Geekbrains

**Реализация backend части транзакций на основе Spring фреймворка и базы данных MySQL**

Программа: Разработчик

Специализация: Веб-разработка на Java

Бруев Дмитрий Михайлович

https://github.com/masolik87/DiplomJava

г. Санкт-Петербург

2024 г.

Оглавление

[Введение 3](#_Toc161444661)

[Глава 1 Обзор архитектур и типов веб-приложений 5](#_Toc161444662)

[1.1 Онлайн архитектура. Модель клиент-сервер 6](#_Toc161444663)

[Глава 2 Исследование предметной области 9](#_Toc161444664)

[2.1 Описание предметной области 10](#_Toc161444665)

[2.1.1 Описание объекта автоматизации 10](#_Toc161444666)

[2.1.2 Описание рассматриваемого бизнес-процесса 11](#_Toc161444667)

[2.2 Описание выбранного средства разработки 12](#_Toc161444668)

[2.3 Заключение по разделу «Исследование предметной области» 21](#_Toc161444669)

[Глава 3 Проектирование и реализация веб-приложения 21](#_Toc161444670)

[3.1 Проектирование баз данных 21](#_Toc161444671)

[3.2 Инфраструктура информационной системы 22](#_Toc161444672)

[3.3 Реализация веб-приложения 23](#_Toc161444673)

[3.3.1 Конфигурирование и выстраивание архитектуры 23](#_Toc161444674)

[3.3.2 Реализация микросервиса ресурсов 24](#_Toc161444675)

[3.3.3 Реализация микросервиса транзакции 27](#_Toc161444676)

[3.4 Заключение по разделу «Проектирование и реализация веб-приложения» 28](#_Toc161444677)

[Заключение 29](#_Toc161444683)

[Список источников](#_Toc161444684) 30

# Введение

В современном мире существует множество веб-приложений, в связи с этим, растет спрос на разработчиков в сфере Веб. В наше время развитие информационных технологий приводит к тому, что в современном бизнесе необходимо использовать новейшие разработки и инновационные подходы для эффективного управления и развития компании. В связи с этим все больше предприятий начинают внедрять в свою деятельность информационные системы, которые помогают автоматизировать процессы управления и повышают эффективность бизнеса.

Одной из наиболее актуальных отраслей для внедрения информационных систем является сфера денежных переводов. Денежные переводы on-line являются неотъемлемой частью современного мира, ими можно оплатить услуги или товары. Отличным инструментом для разработки таких вэб-приложений являются веб-приложения на основе фреймворка Spring.

Целью данного дипломного проекта является разработка информационной системы обработки и транзакции с использованием фреймворка Spring. Данный проект позволит пользователям осуществлять переводы со счёта на счёт.

Для реализации данного проекта был выбран язык программирования Java, фреймворк Spring, база данных MySQL и другие технологии. Разработка велась с использованием принципов объектно-ориентированного программирования, шаблонов проектирования и лучших практик разработки программного обеспечения.

В ходе выполнения данной дипломной работы за все время обучения были определены следующие этапы для постепенной реализации веб-приложения:

* Проектирование структуры базы данных для хранения информации о пользователях, состоянии счёта и других сущностях.
* Настройка среды разработки и установка необходимых инструментов.
* Разработка бэкенд-части приложения с использованием фреймворка Spring.
* Реализация функционала для работы с пользователями, счетами и другими сущностями.
* Создание тестового веб-интерфейса для взаимодействия пользователей с приложением.
* .

Предлагаемый метод решения поставленных задач включает в себя:

* Анализ существующих решений.
* Литературный обзор технологий.
* Анализ необходимого функционала.
* Синтез частей веб-приложения.
* Разработка архитектуры веб-приложения.
* Разработка системы управления веб-приложения.
* Тестирование веб-приложения.

Данное веб-приложение может быть использовано как основа для создания собственной компании по осуществлению денежных переводов, а также как учебный проект для изучения работы с Java Spring и разработки веб-приложений.

# Глава 1 Обзор архитектур и типов веб-приложений

Для разработки веб-приложения необходимо определить, как внутреннюю архитектуру, так и архитектуру сети.

Внутренняя архитектура определяется с использованием шаблонов проектирования приложений и включает логику и концепцию приложения.

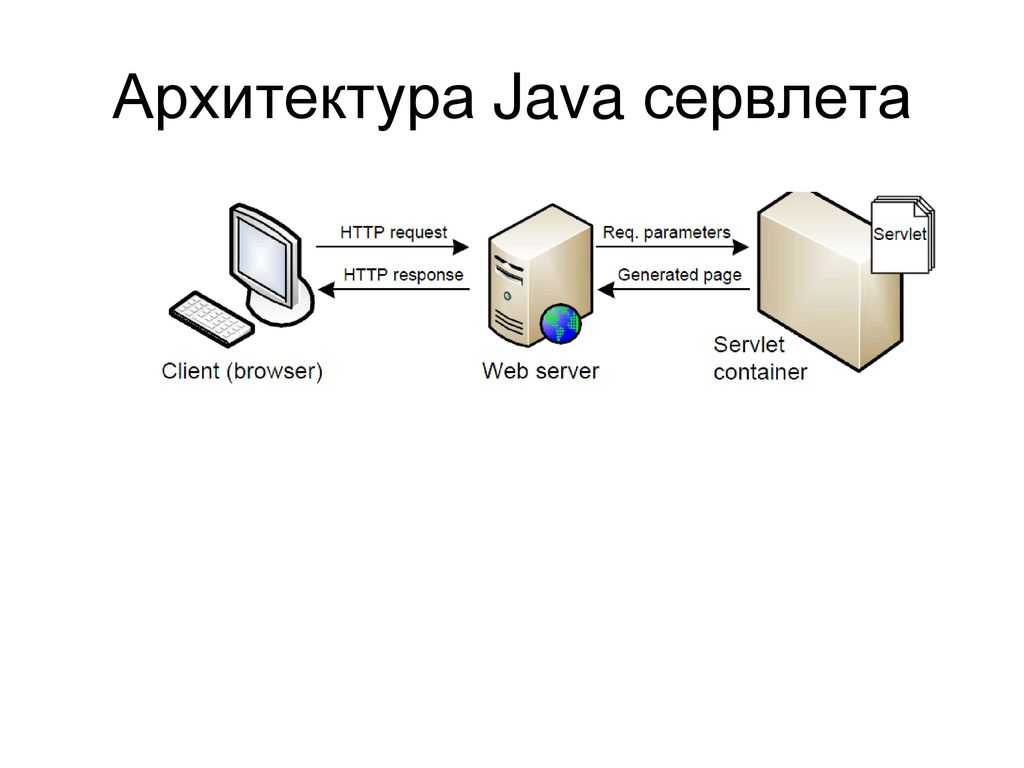
Сетевая архитектура определяет различные аспекты взаимодействия между компонентами приложения. Большое количество существующих исследований и реализаций различают два основных типа:

* Онлайн архитектура.
* Облачная архитектура.

## 1.1 Онлайн архитектура. Модель клиент-сервер

Онлайн архитектура использует серверы только для обработки логики и хранения данных. Как показано на рисунке 1, клиент должен выполнять задачи хранения метаданных, обработки и выполнения запросов, рендеринга страницы и захвата действий пользователя. Обмен данными между этими двумя экземплярами основан на данных приложения, что означает доставку небольшого трафика с высокой надежностью и малой задержкой [1].

Рисунок 1 ‑ Онлайн архитектура



В литературе описывается только одна модель для реализации этой архитектуры – клиент-сервер.

В области информационных технологий клиент-сервер представляет собой модель сетевой архитектуры, которая включает клиентские системы и серверные системы. Клиент-серверное приложение ‑ категория распределенной системы, состоящая как из клиентского, так и из серверного программного обеспечения. Во время работы сервер постоянно ожидает запросы от любого клиента, а клиентский процесс при необходимости запускает соединение с сервером. Клиент ‑ это компьютерное аппаратное устройство с программным обеспечением, которое обращается к сервису, предоставляемому сервером. Сервер ‑ компьютер, на котором запущено специальное программное обеспечение, которое предоставляет услуги для нужд других машин.

По мнению авторов, модель взаимодействия клиент-сервер делится на четыре типа:

1. Одноуровневая архитектура.
2. Двухуровневая архитектура.
3. Трехуровневая архитектура.
4. Многоуровневая архитектура.

Каждый из типов реализует основные принципы модели архитектуры клиент-сервер, а затем расширяется до *N*-уровня.

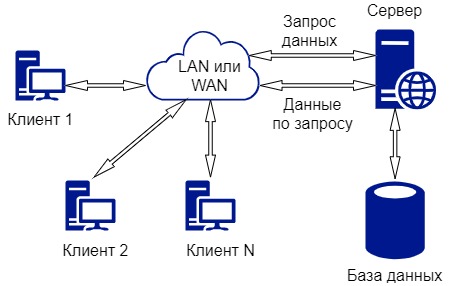
Как показано на рисунке 2, клиент отображает только информацию, предоставленную сервером, а сервер выполняет все необходимые расчеты.

В качестве примера использования этой модели автор использует SPA (одностраничное приложение), доступ с терминала к удаленному серверу и удаленному рабочему столу.

Рисунок 2 ‑ Одноуровневая архитектура

Двухуровневая архитектура. Архитектура приложения делится на две части: клиентское приложение (клиентский уровень) и база данных (уровень данных) [2]. Клиентская сторона выполняет задачи по визуализации полученных данных и реализации логики приложения, а сервер отвечает за управление базой данных (рисунок 3).

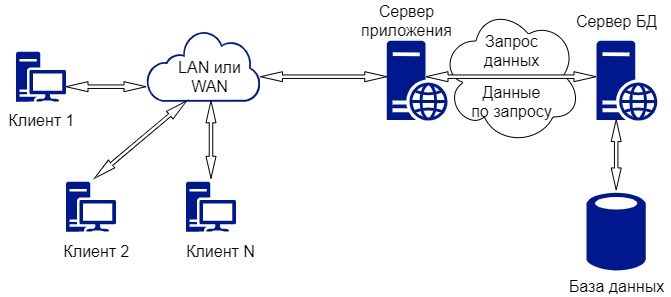
Рисунок 3 ‑ Двухуровневая архитектура



Клиентское устройство отправляет запрос на сервер, а сервер обрабатывает запрос и отправляет данные обратно в клиентскую систему.

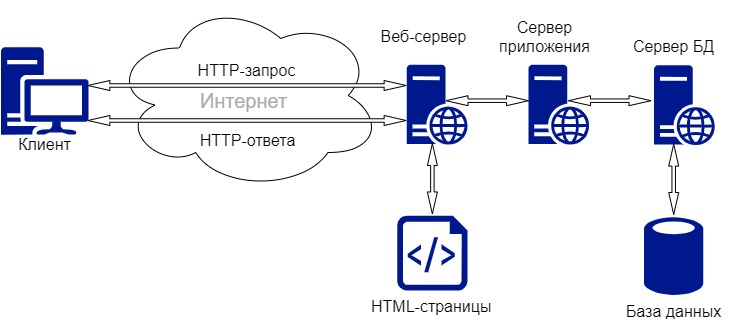
Как показано на рисунке 4, трехуровневая архитектура включает добавление сервера, который обрабатывает логику приложения, в то время как клиент выполняет рендеринг страниц и работает с запросами, при этом хранит метаданные. Сервер менеджера баз данных также присутствует в этой архитектуре.

Рисунок 4 ‑ Трехуровневая архитектура



Многоуровневая архитектура, аналогична трехуровневой (рисунок 5), но количество серверов приложений увеличено и представлено на отдельных уровнях для распределения бизнес-логики.

Рисунок 5 ‑ Многоуровневая архитектура



На основании обзора архитектуры можно сделать вывод, что каждый из этих типов, начиная с одноуровневой архитектура, может повысить надежность, безопасность и производительность сетевых приложений и сети в целом.

# Глава 2 Исследование предметной области

## 2.1 Описание предметной области

### 2.1.1 Описание объекта автоматизации

Конечными пользователями разрабатываемой веб-платформы являются разработчики клиентской части веб-приложения, которым предоставляется возможность подключения своих клиентов к серверам обработки логики работы веб-приложения и хранения данных, а также бэкенд разработчики микросервисов для интеграции своих микросервисов и управления уже предоставленных микросервисов.

Объектами автоматизации являются процессы использования функционала созданных микросервисов потребителем: процесс создания заказа, его резервирования и оплаты, логирование вызываемых действий и организация работы с базой данных

Данная система предполагает наличие следующих основных участников бизнес-процессов веб-платформы:

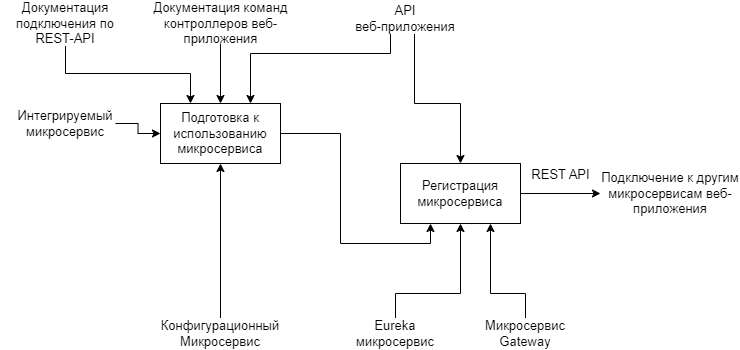
* Разработчик интегрируемых сервисов веб-приложения.
* Конечный потребитель, использующий функционал интегрируемого в веб-платформу приложения.

Опираясь на базовые потребности пользователей, веб-приложение должно предоставлять функционал универсальной интеграции клиентов и микросервисов. При этом, важным аспектом интеграции является отсутствие ограничений внутренней логики интегрируемых приложений для интеграции в разрабатываемое веб-приложение. Для работы с клиентами реализуется: функционал оформления заказа; функционал просмотра страницы; функционал просмотра товаров; возможность интеграции своего клиентского функционала.

### 2.1.2 Описание рассматриваемого бизнес-процесса

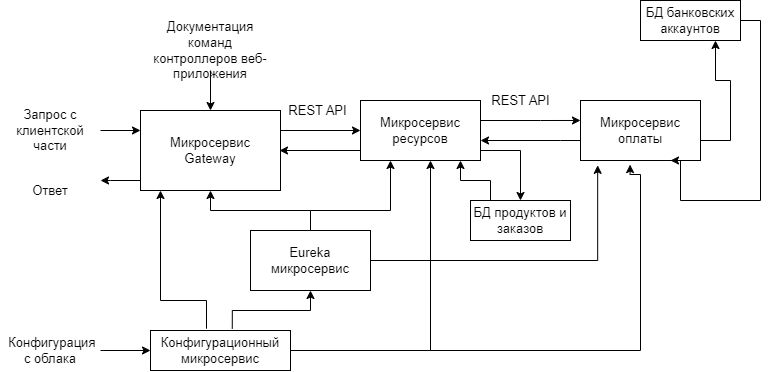
В данной работе главными рассматриваемыми бизнес-процессами являются: взаимодействие микросервисов веб-приложения друг с другом и возможность последующей интеграции микросервисов в архитектуру без наложения ограничений.

На рисунке 8 представлена диаграмма второго уровня, декомпозирующая диаграмму первого уровня.

  
Рисунок 8 ‑ Диаграмма второго уровня в нотации IDEF0

Декомпозиция главного бизнес-процесса показывает, что процесс использования приложения разбивается на два основных процесса: подготовка к использованию микросервисов; регистрация микросервисов. При этом, обмен данными между микросервисами осуществляется через REST API.

На рисунке 9 представлена диаграммы третьего уровня, декомпозирующая процесс работы с микросервисами.

  
Рисунок 9 ‑ Диаграмма третьего уровня в нотации IDEF0

Согласно реализуемой архитектуре, поступает запрос с клиентской части на микросервис Gateway. Gateway при старте, загружает конфигурацию и регистрируется как Eureka клиент. Согласно заголовку запроса, Gateway определяет, для какого сервиса был передан запрос и пересылает клиентский запрос. В реализованном веб-приложении, микросервис Gateway обменивается данными с клиентом и с микросервисом ресурсов, однако, архитектура предусматривает регистрацию других микросервисов (например, сервиса авторизации Spring Security [7]) через Gateway. Получив клиентский запрос, микросервис ресурсов производит обработку данных, заносит данные в базу данных и, если запрос предусматривает произведение оплаты, направляет запрос на микросервис оплаты, ожидая ответ от сервиса оплаты. Следует отметить, что микросервис ресурсов и оплаты тоже загружают конфигурацию с микросервиса конфигурации и регистрируются как Eureka клиенты.

## 2.2 Описание выбранного средства разработки

Java - это высокоуровневый, объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems. Java является одним из самых популярных языков программирования в мире благодаря своей платформе независимости и простоте использования.

Java позволяет разработчикам создавать различные типы приложений, включая веб-приложения, мобильные приложения, десктопные приложения и многое другое. Он имеет обширную библиотеку классов, которая облегчает разработку и ускоряет процесс написания кода.

Java выполняет код на виртуальной машине Java (JVM), что позволяет программам работать на различных операционных системах без необходимости перекомпиляции. Кроме того, Java обеспечивает высокий уровень безопасности и устойчивости приложений.

Этот язык программирования широко используется в корпоративной среде и в разработке больших проектов, так как он обладает хорошей масштабируемостью и производительностью. Java также позволяет создавать распределенные приложения, что делает его идеальным выбором для разработки многопользовательских систем.

Java Spring - это фреймворк для разработки веб-приложений на языке программирования Java. Он предлагает множество готовых компонентов и библиотек для упрощения создания, развертывания и управления веб-приложениями. Spring имеет модульную архитектуру, что позволяет разработчикам выбирать только те компоненты, которые им необходимы, и интегрировать их в свои проекты.

Java Spring предлагает различные инструменты для управления зависимостями, инверсии управления, аспектно-ориентированного программирования, создания RESTful веб-сервисов, обработки транзакций и многое другое. Он также обладает богатой документацией и активным сообществом разработчиков, что делает его одним из наиболее популярных фреймворков для разработки веб-приложений на языке Java.

За хранение данных отвечает MySQL. MySQL — это бесплатная и открытая система управления базами данных (СУБД), которая используется для хранения и управления большими объемами данных [10]. MySQL широко распространен и популярен среди разработчиков благодаря своей надежности, производительности и простоте в использовании. Он поддерживает множество функций и возможностей, таких как транзакции, хранимые процедуры, триггеры и многое другое. MySQL также имеет мощный язык запросов SQL, который позволяет легко извлекать и манипулировать данными в базе данных. Благодаря своей гибкости и расширяемости, MySQL может быть использован как для небольших локальных проектов, так и для крупных веб-приложений и корпоративных систем.

Spring Boot Starter Web является одним из стартеров в Spring Boot, который предоставляет все необходимые зависимости для создания веб-приложений. Этот стартер включает в себя все необходимые библиотеки и настройки для работы с веб-приложениями, такие как встроенный веб-сервер, обработка HTTP-запросов, обработка шаблонов и многое другое.

Spring Boot Starter Web позволяет разработчикам быстро создавать веб-приложения, избегая необходимости самостоятельно настраивать и подключать все необходимые зависимости. Кроме того, данный стартер предлагает простую и удобную конфигурацию веб-приложений, что упрощает разработку и ускоряет время развертывания.

Таким образом, Spring Boot Starter Web является отличным выбором для разработчиков, которые хотят создавать эффективные и мощные веб-приложения на основе Spring Boot.

Для манипуляций с данными, в проекте используется Spring Data и MySQL драйвер. Java MySQL Spring Data — это набор инструментов, который облегчает работу с базой данных MySQL в Java приложениях с использованием Spring Framework [12]. Spring Data позволяет упростить создание и выполнение запросов к базе данных, а также управление транзакциями.

С помощью Java MySQL Spring Data можно легко создавать сущности и их отношения в базе данных, выполнять CRUD операции (создание, чтение, обновление, удаление данных), работать с запросами на языке SQL и JPQL, использовать возможности транзакционного управления и многое другое.

Spring Data предоставляет аннотации и API для управления взаимодействием с базой данных, что позволяет разработчикам сосредоточиться на бизнес-логике приложения, минимизируя затраты на разработку и поддержку кода для доступа к данным.

Использование Java MySQL Spring Data делает разработку приложений на Java более эффективной, гибкой и удобной, позволяя создавать мощные и надежные приложения, работающие с базой данных MySQL. Spring Boot Starter Data JPA предоставляет удобные инструменты для работы с Java Persistence API (JPA) в проектах, основанных на Spring Boot. Эта зависимость включает в себя все необходимые библиотеки для подключения к базе данных, выполнения запросов и работы с объектно-реляционным отображением данных. Использование данной зависимости позволяет ускорить разработку приложений, связанных с базами данных, и сделать их более простыми в поддержке и масштабировании.

Spring Lombok — это библиотека для упрощения написания кода на языке Java при работе с фреймворком Spring. Она позволяет уменьшить количество шаблонного кода, который нужно писать для создания геттеров, сеттеров, конструкторов и других методов, благодаря использованию аннотаций.

Spring Lombok предоставляет различные аннотации, такие как @Getter, @Setter, @ToString, @NoArgsConstructor и многие другие, которые автоматически создают необходимые методы и уменьшают объем кода. Это делает код более чистым, читаемым и уменьшает вероятность ошибок при его написании.

Благодаря использованию Spring Lombok, разработчики могут более эффективно работать с фреймворком Spring, ускоряя процесс разработки и снижая объем написанного кода.

Для конфигурирования микросервисов применяется Spring Boot Configuration Processor и Spring Cloud Starter Config.

Spring Boot Configuration Processor - Spring Cloud Starter Config это инструмент, который позволяет генерировать метаданные для настройки во время компиляции в проектах на Spring Boot. Этот процессор автоматически собирает информацию о настройках приложения из Java классов и аннотаций, и создает файлы метаданных, которые будут использоваться при конфигурировании приложения.

Spring Boot Configuration Processor позволяет упростить процесс конфигурирования приложения и обеспечить более надежную работу за счет привязки метаданных к исходному коду. Этот инструмент позволяет избежать ошибок в конфигурации, так как компилятор проверяет правильность значений настроек на этапе сборки проекта.

В конечном итоге, использование Spring Boot Configuration Processor помогает разработчикам создавать более надежные и безопасные приложения за счет автоматизации процесса конфигурирования и уменьшения вероятности ошибок.

Spring Cloud Starter Config — это стартовый модуль для использования конфигураций в приложениях, разработанных с помощью Spring Cloud. Этот модуль позволяет централизованно управлять конфигурациями приложения внешним образом, путем использования распределенного конфигурационного сервера.

Spring Cloud Starter Config предоставляет возможность подключать внешние конфигурационные источники, такие как Git-репозитории (данный способ применяется в проекте) или директории файлов, и автоматически обновлять конфигурации без перезапуска приложения. Это обеспечивает гибкость и удобство в управлении конфигурациями приложения.

Кроме того, Spring Cloud Starter Config поддерживает шифрование конфиденциальных данных в конфигурациях, что обеспечивает безопасность конфиденциальной информации в приложениях.

Использование Spring Cloud Starter Config позволяет разработчикам управлять конфигурациями приложения более эффективно и безопасно, обеспечивая гибкость и надежность в работе с конфигурациями.

Java Spring Eureka - это инструмент, который представляет собой серверный реестр для микросервисов . Он позволяет микросервисам находить друг друга и обеспечивает им возможность взаимодействия. Eureka является частью Spring Cloud и предоставляет возможность для установки и запуска сервис-реестра в распределенной среде.

Eureka использует принцип клиент-сервер и работает на основе архитектуры RESTful. Когда микросервис запускается, он регистрируется в Eureka-сервере, который дает ему уникальный идентификатор. Другие микросервисы могут обращаться к Eureka-серверу, чтобы получить информацию о доступных сервисах и использовать их для взаимодействия.

Eureka обеспечивает хороший уровень отказоустойчивости и масштабируемости, поэтому он отлично подходит для использования в распределенных приложениях, работающих в среде микросервисов. Он также обладает возможностью обнаруживать и реагировать на изменения в сети, такие как добавление новых сервисов или отказ существующих.

Java Spring Eureka упрощает управление и обеспечивает прозрачное взаимодействие между микросервисами, что делает его важным инструментом для разработки современных приложений.

Java Spring Gateway - это инструмент, который используется для маршрутизации запросов от клиента к различным микросервисам [5]. Он представляет собой централизованный сервер, который принимает все запросы от клиентов и направляет их к соответствующим микросервисам на основе определенных правил и настроек.

Java Spring Gateway обеспечивает возможность настройки маршрутизации, аутентификации и авторизации запросов, логирования и мониторинга трафика. Он также может выполнять функции балансировки нагрузки и кеширования данных для улучшения производительности приложения.

Этот инструмент основан на фреймворке Spring, который предоставляет широкие возможности для разработки веб-приложений и микросервисов. Java Spring Gateway обеспечивает высокую отказоустойчивость и масштабируемость, что делает его отличным выбором для построения распределенных систем.

В целом, Java Spring Gateway помогает упростить и улучшить управление запросами и обеспечивает надежную и безопасную работу микросервисов в среде микросервисной архитектуры.

Spring Boot Starter AOP — это модуль Spring Boot, который предоставляет возможность использовать аспектно-ориентированное программирование (AOP) в ваших приложениях на Spring. AOP — это парадигма программирования, которая позволяет выделять повторяющиеся кросс-концерны (например, логирование, транзакции, безопасность) и предоставлять для них специальные аспекты, которые выполняются в определенных точках кода (советы).

Spring Boot Starter AOP позволяет легко настроить аспекты в ваших приложениях, а также предоставляет удобный доступ к различным функциям, таким как определение советов, объединение срезов, управление транзакциями и т.д. Этот модуль интегрируется с другими модулями Spring, такими как Spring Core и Spring Context, что делает его отличным выбором для разработки приложений на платформе Spring.

Spring Boot Starter AOP широко используется в различных областях, таких как веб-приложения, микросервисы, обработка данных и многое другое, где требуется управление кросс-концернами в приложениях. Он помогает улучшить модульность и читаемость кода, а также обеспечивает гибкость и масштабируемость разработки приложений на Spring.

Для мониторинга работы приложения может применяться Spring Boot Starter Actuator. Spring Boot Starter Actuator – это модуль, который предоставляет возможности мониторинга и управления приложением Spring Boot. Он содержит набор встроенных метрик, которые позволяют отслеживать различные аспекты работы приложения, такие как использование памяти, загрузка процессора, количество запросов и другие показатели. Кроме того, Actuator предоставляет различные эндпоинты, через которые можно получить информацию о приложении, изменить его конфигурацию, выполнить дополнительные действия и т.д. Этот модуль является полезным инструментом для мониторинга и управления приложением в реальном времени.

Micrometer-core — это библиотека для метрик, которая предоставляет простой способ собирать и измерять различные метрики в приложениях на Java. Она позволяет отслеживать различные аспекты работы приложения, такие как обработка запросов, использование ресурсов, состояние памяти и другие важные показатели. Micrometer-core предоставляет различные реализации репортеров для отправки метрик в различные системы мониторинга, такие как Prometheus, Graphite, InfluxDB и другие. Благодаря простому и гибкому интерфейсу библиотеки, разработчики могут легко интегрировать сбор метрик в свои приложения и получать ценную информацию о работе приложений в реальном времени.

Prometheus — это открытое программное обеспечение, разработанное компанией SoundCloud и позднее переданное в общественное достояние. Оно представляет собой систему мониторинга и сбора метрик, специально разработанную для контейнеризированных сред, таких как Docker. Prometheus позволяет отслеживать состояние различных компонентов вашего приложения или инфраструктуры и предоставляет возможность анализировать данные, чтобы выявить потенциальные проблемы и улучшить производительность.

Программа позволяет собирать метрики с различных источников, таких как веб-серверы, базы данных, приложения и другие компоненты системы. С помощью Prometheus можно создавать гибкие и настраиваемые дашборды для мониторинга состояния системы в реальном времени. Он также предлагает широкий спектр инструментов для анализа данных и создания уведомлений о проблемах.

Prometheus имеет расширяемую архитектуру, что делает его удобным для интеграции с другими инструментами мониторинга и управления. Он широко используется в индустрии разработки программного обеспечения и системного администрирования для обеспечения надежной и эффективной работы инфраструктуры.

Отображением метрик Prometheus занимается Grafana. Grafana — это платформа для мониторинга и анализа данных, которая обеспечивает возможность создания красивых и информативных дашбордов и графиков. Она позволяет интегрировать данные из различных источников, таких как базы данных, метрические системы и сервисы мониторинга, и визуализировать их в удобной форме.

Grafana обладает широким набором функций, включая возможность настройки и настройки дашбордов, создание алертов, фильтрацию данных, использование шаблонов и многое другое. Она также поддерживает множество плагинов и интеграций, что делает её гибким инструментом для мониторинга различных систем и приложений.

В общем, Grafana позволяет аналитикам и разработчикам быстро и удобно визуализировать данные и получать ценную информацию для принятия решений и улучшения производительности системы.

Логи работы веб-приложения пишутся в файл при помощи инструмента интеграции. Spring Boot Starter Integration представляет собой стартовый набор для интеграции различных интеграционных технологий и платформ в приложения, разрабатываемые с использованием Spring Boot [20]. Этот стартер облегчает интеграцию с различными внешними системами и сервисами, такими как базы данных, веб-сервисы, кэши, сообщения и многое другое.

Spring Boot Starter Integration позволяет разработчикам легко настроить и использовать интеграционные компоненты в своих приложениях, упрощая процесс интеграции и уменьшая объем шаблонного кода. Включенные в стартер зависимости обеспечивают доступ к множеству интеграционных технологий, что позволяет быстро и эффективно создавать полнофункциональные приложения, взаимодействующие с различными системами.

Spring Boot Starter Integration активно поддерживается и обновляется сообществом разработчиков, что гарантирует совместимость с последними версиями Spring Boot и обновлениями интеграционных платформ. Благодаря этому стартеру, интеграция внешних систем становится проще и быстрее, а разработка приложений на основе Spring Boot - более эффективной и удобной.

Spring Integration File — это модуль Spring Integration, который обеспечивает возможности для работы с файлами [20]. Он предоставляет компоненты для чтения, записи и манипулирования файлами в приложениях Spring.

С помощью Spring Integration File можно легко настраивать интеграцию с файловой системой, обрабатывать файлы, переносить данные между различными источниками и назначениями, а также осуществлять мониторинг файлов и выполнение различных действий в зависимости от изменений в файлах.

Этот модуль предоставляет различные компоненты, такие как файловый источник (File Source), файловый приемник (File Sink), маршрутизаторы и фильтры для управления потоком данных, а также возможность настройки обработчиков событий для файлов.

Spring Integration File облегчает и упрощает работу с файлами в приложениях Spring, предоставляя гибкую конфигурацию и возможность интеграции с другими компонентами Spring Integration.

Junit Mockito — это инструмент для тестирования, который позволяет создавать и использовать моки (заглушки) объектов в Java . Mockito позволяет создавать объекты, которые ведут себя аналогично настоящим объектам, но не требуют наличия конкретной реализации.

С помощью Junit Mockito можно эффективно тестировать код, который зависит от внешних ресурсов или взаимодействует с другими компонентами приложения. Mockito позволяет заменить эти зависимости моками и создать управляемую среду для тестирования.

Junit Mockito облегчает написание юнит-тестов и улучшает их структуру, делая код более поддерживаемым и расширяемым. Он также помогает уменьшить время выполнения тестов и упрощает процесс отладки.

Использование Junit Mockito в комбинации с фреймворком JUnit позволяет создавать надежные и эффективные тесты для Java-приложений.

## 2.3 Заключение по разделу «Исследование предметной области»

В данном разделе было произведено описание предметной области: описание объекта автоматизации, объектами автоматизации которого выступали клиент веб-приложения и разработчик интегрируемых сервисов; описание рассматриваемого бизнес-процесса ‑ описание процессов, протекающих при использовании веб-приложения.

Были выбраны средства разработки веб-платформы ‑ проведено описание инструментов разработки серверной части веб-платформы.

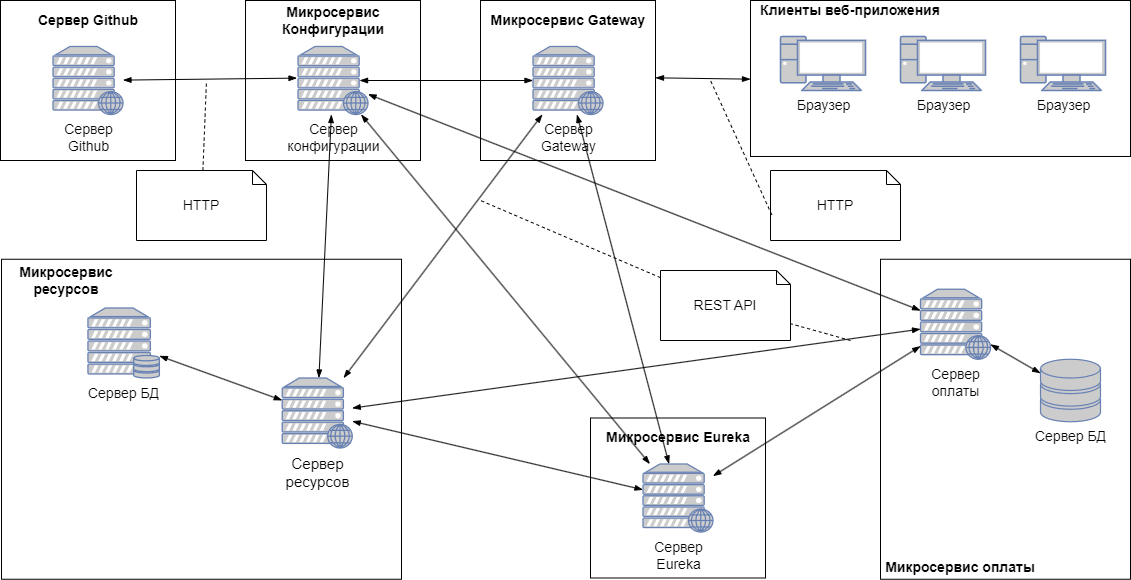
# Глава 3 Проектирование и реализация веб-приложения

## 3.1 Проектирование баз данных

Разрабатываемое веб-приложение является приложением для осуществления транзакций пользователями. В связи с этим требуются сущности для хранения данных о пользователях и состоянии счетов в базе данных, с которой работает ресурсный микросервис. Для сервиса транзакций требуется база данных с сущностями пользователя и его банковского аккаунта.

## 3.2 Инфраструктура информационной системы

Микросервисы ресурсов и оплаты, Eureka, конфигурации и Gateway, а также клиентские приложения составляют общую архитектуру веб-приложения.

  
Рисунок 12 ‑ Архитектура веб-приложения

Составные элементы архитектуры веб-приложения и связи между ними изображены на рисунке 12.

В нее входят:

* Клиенты веб-приложения:
* Клиенты направляют запрос на микросервис Gateway для получения информации о продуктах или оформления заказа.
* Микросервис Gateway:
* Сервер Gateway. Данный сервер является связующей частью между браузерным клиентом и микросервисом ресурсов. Сервер так же служит для взаимодействия с другими микросервисами, которые могут быть интегрированы в дальнейшем.
* Микросервис конфигурации:
* Сервер конфигурации. Данный сервер является хранилищем конфигурации для микросервисов: ресурсов, оплаты, Eureka, Gateway.
* Сервер Githib:
* Хранит в себе конфигурационные файлы для микросервисов, которые передает по запросу микросервису конфигурации.
* Микросервис Eureka:
* Регистрирует микросервисы: конфигурации, ресурсов, оплаты, Gateway.
* Микросервис ресурсов:
* Сервер ресурсов. Сервер содержит внутреннюю логику обработки данных продуктов, работы с базой данных, отправки и принятия данных с микросервиса оплаты.
* Сервер базы данных. База данных хранит информацию о заказах и продуктах.
* Микросервис оплаты:
* Сервер оплаты. Сервер содержит внутреннюю логику обработки оплаты заказа, работы с базой данных, отправки и принятия данных с микросервиса ресурсов.
* Сервер базы данных. База данных хранит информацию о пользователях и банковских аккаунтов.

## 3.3 **Реализация веб-приложения**

### 3.3.1 Конфигурирование и выстраивание архитектуры

Веб-приложение будет запущенно на localhost. Для начала требуется определить и реализовать микросервис конфигурации. Чтобы его реализовать, достаточно подключить зависимости: spring-boot-starter; spring-cloud-config-server.

Рисунок 15 – Конфигурационный файл Eureka микросервиса

Сконфигурируем микросервис ресурсов. Чтобы его реализовать, достаточно подключить зависимости: spring-boot-starter-web; Lombok; spring-boot-starter-test; spring-cloud-config; spring-boot-configuration-processor; spring-cloud-starter-openfeign; spring-boot-starter-data-jpa; mysql-connector-j; spring-boot-starter-aop; spring-boot-starter-integration; spring-integration-file. Над главным классом приложения навешивается аннотации

Этот файл является конфигурационным файлом для серверного приложения, написанного на Java с использованием фреймворка Spring. Давайте разберем каждую часть конфигурации: server.port = 8080: устанавливает порт, на котором будет запущен сервер. spring: application: name: resource-server: устанавливает имя приложения. Datasource.url устанавливает URL для подключения к базе данных MySQL. username устанавливает имя пользователя для подключения к базе данных.

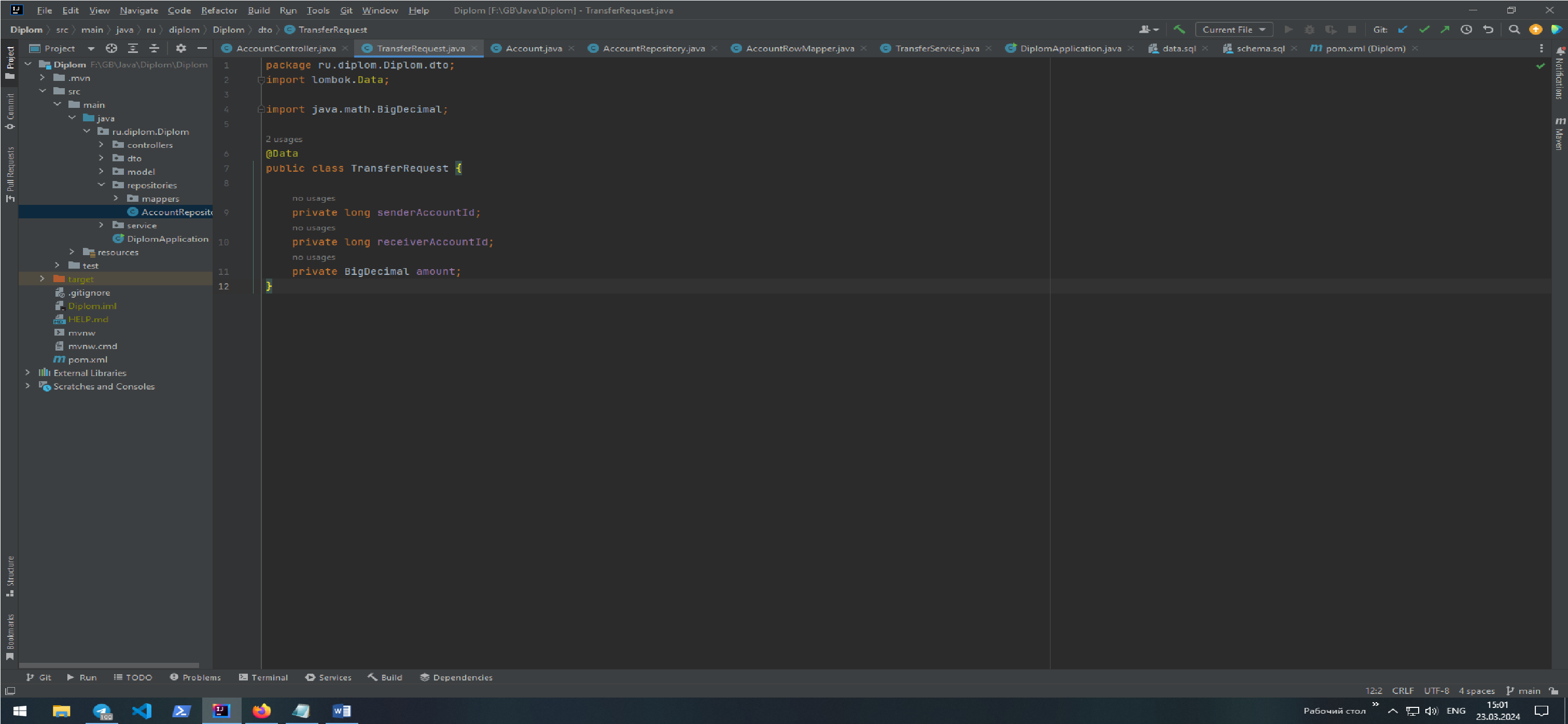
Сконфигурируем микросервис транзакции. Чтобы его реализовать, достаточно подключить зависимости: spring-boot-starter-web; spring-boot-starter-data-jdbc; Lombok; spring-boot-starter-test; h2database.

### 3.3.2 Реализация микросервиса ресурсов

Разработка приложения начинается с создания классов моделей, согласно описанной ранее диаграмме классов. Классы моделей вынесены в приложение А. На классы навешиваются: аннотация @Data генерирует код для геттеров, сеттеров, методов toString(), equals() и hashCode(); аннотация @NoArgsConstructor создает конструктор без параметров; аннотация @Entity указывает, что класс является сущностью JPA; аннотация @Table(name = "orders") указывает на таблицу в базе данных, в которой будет храниться информация о заказах.

В DTO классе TransferRequest содержатся три атрибута: senderAccountId (аккаунт отправителя), receiverAccountId (аккаунт получателя) и amount(сумма транзакции) как представлено на рисунке 20. Все атрибуты являются приватными и доступны только через соответствующие методы getter и setter.

Рисунок 20 – Репозиторий заказа

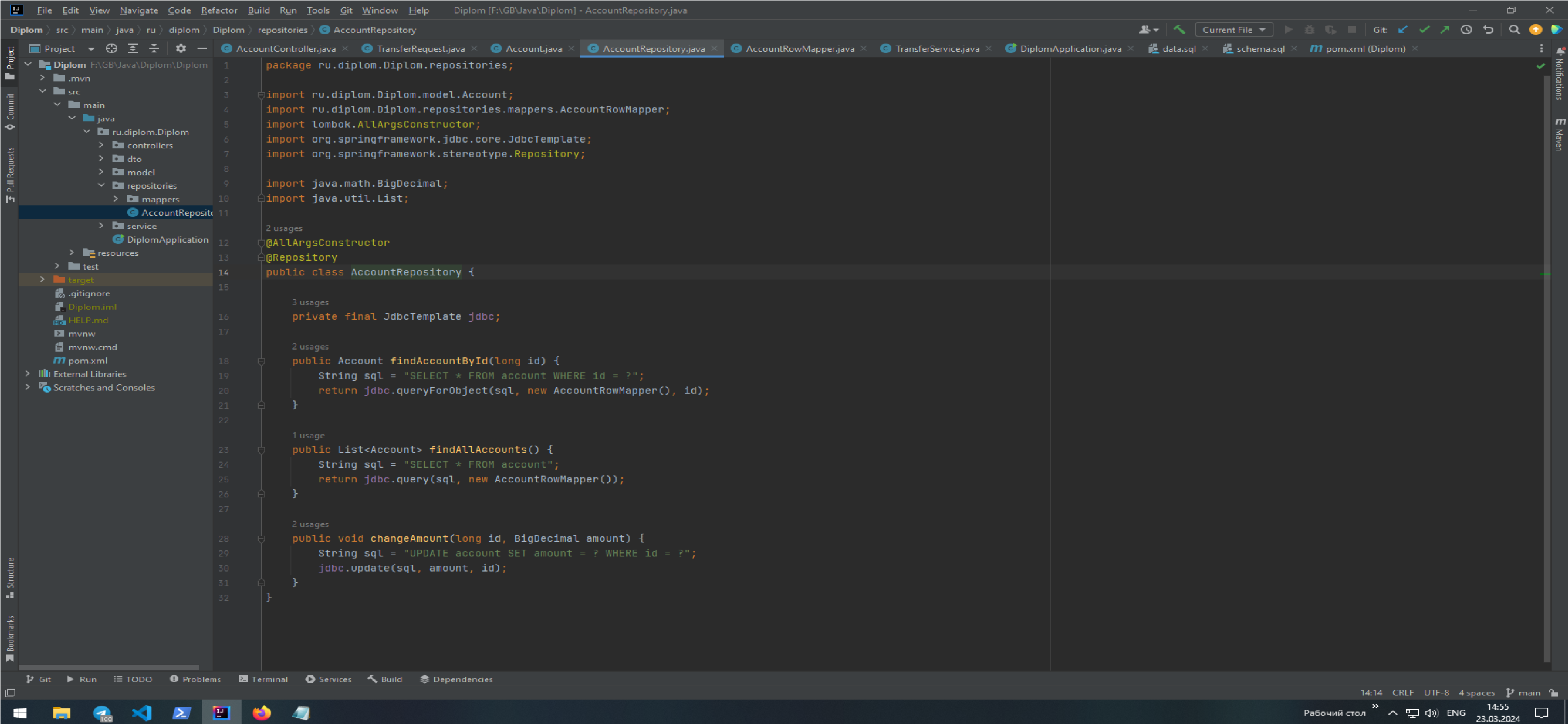


DTO (Data Transfer Object) - это объект, который используется для передачи данных между слоями приложения. Он содержит только данные и не содержит логики. DTO позволяет передавать данные без необходимости передачи всего объекта с его методами и свойствами.

Таким образом, класс PaymentInvoice представляет собой простой объект для передачи данных о платежном листе между различными компонентами приложения. Он содержит минимальный набор данных, необходимых для выполнения определенных операций, и предназначен для удобного и безопасного обмена этими данными.

Далее, создаются репозитории для работы с моделями. Например, представленный на рисунке 20 репозиторий транзакции.

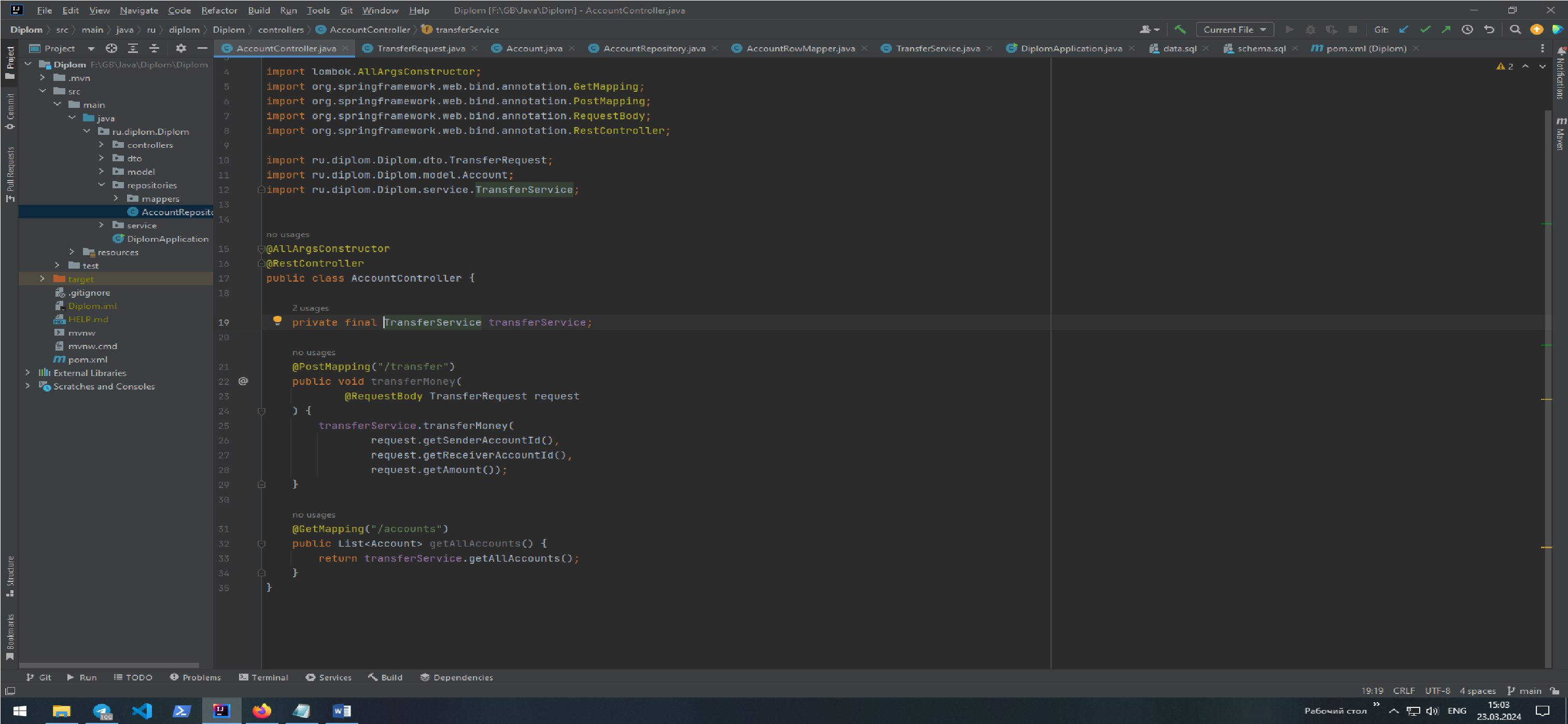
Рисунок 20 – Репозиторий транзакции



@Repository — это аннотация Spring Framework, которая указывает на то, что данный интерфейс представляет репозиторий данных. Репозиторий данных является частью паттерна проектирования DAO (Data Access Object) и обеспечивает доступ к данным в базе данных. Интерфейс OrderRepository представляет собой репозиторий для сущности Order. Этот интерфейс наследуется от интерфейса JpaRepository, который предоставляет базовые методы для работы с данными в базе данных. В частности, JpaRepository позволяет выполнять CRUD операции (Create, Read, Update, Delete) с объектами типа Order. В данном случае, интерфейс OrderRepository работает с сущностью Order. Таким образом, данный интерфейс предоставляет абстракцию для работы с данными транзакцийи упрощает выполнение операций с ними в приложении, используя основные методы, предоставленные JpaRepository.

Для передачи данных о транзакции между клиентами определим класс AccountController (рисунок 21).

Рисунок 21 – класс AccountController



Следующим шагом реализации микросервиса, является определение его сервисов.

Класс TransferService является сервисным классом, который предоставляет различные методы для работы с данными в базе данных. Он использует AccountRepository для взаимодействия с базой данных.

Метод getAllAccounts () возвращает список всех аккаунтов из базы данных.

Приступим к реализации контроллеров.

класс AccountController является контроллером Spring, который отвечает за обработку запросов, связанных с транзакциями. Он помечен аннотацией @RestController, что означает, что результат каждого метода будет автоматически преобразован в формат JSON и возвращен клиенту.

В конструкторе класса используется аннотация @AllArgsConstructor, что генерирует конструктор, принимающий все аргументы.

Аннотация @GetMapping устанавливает базовый URL для всех методов в классе. В данном случае все эндпоинты начинаются с "/accounts".

### 3.3.3 Реализация микросервиса транзакции

Модели Account. Была добавлена модель DTO TransferRequest, описанная в микросервисе ресурсов.

Метод transferMoney отвечает за проведение транзакции. В этом методе сначала извлекается аккаунт пользователя по его имени из переданных данных. Далее проверяется, достаточно ли средств на аккаунте для проведения транзакции. Если баланс меньше суммы платежа, то выбрасывается исключение BalanceException. Если средств достаточно, то сумма транзакции вычитается из баланса аккаунта.

Метод помечен аннотацией @Transactional, что гарантирует, что операции будут выполнены в рамках одной транзакции базы данных. Это помогает обеспечить целостность данных при обработке платежей.

## 3.4 Заключение по разделу «Проектирование и реализация веб-приложения»

В данном разделе были описаны: роли пользователей веб-платформы, приведена инфраструктура системы, а также были приведены описания этапов реализации веб-приложения и реализованного функционала веб-приложения.

# Заключение

В ходе выполнения дипломной работы, была спроектирована и реализована серверная часть веб-приложения микросервисов транзакции и ресурсов, позволяющая легко интегрировать новых клиентов и новые микросервисы.

# Список источников

1. Oswaldo Sebastian Penaherrera-Pulla. Measuring Key Quality Indicators in Cloud Gaming: Framework and Assessment Over Wireless Networks / Carlos Baena, Sergio Fortes, Eduardo Baena, Raquel Barco // Sensors. – 2021. – Vol. 4, №11. – P.1387-1411.
2. Santosh Kumar. A review on client-server based applications and research opportunity // International Journal of Scientific Research. – 2019. – Vol. 10, № 7. – P.33857-33862.
3. Spring Cloud Config и Git. Хранение и распространение конфигураций приложений. Хабр. [Электронный ресурс]. Режим доступа: свободный. Ссылка на ресурс: https://habr.com/ru/articles/764402/ (Дата обращения: 16.03.2024).
4. Introduction to Spring Cloud Netflix – Eureka. Baeldung. [Электронный ресурс]. Режим доступа: свободный. Ссылка на ресурс: https://www.baeldung.com/spring-cloud-netflix-eureka (Дата обращения: 16.03.2024).
5. Spring Cloud Gateway. Spring docs. [Электронный ресурс]. Режим доступа: свободный. Ссылка на ресурс: https://docs.spring.io/spring-cloud-gateway/docs/current/reference/html/ (Дата обращения: 16.03.2024).
6. Нотация IDEF0. Business Studio. [Электронный ресурс]. Режим доступа: свободный. Ссылка на ресурс: https://www.businessstudio.ru/wiki/docs/current/doku.php/ru/csdesign/bpmodeling/idef0#:~:text=IDEF0%20%2D%20%D0%BD%D0%BE%D1%82%D0%B0%D1%86%D0%B8%D1%8F%20%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B3%D0%BE%20%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F%2C%20%D0%B8%D1%81%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D1%83%D0%B5%D0%BC%D0%B0%D1%8F,%D0%BC%D0%B0%D1%82%D0%B5%D1%80%D0%B8%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D1%85%20%D0%BE%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BE%D0%B2%2C%20%D1%81%D0%B2%D1%8F%D0%B7%D1%8B%D0%B2%D0%B0%D1%8E%D1%89%D0%B8%D1%85%20%D1%8D%D1%82%D0%B8%20%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%B8. (Дата обращения: 16.03.2024).
7. JWT-аутентификация при помощи Spring Boot 3 и Spring Security 6. Хабр. [Электронный ресурс]. Режим доступа: свободный. Ссылка на ресурс: https://habr.com/ru/articles/784508/ (Дата обращения: 16.03.2024).
8. Что такое Java? AWS. [Электронный ресурс]. Режим доступа: свободный. Ссылка на ресурс: https://aws.amazon.com/ru/what-is/java/ (Дата обращения: 16.03.2024).
9. Spring для ленивых. Основы, базовые концепции и примеры с кодом. Часть 1. Javarush. [Электронный ресурс]. Режим доступа: свободный. Ссылка на ресурс: https://javarush.com/groups/posts/spring-framework-java-1 (Дата обращения: 16.03.2024).
10. MySQL Documentation. MySQL. [Электронный ресурс]. Режим доступа: свободный. Ссылка на ресурс: https://dev.mysql.com/doc/ (Дата обращения: 16.03.2024).
11. Building an Application with Spring Boot. Spring. [Электронный ресурс]. Режим доступа: свободный. Ссылка на ресурс: https://spring.io/guides/gs/spring-boot (Дата обращения: 16.03.2024).
12. JPA и Spring Data JPA. Javarush. [Электронный ресурс]. Режим доступа: свободный. Ссылка на ресурс: https://javarush.com/quests/lectures/questspringboot.level03.lecture02 (Дата обращения: 16.03.2024).
13. Lombok. Полное руководство. Хабр. [Электронный ресурс]. Режим доступа: свободный. Ссылка на ресурс: https://habr.com/ru/companies/piter/articles/676394/ (Дата обращения: 16.03.2024).
14. Spring Cloud OpenFeign. Spring. [Электронный ресурс]. Режим доступа: свободный. Ссылка на ресурс: https://spring.io/projects/spring-cloud-openfeign (Дата обращения: 16.03.2024).
15. Introduction to Spring AOP. Baeldung. [Электронный ресурс]. Режим доступа: свободный. Ссылка на ресурс: https://www.baeldung.com/spring-aop (Дата обращения: 16.03.2024).
16. Building a RESTful Web Service with Spring Boot Actuator. Spring. [Электронный ресурс]. Режим доступа: свободный. Ссылка на ресурс: https://spring.io/guides/gs/actuator-service (Дата обращения: 16.03.2024).
17. Micrometer Documentation. Micrometer. [Электронный ресурс]. Режим доступа: свободный. Ссылка на ресурс: https://docs.micrometer.io/micrometer/reference/ (Дата обращения: 16.03.2024).
18. GETTING STARTED. Prometheus. [Электронный ресурс]. Режим доступа: свободный. Ссылка на ресурс: https://prometheus.io/docs/prometheus/latest/getting\_started/ (Дата обращения: 16.03.2024).
19. Сбор метрик Spring Boot приложения c помощью Prometheus и Grafana. Хабр. [Электронный ресурс]. Режим доступа: свободный. Ссылка на ресурс: https://habr.com/ru/articles/548700/ (Дата обращения: 16.03.2024).
20. Spring Boot Starter Integration. Spring. [Электронный ресурс]. Режим доступа: свободный. Ссылка на ресурс: https://spring.io/projects/spring-integration (Дата обращения: 16.03.2024).
21. Разработка через тестирование. Совместное использование JUnit 5 и Mockito. Хабр. [Электронный ресурс]. Режим доступа: свободный. Ссылка на ресурс: https://habr.com/ru/companies/sberbank/articles/773142/ (Дата обращения: 16.03.2024).