Компилятор способен выявить ошибки ещё до выполнения кода, то есть на ранних стадиях. Контроль выполнения позволяет предотвратить сбои в памяти (например, из-за неточного указателя). Сами указатели можно применять не везде, а только там, где это необходимо (скажем, в работе со связными списками).
* Независимость от аппаратной части и ОС. Важно лишь наличие исполняющей среды и JVM. А компьютерная архитектура в целом значения не имеет. Байт-код легко интерпретируется на любой машине. Подход доказал свою состоятельность во многом благодаря динамической компиляции. Кроссплатформенностью отличается также интерфейс, реализованный в системных библиотеках.
* Динамичность и адаптируемость. Эта особенность позволяет Java не теряться в постоянно изменяющейся среде. При необходимости можно добавить в библиотеки новые объекты, методы. При этом трогать приложения, использующие данные библиотеки не нужно. Отследить информацию о структуре объектов, их поведении, о ходе выполнения программы очень легко.
* Удобные и эффективные сетевые возможности. Приложения умеют находить нужные объекты в сети и открывать к ним доступ. Причём так же легко, как будто мы имеем дело с локальной файловой системой. Имеется обширная программная библиотека для передачи данных по самым распространённым протоколам: FTP, HTTP, TCP/IP. Работает механизм вызова удалённых методов.

Важно отметить, что Java – это трио, состоящее из языка программирования, мощного универсального обработчика и внушительной библиотеки. Все эти наработки доступны программистам. Им не нужно разрабатывать с нуля многие необходимые процедуры (доступ к сети, базам данных и т.п). Это также весомый аргумент в пользу Java.

Для написания данного проекта был выбран язык Java 17 версии, который представляет собой мощный инструмент для разработчиков, предлагающий множество улучшений и преимуществ, которые повышают производительность, безопасность и масштабируемость приложений. От улучшенной производительности и оптимизированного времени запуска до расширенных функций безопасности и поддержки современных платформ. Также Java 17 не только гарантирует доступ к последним технологиям, но и обеспечивает долгосрочную поддержку и стабильность. Его активное развитие, экосистема и широкое распространение делают его надежным и перспективным языком для решения имеющихся задач.

Spring framework.

С помощью Spring Java-разработчики создают серверное ПО для работы с современными десктопными, мобильными и веб-приложениями. Spring полезен как для простых сервисов, так и для больших проектов из множества модулей и cотен функций.

Spring помогает:

* Проектировать приложения. Создавать каркас, с которым в дальнейшем будут работать программисты.
* Писать меньше рутинных функций. Фреймворк содержит множество базовых инструментов, которые можно не писать самостоятельно. Например, быструю настройку полноценной интеграции с базой данных или передачу информации через интернет.
* Объединять между собой разные модули приложения. Строить зависимости и подключать их друг к другу без риска нарушить связи между компонентами.

Java Spring Framework — это большой набор разных более маелньких фреймворков. Каждый из них нужен для работы над определёнными приложениями или их частями.

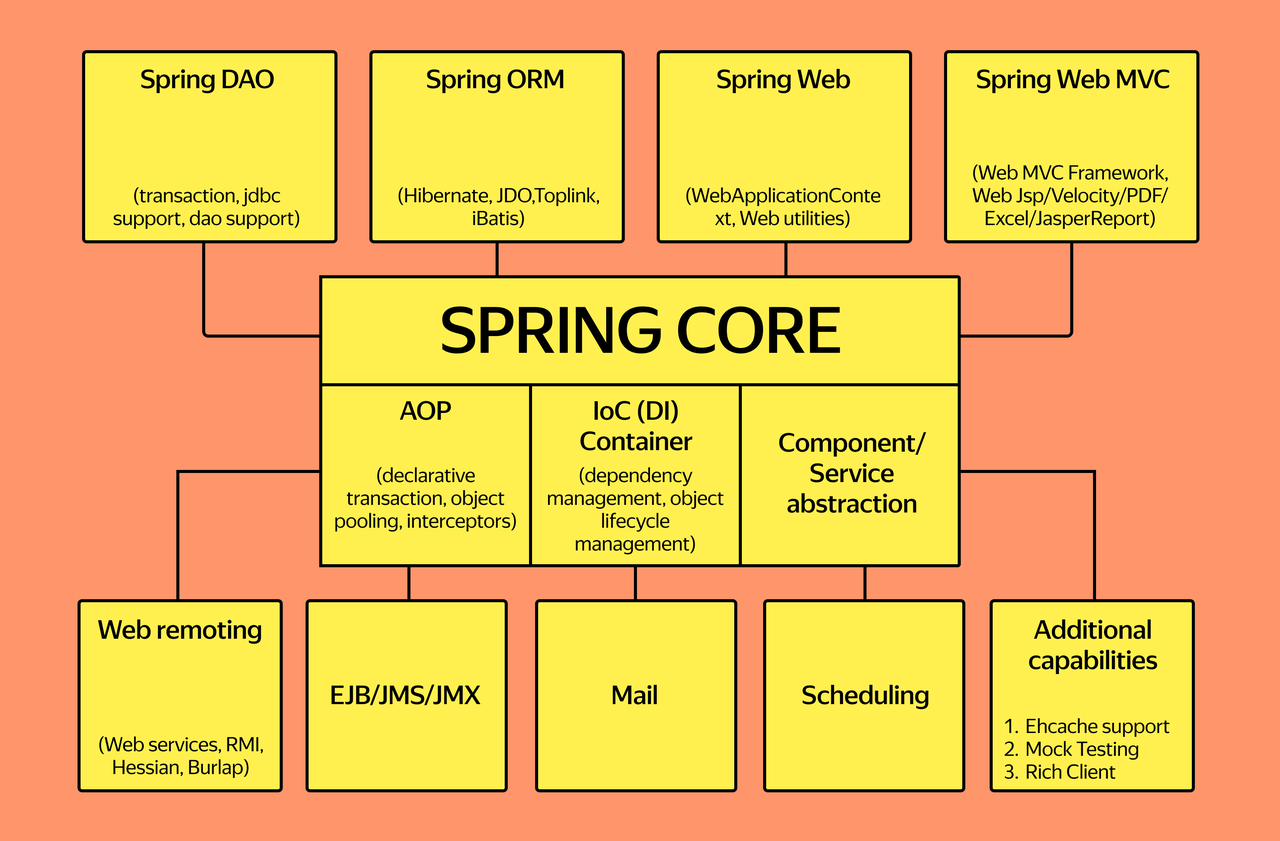


Рис. 3 – Модули Spring

Основные компоненты, входящие в Spring, можно подключать по отдельности, в зависимости от нужд проекта

Рассмотрим несколько самых популярных компонентов, с которыми программисты чаще всего сталкиваются в работе (Рис. 3).

* IoC. Этот модуль лежит в основе Spring и контролирует взаимодействие разных компонентов между собой. С помощью специальных аннотаций можно описать зависимости между компонентами, соединяя их в единую архитектуру. В итоге компоненты работают сами по себе, а связями между ними управляет специальный контейнер. Это позволяет даже в случае изменения компонентов сохранять работоспособность всей системы.
* Модуль доступа к данным. Объединяет сразу несколько библиотек и отвечает за взаимодействие программ с базами данных. Этот модуль превращает объекты Java в записи базы данных, позволяет вносить изменения, организовывать быстрый и безопасный доступ к данным.
* Spring Cloud. Этот модуль позволяет интегрировать между собой разные элементы систем, соединённые по сети: компьютеры, серверы, PaaS-платформы и другие. Для некоторых популярных решений есть отдельные подмодули, например Spring Cloud Azure. Внутри Spring Cloud есть инструменты для маршрутизации запросов, балансировки нагрузки на отдельные сервисы, обмена сообщениями.
* AOP-модуль. Позволяет реализовать не совсем стандартную парадигму — аспектно-ориентированное программирование. С его помощью программу можно сделать более выразительной и чёткой за счёт разделения на отдельно функционирующие части с разными зонами ответственности. АОП позволяет неявно работать с разными функциями системы так, что они не могут помешать работе друг друга. Это улучшает архитектуру приложения и позволяет программисту не отвлекаться от бизнес-задач на рутину.
* Модуль транзакций. В Java транзакциями называют последовательность запросов к БД. Модуль транзакций выполняет важные запросы надёжнее и гарантирует сохранность данных.
* Модуль MVC. Этот модуль реализует популярную схему веб-приложений — разделение её на три части. Первая часть отвечает за данные, вторая — за отображение интерфейсов, третья — за изменение данных в ответ на действия пользователя.
* Модуль авторизации. Этот модуль позволяет настраивать авторизацию и аутентификацию в системе. Он обеспечивает разделение прав пользователей и защиту паролей.
* Модуль бизнес-приложений. Это фреймворк Spring Roo, который позволяет быстро создавать структуру крупных приложений и универсальные шаблонные решения.
* Модуль интеграции. Он называется Spring Integration и предназначен для того, чтобы интегрировать корпоративные приложения в единую систему: обмениваться сообщениями, активировать функции и строить единую архитектуру.
* Spring Boot. Это отдельный модуль, который упрощает настройку фреймворка Spring и ускоряет запуск проектов. Он может автоматически сконфигурировать приложение и создать веб-сервер для его запуска. Большинство новых приложений Spring создаётся с помощью Spring Boot.

Преимущества Spring.

* Универсальность. В Spring встроено множество инструментов, поэтому зачастую его одного достаточно для решения большинства задач разработки.
* Ускорение работы. Фреймворк позволит сделать за пару дней то, на что без него ушли бы месяцы разработки.
* Избавление от рутины. Фреймворк позволяет автоматически решать однотипные, рутинные задачи. Это позволяет программистам сосредоточиться на творческой работе.
* Большое сообщество. Spring существует давно и очень популярен у разработчиков. Есть подробная документация, часть которой переведена на русский. В профильных сообществах можно найти ответ на любой вопрос. У фреймворка открытый исходный код, и энтузиасты постоянно его улучшают.
* Доступность. Обычно технологии, связанные с Java, платные и закрытые. Spring же бесплатен, так что его легко использовать даже для личных целей.

Выбор фреймворка Spring для данного проекта обусловлен этими и другими преимуществами. Возможность использовать готовые проверенные решения инкапсулированные внутри фреймворка позволяют реализовать проект быстрее и надёжнее.

Hibernate.

Hibernate - это фреймворк на Java, который поставляется со слоем абстракции и обрабатывает реализации внутри компании. Реализации включают в себя такие задачи, как написание запроса для CRUD операций или установление соединения с базами данных и т.д.

Данный фреймворк – это программное обеспечение, обеспечивающее абстракцию на нескольких технологиях, таких как JDBC, сервлет и т.д.

Hibernate разрабатывает логику сохранения, которая хранит и обрабатывает данные для более длительного использования. Это легкий инструмент ORM и, что наиболее важно, с открытым исходным кодом, что дает ему преимущество перед другими фреймворками.

Это метод, который отображает объект, хранящийся в базе данных. Инструмент ORM упрощает создание данных, манипулирование ими и доступ к ним. Он внутренне использует Java API для взаимодействия с базами данных (Рис. 4).

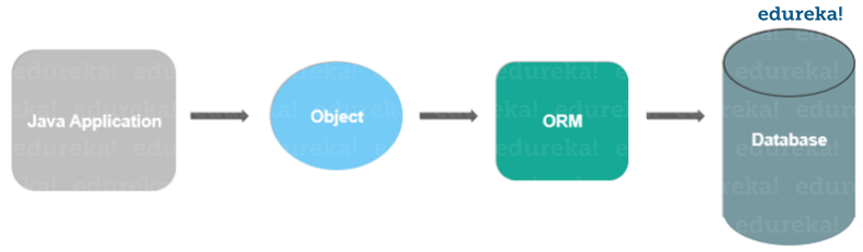


Рис.4 – Структура Hibernate

Рассмотрим необходимость использования hibernate в Java.

* Hibernate устраняет недостатки других технологий, таких как JDBC. Давайте посмотрим, насколько он оптимизирует задачи лучше, чем JDBC.
* Hibernate устраняет зависимость от базы данных, с которой сталкивается JDBC.
* Изменение баз данных обходится дорого, работая на JDBC, hibernate с честью решает эту проблему.
* Переносимость кода невозможна при работе с JDBC, с чем легко справляется hibernate.
* Hibernate усиливает связь на уровне объектов. Он преодолевает часть обработки исключений, которая является обязательной при работе с JDBC.
* Hibernate устраняет взаимосвязь на уровне объектов. Это сокращает длину кода с улучшенной удобочитаемостью за счет решения проблемы с котельной панелью.
* Hibernate предоставляет оптимальные и эффективные решения для любой задачи, преодолевая все недостатки JDBC. Давайте рассмотрим различные операции, а также технологии и базы данных, с которыми мы можем работать при использовании hibernate Framework в Java.

Будучи фреймворком с открытым исходным кодом, он доступен для всех без каких-либо затрат. Исходный код hibernate можно найти в Интернете, который также допускает модификации.

Преимущество облегченного фреймворка можно увидеть в значительно меньшем пакете для установки. Эффективность повышается за счет отказа от использования какого-либо контейнера для выполнения. Хотя hibernate может работать с несколькими технологиями одновременно, но это не означает, что hibernate не может работать в одиночку. Мы также можем работать с hibernate в одиночку, то есть без каких-либо технологий.

Hibernate имеет особую природу, при которой ему не нужно реализовывать интерфейсы hibernate API или расширяться из классов hibernate API, поскольку классы разработки приложений hibernate слабо связаны.

Функциональные возможности, поддерживаемые Hibernate.

* Hibernate использует язык запросов Hibernate, что делает его независимым от базы данных.
* Он поддерживает автоматические операции DDL.
* Hibernate поддерживает автоматическую генерацию первичного ключа.
* Он поддерживает кэш-память.
* Обработка исключений не является обязательной для hibernate.
* Наиболее важным является то, что hibernate - это инструмент ORM.
* Поддерживаемые базы данных в Hibernate

Ниже приведены базы данных, поддерживаемые hibernate в Java.

* Движок базы данных HSQL
* MYSQL
* Oracle
* FrontBase
* PostgreSQL
* DB2/NT
* Sybase SQL Server
* Динамический сервер Informix
* База данных Microsoft SQL Server

Hibernate поддерживает практически все основные СУБД, что делает его эффективным и простым в работе.

Технологии, поддерживаемые Hibernate.

* XDoclet Spring
* Maven
* Плагины Eclipse
* J2EE

Преимущества Hibernate в Java.

* Легкий и с открытым исходным кодом – легкий и открытый исходный код делает его доступным и эффективным.
* Повышенная производительность – использование кэш-памяти способствует повышению производительности.
* Независимость от базы данных – независимость от базы данных дает ей возможность работать с различными базами данных.
* Автоматические операции DDL – автоматическое создание таблиц избавляет нас от необходимости создавать таблицы вручную.
* Он заботится о отображении баз данных классов Java с использованием XML-файлов без написания какого-либо кода.
* Мы можем напрямую хранить и извлекать данные непосредственно из базы данных, используя простые API.
* Для работы не требуется никакого сервера приложений.
* Минимизирует доступ к базе данных с помощью стратегий интеллектуальной выборки.
* Он обеспечивает простой запрос данных.

Перечисленные преимущества стали ключевым фактором выбора технологии Hibernate в данном проекте. Ее мощные возможности по работе с объектно-реляционным отображением, а также поддержка транзакций и кэширования обеспечивают высокую производительность и стабильность приложения. Hibernate также предоставляет богатый набор инструментов для управления данными, что значительно упрощает разработку и делает ее более эффективной.

PostgreSQL.

PostgreSQL — это СУБД (объектно-реляционная система управления базами данных), написанная на языке программирования C, представляющая собой опенсорс решение с открытым исходным кодом, распространяемая по свободной лицензии.

Применяемая модель объектно-реляционной СУБД означает поддержку всех требований к реляционным БД, а также и механизмов, которые реализуют объектно-ориентированные языки программирования, включая объектные модели, классы и наследование, публичные и приватные функции, процедуры и т.д.

Такая концепция, вместе со свободной лицензией распространения и открытым программным кодом, быстро сделала СУБД PostgreSQL наиболее известной и популярной на рынке. Ее используют многие мировые гиганты сферы IT, например, платформы Alibaba, сервисы Yahoo и другие.

Система управления базами данных PostgreSQL имеет ряд неоспоримых преимуществ перед другими решениями, которые представлены на рынке. Обеспечивается поддержка широкого набора данных. Гибкость, использование не только стандартизированных типов данных, но других, например, координат для геолокации и картографических сервисов, сетевых и MAC-адресов, а также результатов полнотекстового поиска. Применяются надстройки и расширения, которые легко позволят добавить нужный тип данных. Предусмотрен центральный репозиторий PGXN, где представлены тысячи опенсорс модулей на языке С, которые можно загружать абсолютно бесплатно.

Гибкий настраиваемый интерфейс, позволяет организовать хранение данных в таблицах и быстрый доступ к ним пользователей и разнообразных приложений, от веб-порталов до сложных распределенных систем. Максимальная универсальность, поддержка стандартов и присущих для SQL правил формирования запросов. Надежность и безопасность, разграничение прав доступа пользователей и управление ими. Многие сравнивают СУБД PostgreSQL с популярной коммерческой системой Oracle, считая ее наиболее эффективным бесплатным аналогом.

Ниже представлены преимущества, которые выгодно отличают PostgreSQL от других СУБД.

* Интегрированая система расширений, удобная возможность для администратора задавать типы данных, подключать необходимые модули из каталога PGXN, а также использовать различные функции и процедуры, написанные на нескольких языках программирования, включая Питон, Ява, PHP;
* Простота масштабирования и повышение производительности за счет применения индексировании, обобщенного дерева поиска и наследования таблиц;
* Команды и инструменты для анализа времени обработки запросов, переиндексации, удаления мусора;
* Кроссплатформенность, обеспечивается работа с любыми видами приложений и операционными системами, всеми дистрибутивами Линкус, а также Windows и MacOS;
* Поддержка языка разметки XML;
* Свободная лицензия распространения BSD, открытый программный код с возможностью загрузки из репозитория на гит хабе.
* Имеется максимально подробное справочное руководство, а также живое сообщество разработчиков и пользователей.
* Система PostgreSQL считается одной из наиболее стабильных и надежных, выгодно отличаясь от других бесплатных продуктов. Над проектом работает команда опытных разработчиков и предлагаемые релизы поступают в свободный доступ только после тщательного тестирования. Также регулярно обновляется техническая документация, раздел справки. Есть статистика и данные независимых исследований, что по количеству обнаруженных ошибок.

В виду вышеперечисленных преимуществ для хранения данных профилей и сообщений выбрана именно эта система, что позволило надёжно хранить и взаимодействовать с данными.

Docker.

Docker — это платформа для разработки, доставки и запуска контейнерных приложений. Docker позволяет создавать контейнеры, автоматизировать их запуск и развертывание, управляет жизненным циклом. Он позволяет запускать множество контейнеров на одной хост-машине.

Контейнеризация похожа на виртуализацию, но это не одно и то же. Виртуализация запускает полноценный хост на гипервизоре со своим виртуальным оборудованием и операционной системой. При этом внутри одной ОС можно запустить другую ОС. В случае контейнеризации процесс запускается прямо из ядра основной операционной системы и не виртуализирует оборудование. Это означает, что контейнеризованное приложение может работать только в той же ОС, что и основная. Контейнеры не виртуализируют оборудование, поэтому потребляют меньше ресурсов.

Архитектура (компоненты) Docker (Рис. 5).

Рассмотрим подробнее про компоненты, из которых состоит Docker.

Docker daemon. Это некоторый резидентный процесс, который запущен на хост-машине постоянно. Он владеет всей инфраструктурой, а также предоставляет интерфейс взаимодействия с контейнерами, включающего создание и удаление, запуск и остановку.

В ранних версиях платформы Docker можно встретить упоминание о dockerd, но на текущий момент демоны уже успели разбиться на отдельные проекты. Все чаще можно встретить его современника — containerd.

Docker client (клиент). Это интерфейс командной строки для управления Docker daemon. Мы пользуемся этим клиентом, когда создаем и разворачиваем контейнеры, а клиент отправляет эти запросы в Docker daemon.

Docker image (образ). Это неизменяемый файл (образ), из которого разворачиваются контейнеры. Приложения упаковываются именно в образы, из которых потом уже создаются контейнеры. В технической литературе можно также встретить описание image как шаблона запуска процесса.

Приведем аналогию на примере установки операционной системы. В дистрибутиве (образе) ОС есть все, что необходимо для ее установки. Но этот образ нельзя запустить, для начала его нужно «развернуть» в готовую ОС. Так вот, дистрибутив для установки ОС — это образ, а установленная и работающая ОС — это контейнер. Но контейнеры обычно разворачиваются одной командой — это намного проще и быстрее, чем установка ОС.

Docker container (контейнер). Это - уже развернутое из образа и работающее приложение.

Docker Registry. Это - репозиторий с образами. Разработчики создают образы своих программ и выкладывают их в репозиторий, чтобы их можно было скачать и воспользоваться ими. Распространенный публичный репозиторий — Docker Hub. В нем собраны образы множества популярных программ или платформ: базы данных, веб-серверы, компиляторы, операционные системы и так далее. Также можно создать свой приватный репозиторий, например, внутри компании. Разработчики будут размещать там образы, которые будут использоваться всей компанией.

Dockerfile. Dockerfile — это инструкция для сборки образа. Это простой текстовый файл, содержащий по одной команде в каждой строке. В нем указываются все программы, зависимости и образы, которые нужны для разворачивания образа.

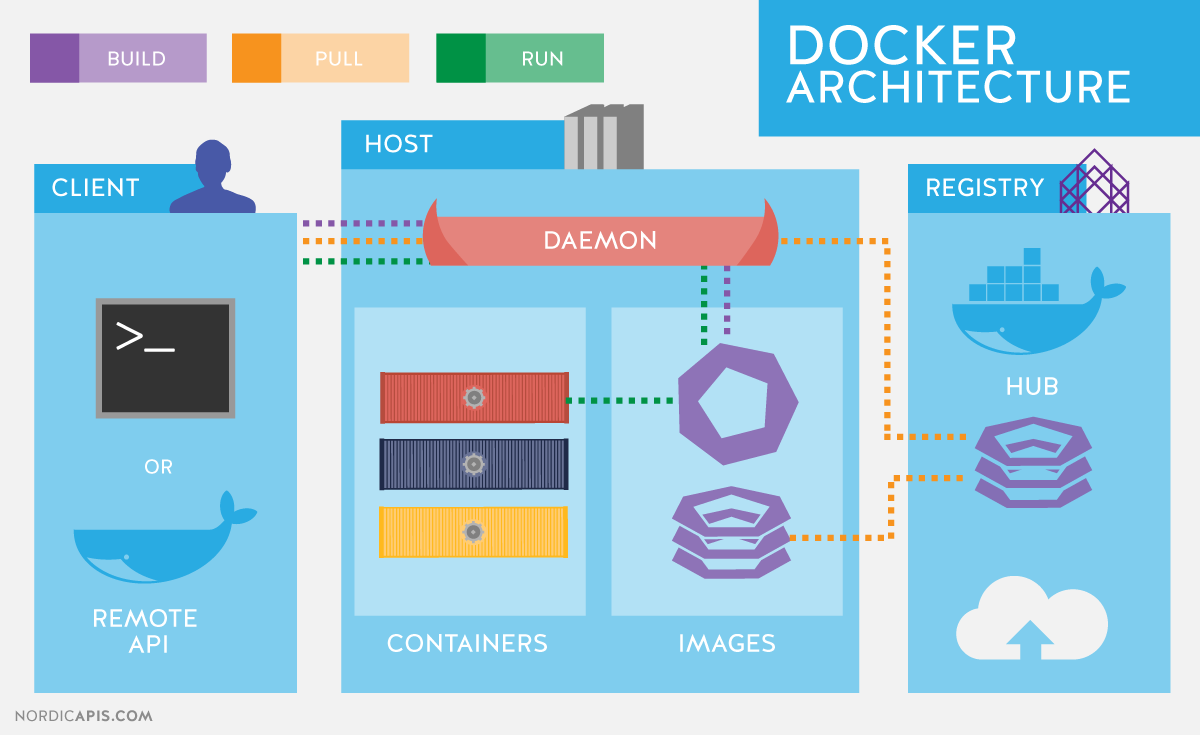


Рис. 5 – Архитектура Docker

Преимущества использования контейнеров Docker.

* Контейнеры упрощают работу как программистам, так и администраторам, которые развертывают эти приложения. Docker решает проблемы зависимостей и рабочего окружения. Контейнеры позволяют упаковать в единый образ приложение и все его зависимости: библиотеки, системные утилиты и файлы настройки. Это упрощает перенос приложения на другую инфраструктуру. Например, разработчики создают приложение в системе разработки — там все настроено, приложение работает. Когда оно готово, его нужно перенести в систему тестирования, а затем в продуктивную среду. Если в одной из них нет нужной зависимости, приложение не будет работать. Программистам придется отвлечься от разработки и совместно с командой поддержки разобраться в ситуации. В контейнерах такой проблемы нет, так как они содержат в себе все необходимое для запуска приложения. Специалисты занимаются разработкой, а не решением инфраструктурных проблем.
* Изоляция и безопасность. Контейнер — это набор процессов, изолированных от основной операционной системы. Приложения работают только внутри контейнеров и не имеют доступа к основной операционной системе. Это повышает безопасность приложений:они не смогут случайно или умышленно навредить основной системе. Если приложение в контейнере завершится с ошибкой или зависнет, это никак не затронет основную ОС.
* Ускорение и автоматизация развертывания приложений и масштабируемость. Контейнеры упрощают развертывание приложений. В классическом подходе для установки программы нужно совершить несколько действий: выполнить скрипт, изменить файлы настроек и так далее. В этом процессе не исключена вероятность человеческой ошибки: пользователь запустит скрипт два раза, перепутает последовательность или что-то не поймет. Контейнеры позволяют полностью автоматизировать этот процесс, так как включают в себя все нужные зависимости и порядок выполнения действий. Также контейнеры упрощают развертывание на нескольких серверах. В классическом подходе для того, чтобы развернуть одно и то же приложение на нескольких машинах, нужно будет повторять одни и те же действия. Контейнеры избавляют от этой рутинной работы и позволяют автоматизировать развертывание.
* Контейнеры приближают к микросервисной архитектуре. Контейнеры хорошо вписываются в микросервисную архитектуру. Это подход к разработке, при котором приложение разбивается на небольшие компоненты, по возможности независимые. Обычно противопоставляется монолитной архитектуре, где все части системы сильно связаны друг с другом. Это позволяет разрабатывать новую функциональность быстрее, ведь в случае с монолитной архитектурой изменение какой-то части может затронуть всю остальную систему.
* Docker compose — одновременно развернуть несколько контейнеров. Docker-compose позволяет разворачивать и настраивать несколько контейнеров одновременно. Например, для веб-приложения нужно развернуть стек LAMP: Linux + Apache, MySQL, PHP. Каждое из приложений — это отдельный контейнер для ОС Linux. Но в этой ситуации нам нужны именно все контейнеры вместе, а не отдельно взятое приложение. Docker-compose позволяет развернуть и настроить все приложения одной командой, а без него пришлось бы разворачивать и настраивать каждый контейнер отдельно.
* Хранение данных в Docker. Одна из главных особенностей контейнеров — эфемерность. Это означает, что контейнеры могут быть в любой момент остановлены, перезапущены или уничтожены. При этом все накопленные данные в контейнере будут потеряны. Поэтому приложения нужно разрабатывать так, чтобы они не полагались на хранилище данных в контейнере, это называется принципом Stateless. Это хорошо подходит для приложений или сервисов, которые не сохраняют результаты своей работы. Например, функции расчета или преобразования данных: им на вход поступил один набор данных, они его преобразовали или рассчитали и вернули результат. Все, ничего никуда сохранять не нужно. Но далеко не все приложения такие, и есть много данных, которые нужно сохранить. В контейнерах для этого предусмотрены несколько способов.
* Тома (Docker volumes). Это способ, при котором Docker сам создает директории для хранения данных. Их можно сделать доступными для разных контейнеров, чтобы они могли обмениваться данными. По умолчанию эти директории создаются на хост-машине, но можно использовать и удаленные хранилища: файловый сервер или объектное хранилище.
* Монтирование каталога (bind mount). В этом случае директория сначала создается на хост-машине а уже потом монтируется в контейнеры.
* Но этот способ не рекомендуется, потому что он усложняет резервное копирование, миграцию и совместное использование данных несколькими контейнерами.

Базы данных с профилями и сообщениями находятся именно в докер котейнерах. Это обусловенно вышеперечисленными преимуществами и способствует более удобой разработке и использованию проекта.

Intellij Idea (Рис. 6).

* Причины использования IntelliJ IDEA в качестве среды разработки для модели социальной сети на Java.
* Удобство и эффективность. Интеллектуальный автодополнение. IntelliJ IDEA обладает мощной системой автодополнения кода, которая не только предлагает варианты, но и анализирует контекст и предлагает наиболее подходящие. Это значительно ускоряет процесс написания кода и снижает количество ошибок.
* Рефакторинг. Инструменты рефакторинга позволяют легко изменять структуру кода, переименовывать переменные, методы и классы, не нарушая функциональность приложения. Это особенно полезно при разработке сложных проектов, таких как модель социальной сети.
* Отладка. Инструменты отладки в IntelliJ IDEA позволяют пошагово выполнять код, анализировать значения переменных, отслеживать стек вызовов, что значительно упрощает поиск и исправление ошибок.
* Поддержка Java. IntelliJ IDEA глубоко интегрирована с Java и обеспечивает поддержку всех современных функций языка, включая лямбда-выражения, потоки, модули и многое другое.
* Профилирование. IntelliJ IDEA позволяет профилировать приложение, анализировать его производительность и выявлять узкие места. Это важно для оптимизации кода и обеспечения стабильной работы социальной сети при большом количестве пользователей.
* Тестирование. IntelliJ IDEA предоставляет инструменты для написания и запуска юнит-тестов, а также интеграционные тесты. Это необходимо для обеспечения качества кода и раннего обнаружения ошибок.
* Разработка веб-приложений. IntelliJ IDEA предоставляет плагины для работы с популярными Java-фреймворками, такими как Spring, Spring Boot, Hibernate, Struts. Это упрощает разработку веб-компонентов для модели социальной сети.
* Deployment. Intelli IDEA позволяет легко развернуть приложение на сервере, а также управлять его работой и настройкой.
* Большое количество плагинов. Intelli IDEA имеет широкий выбор плагинов, расширяющих её функциональность и добавляющих поддержку новых технологий.
* Наличие качественной документации. Intelli IDEA имеет широкую и детальную документацию, которая помогает быстро изучить все возможности IDE и решить возникшие вопросы.

В целом, IntelliJ IDEA предоставляет все необходимые инструменты и функции для эффективной разработки модели социальной сети на Java, повышает производительность разработки, обеспечивает высокое качество кода и упрощает процесс развертывания приложения.

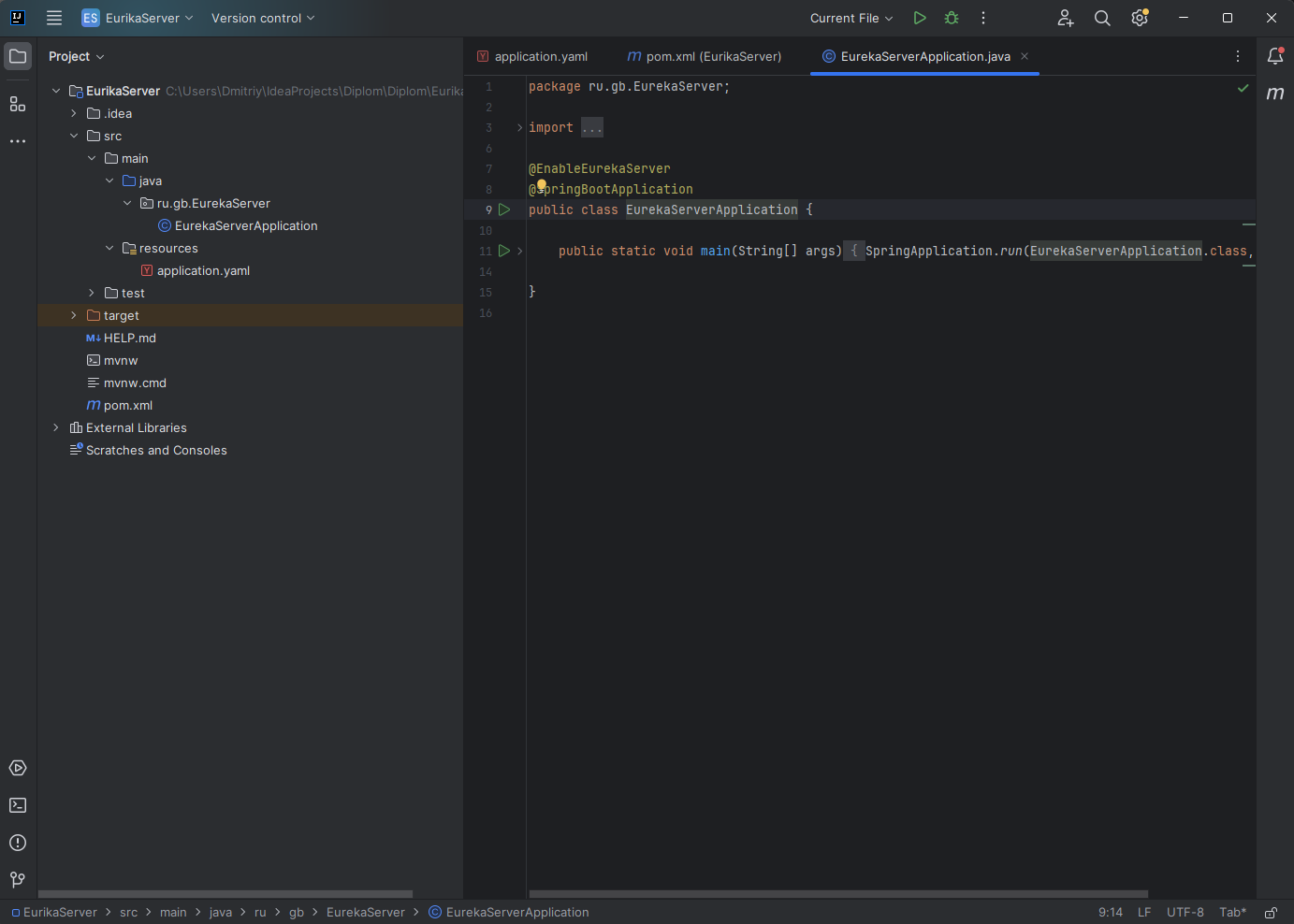
****

Рис. 6 – Интерфейс Intellij Idea

1.2 Определение функциональных требований к программе.

Социальные сети стали неотъемлемой частью нашей жизни, предоставляя платформу для общения, обмена информацией и построения отношений. Они позволяют людям, группам и целым предприятиям не только связываться друг с другом, но и оперативно делиться информацией, которая может быть решающей в различных ситуациях. Задача программы, создающей социальную сеть, прежде всего, заключается в обеспечении такой возможности.

Основная функция программы – обмен сообщениями, позволяющий пользователям отправлять и получать информацию. Для удобства поиска и нахождения нужных сообщений в программе должен присутствовать эффективный фильтр. Он позволяет находить сообщения по различным критериям, таким как дата отправки, автор, ключевые слова в тексте и т.д.

Помимо обмена сообщениями, любая социальная сеть должна обладать рядом других ключевых функций:

1. Профили пользователей: Чтобы отправлять и получать сообщения, необходимы профили пользователей. Программа должна позволять создавать, редактировать и управлять профилями, предоставляя пользователям возможность определять свои личные настройки.

2. Безопасность: Социальные сети должны обеспечивать высокий уровень безопасности, гарантируя, что каждый профиль имеет доступ только к своим данным и функциям. Не должно быть возможности несанкционированного доступа к чужим данным, а также подделки или изменения информации.

3. Поиск профилей: Для того, чтобы найти конкретного пользователя и отправить ему сообщение, необходимо иметь возможность его поиска. Программа должна предлагать удобные инструменты для поиска по различным критериям. Например, по имени, возрасту, интересам, географическому положению, месту работы или учебы, а также по списку общих знакомых.

4. Хранение информации о сообщениях: Сообщения должны храниться в системе в течение определенного времени, чтобы пользователи могли к ним вернуться в любое время, а разработчики имели возможность анализировать активность пользователей, отслеживать возникновение проблем и совершенствовать программу.

Базовые функции, описанные выше, являются основой любой социальной сети. В будущем они могут видоизменяться и дополняться новыми функциями, чтобы удовлетворить изменяющиеся потребности пользователей и обеспечить прогрессивное развитие социальных сетей.

1.3 Архитектура программы (Рис. 7).

Разработка программы осуществлялась с использованием микросервисной архитектуры. Архитектура микросервисов – это подход к проектированию, при котором приложение состоит из слабо связанных, независимо развертываемых сервисов. В отличие от монолитных архитектур, где все функциональные возможности тесно интегрированы в единую кодовую базу, микросервисы обеспечивают модульность и разделение задач. Каждый микросервис представляет собой определенную бизнес-функцию и взаимодействует с другими сервисами через четко определенные API. Одним из ключевых принципов архитектуры микросервисов является ограниченный контекст, который подчеркивает необходимость определения четких границ для каждого сервиса на основе его функциональности. Это позволяет командам работать автономно над отдельными сервисами, не мешая работе друг друга. Кроме того, микросервисы способствуют сохранению многоязычия, позволяя командам выбирать наиболее подходящую технологию хранения данных для каждой службы.

В данной программе помимо функциональных сервисов присутствует Eureka Server — служба обнаружения для имеющихся микросервисов. Клиентские приложения могут самостоятельно регистрироваться в нем, а другие микросервисы могут обращаться к Eureka Server для поиска необходимых им микросервисов (Рис. 8).

Также присутствует API Gateway. Gateway API — инструмент управления API между клиентом и набором внутренних служб. Он действует как обратный прокси-сервер для приёма вызовов API, агрегирования различных служб, которые нужны для выполнения запросов, а затем возвращает результат. API-шлюз, принимающий запрос пользователя приложения, направляет его к одной или нескольким серверным службам, собирает соответствующие данные и доставляет их пользователю в виде единого комбинированного пакета. Он также обеспечивает аналитику, защиту от угроз и другие средства безопасности для приложения. Основная роль Gateway API в том, чтобы действовать в качестве единой точки входа и стандартизированного процесса для взаимодействия между приложениями, данными и службами организации и внутренними и внешними клиентами (Рис. 9).

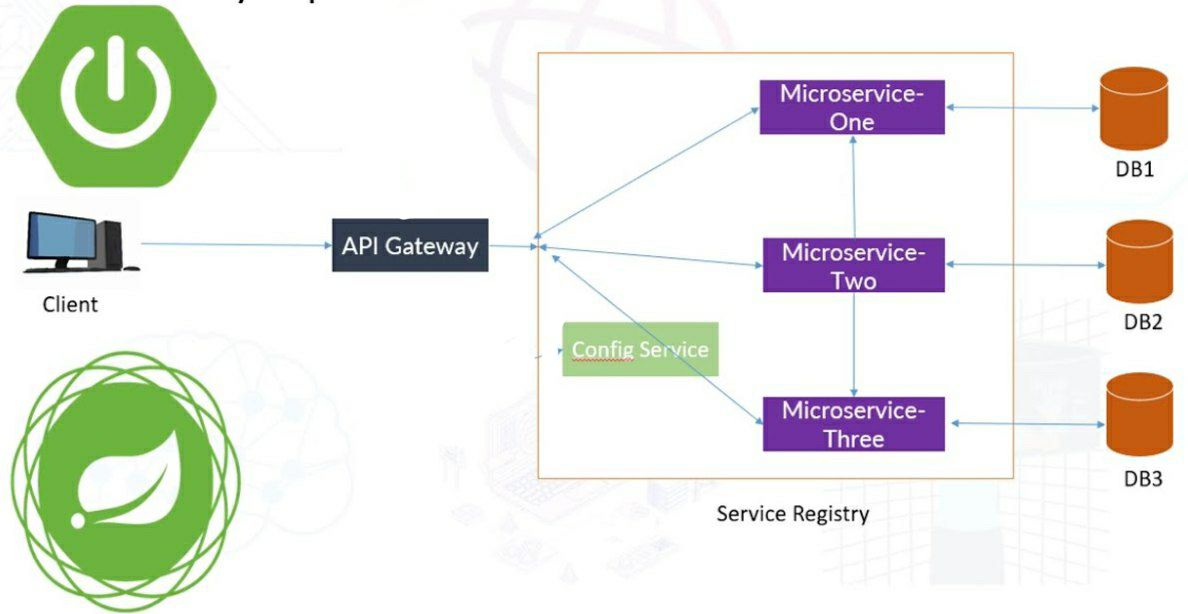


Рис. 7 – Пример микросервисной ахитектуры

Использование микросервисной архитектуры в разработке программы позволило обеспечить высокую гибкость, масштабируемость и независимость отдельных компонентов. Благодаря ограниченному контексту и автономному развертыванию сервисов, команды разработчиков могут работать более эффективно, сокращая время разработки и повышая качество продукта. Eureka Server и API Gateway, встроенные в архитектуру, обеспечивают стабильность и безопасность, а также упрощают взаимодействие между службами. В целом, микросервисная архитектура является эффективным решением для современных программных проектов, позволяющим создавать комплексные и динамичные приложения.

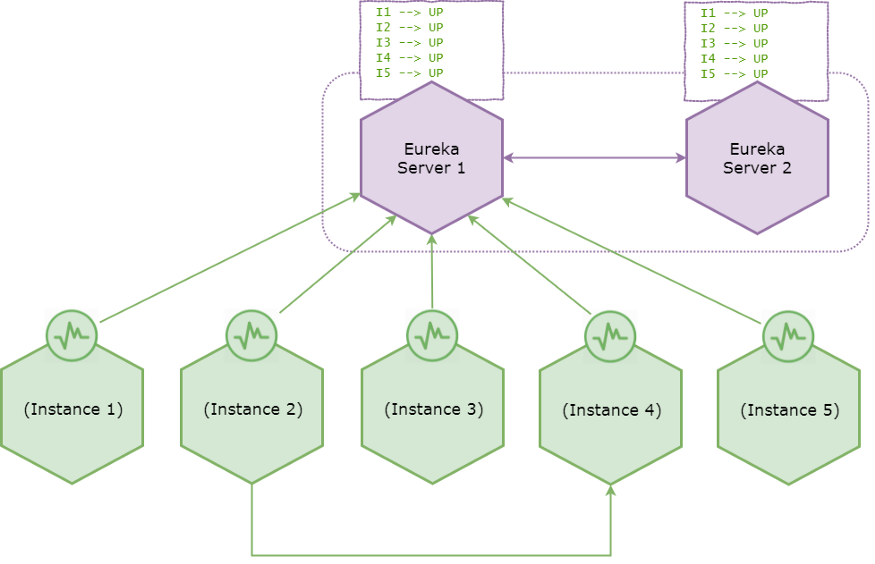


Рис. 8 – Архитектура Eureka Server

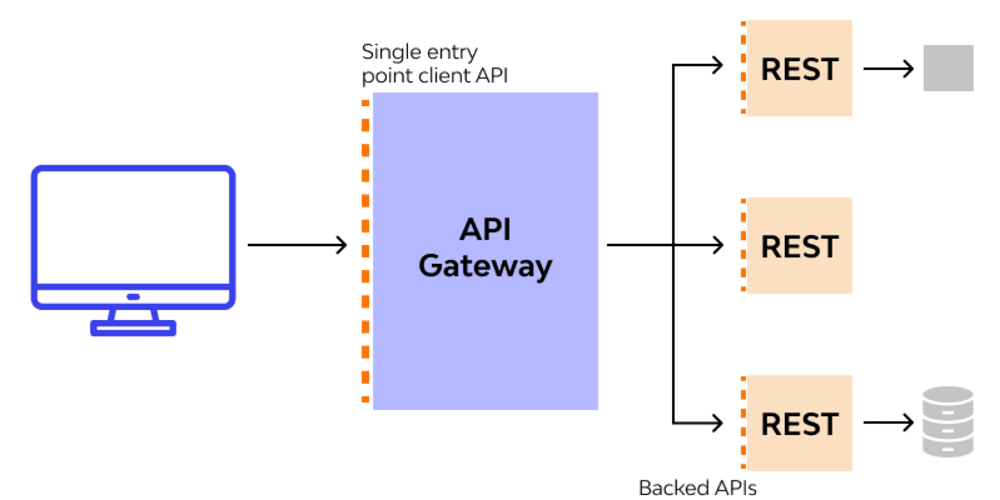


Рис. 9 – Архитектура API Gateway

**Глава 2. Реализация проекта с базовыми функциями социальной сети.**

2.1 Описание конфигурационных файлов проекта.

Properties файлы в проекте Spring используются для хранения конфигурационных параметров, таких как настройки базы данных, URL-адреса API, параметры безопасности и другие свойства. Эти файлы обычно имеют расширение “.properties” и представляют собой простой текстовый файл в формате "ключ=значение". Однако в данной работе используются конфигурационные файлы с разрешением .yml (YAML Ain't Markup Language). Среди его особенностей можно выделить:

1. Читаемость: YAML файлы легче читать и писать, чем XML или properties файлы, благодаря своей простой и интуитивно понятной структуре
2. Иерархическая структура: YAML поддерживает иерархическую структуру данных с помощью отступов, что делает его удобным для представления сложных конфигураций.
3. Поддержка списков и словарей: YAML позволяет определять списки и словари значений, что упрощает представление коллекций данных в конфигурационных файлах.
4. Внедрение значений: В Spring Framework можно использовать специальные аннотации (например, @Value) для внедрения значений из YAML файлов в бины приложения.
5. Профили: YAML файлы могут быть использованы для определения профилей приложений (например, dev, prod) и активации соответствующих конфигураций.
6. Возможность комментирования: YAML позволяет добавлять комментарии к конфигурационным данным, что облегчает понимание и поддержку кода.
7. Поддержка функций Spring: В YAML файлах можно использовать специальные функции Spring Expression Language (SpEL) для более гибкой конфигурации.
8. Использование во многих фреймворках: YAML формат широко используется не только в Spring Framework, но и в других фреймворках и инструментах разработки.

Использование YAML формата для конфигурации приложений в Spring Framework помогает сделать конфигурацию более читаемой, гибкой и удобной для разработчиков. Поэтому, проекте используется формат YAML представленный ниже для описания конфигурации проекта.

Eureka Server (Рис. 10).

Конфигурационный файл application.yml содержит параметры настройки для приложения Spring Boot, которое является сервером Eureka. Сервер Eureka используется для управления и регистрации микросервисов в распределенной системе.

Структура.

Конфигурационный файл состоит из трех основных разделов:

1. server: port: Устанавливает порт, на котором будет работать сервер Eureka (по умолчанию 8761).

2. spring: application: name: Имя приложения Spring Boot (в данном случае "EurekaServer").

3. eureka: client: register-with-eureka: Определяет, должен ли сервер Eureka регистрироваться в другом сервере Eureka (установлено значение "false", поскольку этот сервер является основным).

eureka: client: fetch-registry: Определяет, должен ли сервер Eureka запрашивать реестр у других серверов Eureka (установлено значение "false").

eureka: client: service-url: defaultZone: Устанавливает URL-адрес для доступа к другим серверам Eureka (в данном случае установлен тот же URL, поскольку этот сервер является основным).

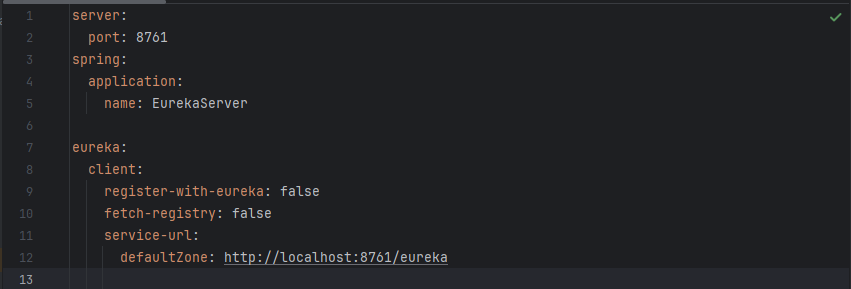


Рис. 10 – Конфигурационный файл Eureka Server

API Gateway (Рис. 11).

Конфигурационный файл application.yml содержит параметры настройки для приложения Spring Boot, которое является шлюзом API. Шлюз API действует как единая точка входа для клиентов и маршрутизирует запросы к соответствующим микросервисам.

Структура.

Конфигурационный файл состоит из трех основных разделов.

1. server: port: Устанавливает порт, на котором будет работать шлюз API (по умолчанию 8080). В данном случае установлен порт 8765.

2. spring: application: name: Имя приложения Spring Boot (в данном случае "Gateway").

spring: cloud: gateway: routes: Определяет маршруты, которые шлюз API будет использовать для перенаправления запросов.

В данном случае определены три маршрута:

Profile: Маршрутизирует запросы, начинающиеся с "/profile/", к микросервису, работающему по адресу "http://localhost:8081/".

Server: Маршрутизирует запросы, начинающиеся с "/server/", к микросервису, работающему по адресу "http://localhost:8082/".

Profile1: Маршрутизирует запросы, начинающиеся с "/profile1/", к микросервису, работающему по адресу "http://localhost:8083/".

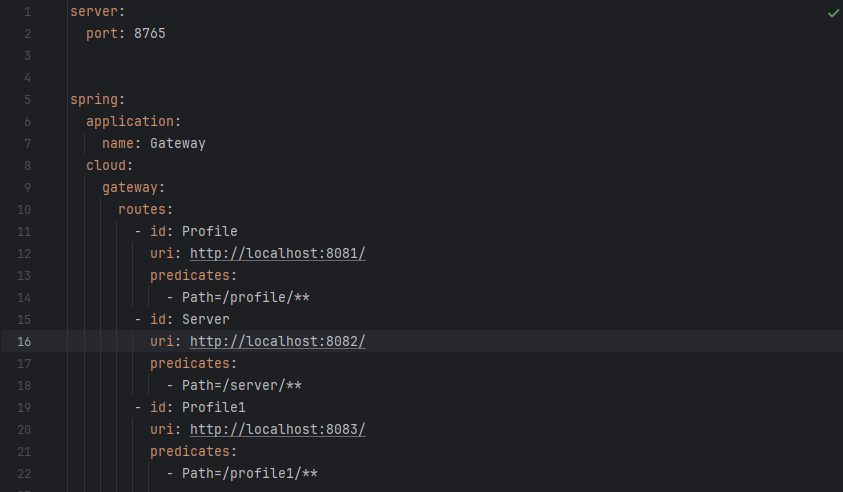


Рис. 11 – Конфигурационный файл API Gateway

Server (Рис. 12).

Конфигурационный файл `application.yml` содержит параметры настройки для приложения Spring Boot, которое является микросервисом. В данном случае микросервис предоставляет функциональность, связанную с профилями пользователей.

Структура.

Конфигурационный файл состоит из нескольких разделов.

1. server: port: Устанавливает порт, на котором будет работать микросервис (по умолчанию 8080). В данном случае установлен порт 8081.

2. spring: application: name: Имя приложения Spring Boot (в данном случае "Profile").

3. jpa: database: Устанавливает используемую базу данных (в данном случае PostgreSQL).

jpa: show-sql: Выводит в консоль SQL-запросы, выполняемые приложением.

jpa: hibernate: ddl-auto: Устанавливает стратегию управления схемой базы данных (в данном случае "update", что означает обновление схемы при необходимости).

datasource: Настраивает основное соединение с базой данных:

datasource: platform: Устанавливает платформу базы данных (в данном случае PostgreSQL).

datasource: url: Устанавливает адрес подключения к базе данных.

datasource: username: Имя пользователя базы данных.

datasource: password: Пароль пользователя базы данных.

datasource: driverClassName: Устанавливает драйвер для подключения к базе данных.

4.second-datasource: Настраивает дополнительное соединение с базой данных (в данном случае для другой базы данных с сообщениями):

second-datasource: platform: Устанавливает платформу базы данных (в данном случае PostgreSQL).

second-datasource: url: Устанавливает адрес подключения к базе данных.

second-datasource: username: Имя пользователя базы данных.

second-datasource: password: Пароль пользователя базы данных.

second-datasource: driverClassName: Устанавливает драйвер для подключения к базе данных.

5. security.enable-csrf: Отключает защиту от межсайтовой подделки запросов (CSRF) для упрощения тестирования.

6. logging: Настраивает уровень ведения журнала:

Устанавливает уровень ведения журнала для пакета `org.springframework.security.web` на "INFO", что означает вывод в консоль информационных сообщений.

У микросервиса Profile также есть конфигурационный файл с

расширением .yml, но существенных различий между ним и конфигурационным файлом микросервиса Server нет.

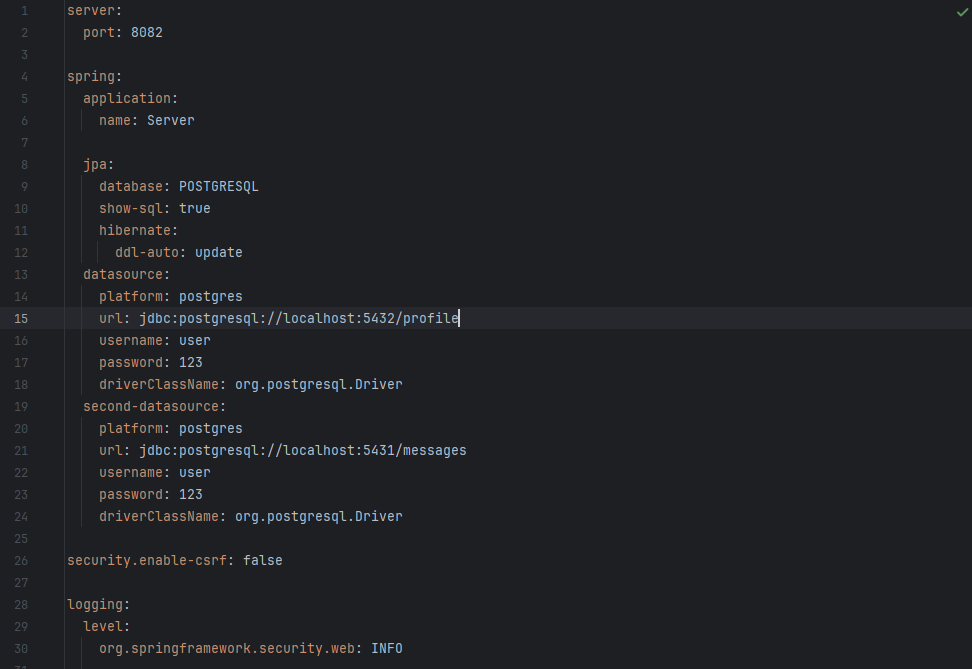


Рис. 12 – Конфигурационный файл Server

2.2 Описание компонентов программы

Основной функционал программы состоит в микросервисах Server и

Profile.

Микросервис Profile состоит из нескольких основных элементов, хранящихся в пакетах (Рис. 13).

* Domain: класс Profile, класс Message.
* Dto: класс TransferRequest.
* Repository: интерфейс MessageRepository, интерфейс

ProfileRepository.

* Service: класс ProfileService.
* Controller: класс ProfileController.
* Config: класс SecurityConfiguration.
* Aspects: класс LogAspect, аннотация TrackUserAnnotation.

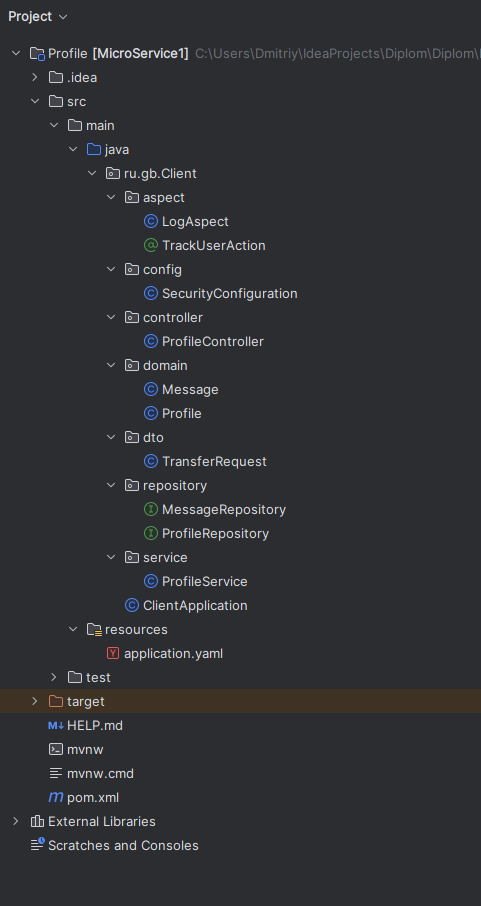


Рис. 13 – Структура микросервиса Profile

Пакет domain.

Класс Profile представляет собой сущность для хранения информации о пользователях. Он сопоставлен с таблицей "profiles" в базе данных. Поля аннотированы соответствующим образом, чтобы JPA мог выполнять сопоставление. Метод getAge() извлекает дату рождения и вычисляет возраст профиля на основе текущего года.

Класс Message также представляет собой сущность, но уже для хранения сообщений между пользователями. Он сопоставлен с таблицей "messages" в базе данных. Поля также аннотированы соответствующим образом, чтобы JPA мог выполнять сопоставление. Этот класс не содержит каких-либо дополнительных методов или логики, кроме автоматически сгенерированных методов геттера и сеттера.

Пакет dto.

Пакет dto (data transfer object) содержит класс TransferRequest, который используется для представления и передачи данных о запросе отправки сообщения между отправителем и получателем.

В нём содержатся 3 поля.

* senderAccountID – идентификатор отправителя типа Long.
* receiverAccountID – идентификатор получателя типа Long.
* message – текст самого сообщения типа String.

Пакет repository (интерфейсы баз данных).

Интерфейс ProfileRepository расширяет JpaRepository<Profile, Long>,

который предоставляет общие методы для работы с сущностью Profile и ее первичным ключом, которым является id профиля типа Long.

JpaRepository - это интерфейс из фреймворка Spring Data JPA, который предоставляет удобные методы для работы с базой данных через JPA (Java Persistence API). Он представляет собой часть абстракции репозитория в Spring Data, которая упрощает доступ к данным из базы данных и уменьшает объем необходимого кода. Предоставляет общие методы для выполнения CRUD-операций (создание, чтение, обновление, удаление) и другие полезные методы, такие как findById и findAll.

Интерфейс MessageRepository также расширяет JpaRepository

<Message, Long>, который предоставляет общие методы для работы с сущностью Message и ее первичным ключом, которым является id сообщения типа Long.

Эти интерфейсы позволяют работать с сущностями Profile и Message в базе данных с помощью простых методов, похожих на методы Java. Они абстрагируют от технических деталей взаимодействия с базой данных, таких как написание SQL-запросов.

Пакет service.

Пакет содержит класс ProfileService, в котором реализована основная функциональная нагрузка мироксервиса. В нём находятся методы для взаимодействия с сообщениями: отправка, прочтение и так далее, методы фильтрации профилей и другие.

Пакет controller.

Пакет содержит класс ProfileController, который представляет собой REST-контроллер с инъекцией зависимости сервиса ProfileService, предназначенный для обработки запросов, связанных с управлением профилями пользователей в проекте, используя методы из инъектированного класса.

Пакет config.

Пакет содержит класс SecurityConfiguration, который является классом конфигурации для настройки безопасности Spring Security в приложении. В классе содержится метод filterChain, который настраивает цепочку фильтров безопасности, отключает защиту CSRF, авторизует запросы на основе ролей и использует базовую аутентификацию HTTP. Метод passwordEncoder, который создает и возвращает экземпляр PasswordEncoder. Он использует PasswordEncoderFactories.createDelegatingPasswordEncoder() для создания делегирующего кодировщика паролей, который поддерживает несколько алгоритмов хэширования. А также метод inMemoryUserDetailsManager создает и возвращает менеджер пользователей в памяти. Он создает одного пользователя с именем "user", паролем "user" (без хэширования) и ролью "USER".

Пакет aspect.

Пакет aspect содержит класс LogAspect и класс-аннотацию

TrackUserAnnotation. Класс LogAspect используется для добавления функциональности логирования в приложение на основе Spring. Он позволяет централизованно перехватывать и регистрировать вызовы методов, аннотированных @TrackUserAction, что может быть полезно для отслеживания действий пользователей, аудита и отладки. Аннотация `@TrackUserAction` предназначена для обозначения методов, которые должны быть перехвачены и зарегистрированы аспектом `LogAspect`. Когда метод, аннотированный `@TrackUserAction`, вызывается, аспект `LogAspect` перехватывает его и регистрирует информацию о вызове, такую как время вызова, имя метода, переданные параметры и роль пользователя.

Параметры аннотации.

* @Retention(RetentionPolicy.RUNTIME) указывает, что аннотация должна сохраняться во время выполнения программы. Это позволяет получить доступ к информации об аннотации во время выполнения с помощью рефлексии.
* @Target(ElementType.METHOD) указывает, что аннотация может применяться только к методам.

Микросервис Server состоит из нескольких основных элементов, хранящихся в пакетах (Рис. 14).

* Domain: класс Profile, класс Message.
* Dto: класс TransferRequest.
* Repository: интерфейс MessageRepository, интерфейс

ServerRepository.

* Service: класс ServerService.
* Controller: класс ServerController.
* Config: класс SecurityConfiguration.
* Aspects: класс LogAspect, аннотация TrackUserAnnotation.

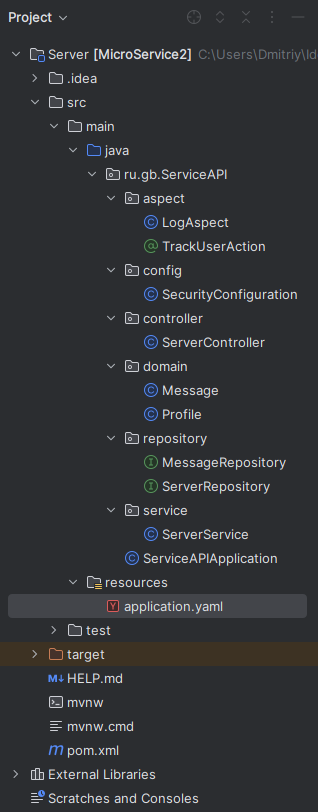


Рис. 14 – Структура микросервиса Server

Пакет domain:

Класс Profile представляет собой сущность для хранения информации о пользователях. Он сопоставлен с таблицей "profiles" в базе данных. Поля аннотированы соответствующим образом, чтобы JPA мог выполнять сопоставление. Метод getAge() извлекает дату рождения и вычисляет возраст профиля на основе текущего года.

Класс Message также представляет собой сущность, но уже для хранения сообщений между пользователями. Он сопоставлен с таблицей "messages" в базе данных. Поля также аннотированы соответствующим образом, чтобы JPA мог выполнять сопоставление. Этот класс не содержит каких-либо дополнительных методов или логики, кроме автоматически сгенерированных методов геттера и сеттера.

Пакет dto.

Пакет dto (data transfer object) содержит класс TransferRequest, который используется для представления и передачи данных о запросе отправки сообщения между отправителем и получателем.

В нём содержатся 3 поля.

* senderAccountID – идентификатор отправителя типа Long.
* receiverAccountID – идентификатор получателя типа Long.
* message – текст самого сообщения типа String.

Пакет repository (интерфейсы баз данных).

Интерфейс ServerRepository расширяет JpaRepository<Profile, Long>, который предоставляет общие методы для работы с сущностью Profile и ее первичным ключом, которым является id профиля типа Long.

JpaRepository - это интерфейс из фреймворка Spring Data JPA, который предоставляет удобные методы для работы с базой данных через JPA (Java Persistence API). Он представляет собой часть абстракции репозитория в Spring Data, которая упрощает доступ к данным из базы данных и уменьшает объем необходимого кода. Предоставляет общие методы для выполнения CRUD-операций (создание, чтение, обновление, удаление) и другие полезные методы, такие как findById и findAll.

Интерфейс MessageRepository также расширяет JpaRepository

<Message, Long>, который предоставляет общие методы для работы с сущностью Message и ее первичным ключом, которым является id сообщения типа Long.

Эти интерфейсы позволяют работать с сущностями Profile и Message в базе данных с помощью простых методов, похожих на методы Java. Они абстрагируют от технических деталей взаимодействия с базой данных, таких как написание SQL-запросов.

Пакет service.

Пакет содержит класс ServerService, в котором реализована основная функциональная нагрузка мироксервиса. В нём находятся методы создания, редактирования и удаления профилей, методы для взаимодействия с сообщениями: отправка, прочтение и так далее, методы фильтрации профилей и другие.

Пакет controller.

Пакет содержит класс ServerController, который представляет собой REST-контроллер с инъекцией зависимости сервиса ServerService, предназначенный для обработки запросов, связанных с управлением профилями пользователей в проекте, используя методы из инъектированного класса.

Пакет config.

Пакет содержит класс SecurityConfiguration, который является классом конфигурации для настройки безопасности Spring Security в приложении. В классе содержится метод filterChain, который настраивает цепочку фильтров безопасности, отключает защиту CSRF, авторизует запросы на основе ролей и использует базовую аутентификацию HTTP. Метод passwordEncoder, который создает и возвращает экземпляр PasswordEncoder. Он использует PasswordEncoderFactories.createDelegatingPasswordEncoder() для создания делегирующего кодировщика паролей, который поддерживает несколько алгоритмов хэширования. А также метод inMemoryUserDetailsManager создает и возвращает менеджер пользователей в памяти. Он создает одного пользователя с именем "user", паролем "user" (без хэширования) и ролью "USER".

Пакет aspect.

Пакет aspect содержит класс LogAspect и класс-аннотацию

TrackUserAnnotation. Класс LogAspect используется для добавления функциональности логирования в приложение на основе Spring. Он позволяет централизованно перехватывать и регистрировать вызовы методов, аннотированных @TrackUserAction, что может быть полезно для отслеживания действий пользователей, аудита и отладки. Аннотация @TrackUserAction предназначена для обозначения методов, которые должны быть перехвачены и зарегистрированы аспектом LogAspect. Когда метод, аннотированный @TrackUserAction, вызывается, аспект LogAspect перехватывает его и регистрирует информацию о вызове, такую как время вызова, имя метода, переданные параметры и роль пользователя.

Параметры аннотации.

* @Retention(RetentionPolicy.RUNTIME) указывает, что аннотация должна сохраняться во время выполнения программы. Это позволяет получить доступ к информации об аннотации во время выполнения с помощью рефлексии.
* @Target(ElementType.METHOD) указывает, что аннотация может применяться только к методам.

2.3 Проведение интеграционного и модульного тестирования.

Модульное тестирование – это инструмент разработки, позволяющий создавать стабильный, надежный и легко поддерживаемый код. Оно подразумевает тестирование отдельных модулей (функций, классов, компонентов) программного обеспечения в изоляции от остальных частей системы.

Преимущества модульного тестирования.

Раннее выявление ошибок. Проверка каждого модуля на ранних этапах разработки позволяет обнаружить и исправить ошибки до того, как они повлияют на другие части системы.

Улучшение качества кода. Тесты служат документацией и демонстрируют, как должен работать каждый модуль.

Упрощение рефакторинга. Тесты гарантируют, что изменения в коде не нарушат функциональность уже существующих модулей. Повышение уверенности в коде: Знание того, что код прошел модульное тестирование, позволяет разработчикам внедрять изменения с меньшими опасениями.

Упрощение интеграции. Модульное тестирование облегчает интеграцию отдельных модулей в единую систему, гарантируя их совместимость.

Методы модульного тестирования.

Тестирование черного ящика. Тестирование функциональности модуля без знания его внутренней реализации.

Тестирование белого ящика. Тестирование, которое включает в себя анализ внутренней реализации модуля.

Тестирование серого ящика. Комбинация тестирования черного и белого ящика.

Фреймворки для модульного тестирования.

JUnit (Java). Популярный фреймворк для написания модульных тестов на языке Java.

PyTest (Python). Мощный и гибкий фреймворк для Python, предоставляющий широкий набор возможностей для модульного тестирования.

Jest (JavaScript). Популярный фреймворк для тестирования JavaScript-кода, известный своей скоростью и удобством использования.

Модульное тестирование – это ключевой элемент разработки качественного и надежного программного обеспечения. Оно позволяет создавать код, который легко поддерживать, развивать и интегрировать, а также значительно сокращает время и ресурсы, необходимые для исправления ошибок.

Интеграционное тестирование - это важный этап тестирования программного обеспечения, который проверяет взаимодействие между различными компонентами системы. Он фокусируется на том, как модули, сервисы и другие части системы работают вместе, обеспечивая правильное взаимодействие данных, потоков управления и общего функционала.

Цель интеграционного тестирования - выявить ошибки, связанные с интеграцией компонентов. Ниже приведены примеры.

Несовместимость интерфейсов. Разные компоненты могут иметь разные форматы данных или протоколы связи, что приводит к ошибкам при обмене информацией.

Проблемы с синхронизацией. Компоненты могут зависеть друг от друга, и ошибки в синхронизации могут привести к некорректным результатам.

Несоответствие данных. Проблемы с переходом данных между компонентами, например, дублирование или потеря информации.

Некорректное управление ресурсами. Компоненты могут использовать общие ресурсы (например, базу данных), и ошибки в их управлении могут привести к конфликтам.

Методы интеграционного тестирования.

Пошаговая интеграция. Компоненты добавляются к системе по одному, постепенно расширяя тестовую среду.

Интеграция сверху вниз. Начинается с тестирования верхнего уровня системы, постепенно переходя к нижним уровням.

Интеграция снизу-вверх. Начинается с тестирования нижнего уровня системы, постепенно переходя к верхним уровням.

Биг-банг интеграция. Все компоненты системы объединяются одновременно для тестирования.

Инструменты для интеграционного тестирования.

Системы управления версиями. Для отслеживания изменений кода и конфигурации.

Инструменты для автоматизации тестирования. Для создания и запуска автоматических тестовых сценариев.

Mock-сервисы. Для имитации поведения внешних систем и сервисов. Системы мониторинга и отладки: Для анализа производительности и выявления ошибок.

Преимущества интеграционного тестирования.

Раннее обнаружение ошибок. Ошибки, связанные с взаимодействием компонентов, обнаруживаются на раннем этапе разработки.

Повышение качества системы. Обеспечивает стабильную и надежную работу всей системы.

Улучшение производительности. Выявляет проблемы с производительностью, возникающие при взаимодействии компонентов. Упрощение процесса разработки: Позволяет быстрее и эффективнее находить и исправлять ошибки.

Интеграционное тестирование является неотъемлемой частью процесса разработки программного обеспечения, обеспечивающей качество и стабильность системы. Важно выбрать подходящий метод и инструменты для эффективного проведения интеграционных тестов, чтобы гарантировать надежность и корректную работу всей системы.

Тесты необходимы для любого приложения. Они гарантируют, что изменения, которые вносятся в процессе разработки приложения, не нарушат уже существующие функции. Кроме того, тесты играют роль документации.

Тесты пишут для уверенности, что логика определенного метода, реализованного в проекте, работает так, как предполагалось. При тестировании конкретного метода обычно требуется проверить несколько сценариев. Для каждого такого сценария пишется тестовый метод в тестовом классе. В проектах Maven тестовые классы размещаются в папке test.

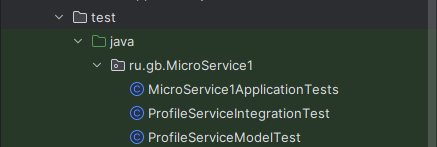


Рис. 15 – Структура тестов микросервиса Profile

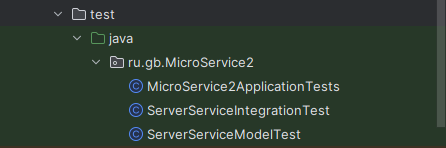


Рис. 16 – Структура тестов микросервиса Server

**Заключение**

В ходе выполнения данной дипломной работы была разработана модель социальной сети, которая демонстрирует ключевые принципы их функционирования.

Разработанная программа позволяет пользователям отправлять, читать и фильтровать сообщения, а также изменять собственный профиль и удобно находить чужие. Благодаря использованным технологиям, программа обеспечивает более эффективное взаимодействие между профилями, а также повышает удобство процесса использования.

В ходе практической реализации программы были проведены тестирование и отладка, что позволило достичь высокого качества и надежности работы бота. Анализ результатов практической реализации показал, что программа является эффективным инструментом оптимизации процессов обмена информацией, её обработки и принятия последующих решений, а также простого общения между людьми.

Разработка социальных сетей представляет собой актуальное и перспективное направление в области информационных технологий, способствуя повышению эффективности организации различных бизнес-процессов. Результаты исследования и практической реализации программы являются важным вкладом в развитие цифровой трансформации и современных методов управления передачи информации в интернете.

В заключении дипломной работы о разработке и реализации базовой модели социальной сети можно подчеркнуть следующее:

Данная работа представляет собой важный шаг в развитии современных технологий управления общением и передачи информации. Разработанная программа представляет собой инновационное решение, способствующее оптимизации процессов обмена информацией, её обработки и принятия последующих решений, а также простого общения между людьми.

Использование технологий социальных сетей и мессенджеров для открывает новые перспективы в области передачи информации, организации событий, принятия решений, обеспечивая более удобное взаимодействие между собеседниками. Анализ результатов практической реализации программы позволяет сделать вывод о её значимости для современного бизнеса и культурной сферы.

Этот проект не только демонстрирует возможности современных информационных технологий, но и подчеркивает важность инноваций в области социальных сетей. Результаты исследования и практической реализации программы являются важным вкладом в развитие цифровой трансформации и современных аспектов любой социальной сети.

Таким образом, разработка модели социальной сети представляет собой актуальное и перспективное направление в области информационных технологий, способствуя повышению эффективности передачи информации и простого общения людей между собой

В завершение, стоит отметить, что разработанная модель социальной сети имеет высокий потенциал для дальнейшего развития и совершенствования. В частности, можно рассмотреть возможность интеграции с другими сервисами и платформами, а также расширение функциональности за счет добавления новых возможностей и опций.

Итак, данная дипломная работа представляет собой важный шаг в развитии современных технологий управления социальными сетями, подчеркивающий важность инноваций и цифровой трансформации в этой области.

Цель работы достигнута.

**Список использованных источников**

1. Алпатов А.Н. Архитектура, проектирование и разработка программных средств [Электронный ресурс]: Учебное пособие / Алпатов А.Н., Рогов И.Е. — М.: МИРЭА – Российский технологический университет, 2023. — 107 c.
2. Аншина М.Л. Архитектура приложений и данных: Учебное пособие / Аншина М.Л. — М.: МИРЭА — Российский технологический университет, 2024. — 15 с.
3. Баланов А. Н. Комплексное руководство по разработке: от мобильных приложений до веб-технологий : учебное пособие для вузов / А. Н. Баланов. — Санкт-Петербург : Лань, 2024. — 43 c.
4. Баланов А. Н. Построение микросервисной архитектуры и разработка высоконагруженных приложений : учебное пособие для СПО / А. Н. Баланов. — Санкт-Петербург : Лань, 2024. — 22 c.
5. Бауэр К., Кинг Г., Грегори Г. Б29 Java Persistence API и Hibernate / пер. с англ. Д. А. Зинкевича; под науч. ред. А. Н. Киселева. – М.: ДМК Пресс, 2017. – 44 с.
6. Головнин, Олег Константинович Г612 Введение в системное программирование и основы жизненного цикла системных программ: учебное пособие / О.К. Головнин, А.А. Столбова. – Самара: Изд-во Самарского университета, 2021. – 40 с.
7. Джон Карнелл, Иллари Уайлупо Санчес К21 Микросервисы Spring в действии / пер. с англ. А. Н. Киселева. – М.: ДМК Пресс, 2022. – 88 с.
8. Жматов Д.В. Системы управления реляционными и нереляционными базами данных: Учебное пособие / Жматов Д.В. — М.: МИРЭА – Российский технологический университет, 2023. — 61 c.
9. Импортозамещающие технологии обеспечения информационной безопасности и защиты данных : учебное пособие / Д.А. Короченцев, Л.В. Черкесова, Е.А. Ревякина [и др.] ; под общей редакцией Л.В. Черкесовой ; Донской государственный технический университет. – Ростов-на-Дону : ДГТУ, 2021. – 229 с.
10. Ксенофонтов Н.В. Кроссплатформенная среда исполнения программного обеспечения: Учебно-методическое пособие / Ксенофонтов Н.В., Иванова И.А., Котилевец И.Д. — М.: МИРЭА – Российский технологический университет, 2023. — 85 с.
11. Лёзин, Илья Александрович Л414 Разработка веб-приложений с использованием Spring Boot: учебное пособие / И.А. Лёзин, И.В. Лёзина. – Самара: Издательство Самарcкого университета, 2023. – 48 с.
12. Старолетов С. М. Основы тестирования программного обеспечения : учебное пособие для СПО / С. М. Старолетов. — 4-е изд., стер. — СанктПетербург : Лань, 2024. — 53 с.
13. Теоретические основы реляционных баз данных: учеб. пособие/А.И.Баранчиков; Рязан. гос. радиотехн. ун-т. Рязань, 2023. 143 с.
14. Уоллс К. У63 Spring в действии. 6-е изд./ пер. с англ.А. Н. Киселева. – М.: ДМКПресс, 2022. – 308 с.
15. Oracle Corporation. Java SE 17 Documentation. [электронный ресурс] URL: <https://docs.oracle.com/en/java/javase/17/docs/> (дата обращения: 05.07.2023)
16. PostgreSQL: официальная документация по использованию PostgreSQL для работы с базами данных. [электронный ресурс] URL: <https://www.postgresql.org/docs/> (дата обращения: 05.07.2023)
17. Spring Boot и Docker: практическое руководство по разработке и развертыванию приложений Spring Boot с использованием Docker. [электронный ресурс] URL: <https://www.baeldung.com/spring-boot-docker> (дата обращения: 05.07.2023)
18. Документация Docker: официальное руководство по использованию Docker. [электронный ресурс] URL: <https://docs.docker.com/> (дата обращения: 05.07.2023)
19. Официальная документация Hibernate API [электронный ресурс] URL: <https://hibernate.org/orm/documentation/6.5/> (дата обращения: 05.07.2023)
20. Официальная документация Spring Framework. [электронный ресурс] URL: <https://docs.spring.io/spring-framework/reference/index.html> (дата обращения: 05.07.2023)