Geekbrains

Создание сервиса для парсинга сайтов и скачивания контента с использованием Spring и Vaadin

Программа:
Веб-разработка на Java
Студент:
Попов Е.А.

Ижевск
2024

Содержание

Введение

Дипломный проект, представленный в данном исследовании, представляет собой решение, разработанное специально для бизнеса, с акцентом на эффективное наполнение интернет-магазинов контентом. Проект реализован с использованием технологий Spring и Vaadin, обеспечивая высокую производительность и удобство использования.

Основное предназначение проекта заключается в парсинге веб-сайтов и загрузке необходимых файлов на компьютер пользователя, что позволяет оптимизировать процесс наполнения интернет-ресурсов контентом.

Архитектура проекта представлена серверной частью, на которой размещен веб-сайт, и клиентским приложением, ответственным за скачивание файлов. Кроме того, в проекте задействован контейнер с базой данных, где хранятся данные пользователей и информация о проектах. Взаимодействие между клиентом и сервером осуществляется при помощи REST-запросов, обеспечивая эффективную передачу данных и управление рабочим процессом.

В проекте существуют два типа пользователей: обычные пользователи и администраторы. Администраторы имеют доступ к функциям управления пользователями и могут просматривать список их проектов (без доступа к содержанию). Администраторы также имеют возможность блокировать пользователей и полностью удалять их аккаунты.

Обычные пользователи могут создавать, удалять и редактировать свои проекты. В проекте можно добавлять артикулы, которые будут использоваться для поиска контента на сайте. Также можно выполнять поиск сразу на нескольких локализованных сайтах для разных стран или регионов. Эта опция добавлена для того, чтобы наиболее полным образом скачивать необходимый контент, тк товары продающиеся в одном регионе и доступные на одном локализованном сайте могут быть недоступны для другого региона.

После настройки проекта в личном кабинете клиентское приложение выполняет скачивание необходимых файлов и генерирует подробный отчет о выполненной работе, обеспечивая прозрачность и контроль над процессом.

Таким образом, данный дипломный проект является важным инструментом для бизнеса, обеспечивающим оптимизацию процесса наполнения интернет-магазинов контентом и повышение эффективности бизнес-процессов.

2. Теоретическая часть

2.1. Что такое парсер и для чего он нужен?

Парсинг — это автоматический процесс сбора и систематизации данных в интернете.

Парсеры способны обрабатывать большие объемы данных быстро, эффективно и, что самое важное - автономно (без участия оператора), что делает их важным инструментом для решения бизнес задач.

Использование парсера позволяет автоматизировать процесс скачивания контента с веб-страниц, что существенно экономит время и усилия операторов. Благодаря парсеру возможно быстрое и точное извлечение необходимой информации из разнообразных источников, обеспечивая эффективное наполнение интернет-магазина контентом и повышение производительности бизнес-процессов.

Парсинг имеет множество применений в различных областях, включая анализ конкурентов, SEO-продвижение, запуск рекламы, наполнение сайтов, анализ контента и сквозную аналитику. Ниже представлены основные функции, которые может выполнять парсер.

1. Анализ конкурентов. Парсинг позволяет собрать информацию о товарах и ценах, предлагаемых другими компаниями. Это полезно для понимания конкурентной среды и формирования стратегий развития бизнеса.
2. SEO-продвижение. Парсинг может помочь собрать семантическое ядро, найти ошибки на веб-сайте и проанализировать поисковую выдачу. Это способствует улучшению позиций в поисковых системах и повышению видимости бренда.

3. Запуск рекламы. Парсинг позволяет собрать базу целевой аудитории или найти потенциальные рекламные площадки. Это позволяет эффективно направлять рекламные кампании и достигать целевой аудитории.
4. Наполнение сайтов. Парсинг может помочь автоматизировать процесс сбора информации из разных источников и наполнения сайта информацией. Это может быть полезно для быстрого наполнения сайта большим объемом контента.

В случае с данным дипломным проектом - парсер – это программное обеспечение, специально разработанное для анализа и обработки данных. В контексте данного проекта парсер осуществляет сканирование HTML кода веб-страниц с целью обнаружения ссылок на контент.

Основной задачей данного дипломного проекта является создание парсера, который будет заниматься извлечением необходимой информации из веб-страниц – это ссылки на файлы и изображения, которые впоследствии будут скачаны и использованы для наполнения интернет-магазина контентом. Таким образом, парсер представляет собой важный инструмент в области веб-разработки, обеспечивающий автоматизацию процесса анализа данных на веб-страницах и извлечение необходимой информации для последующего использования.

2.2. Основы HTML

HTML был придуман в 1980-х годах для создания веб-страниц, которые могли быть легко просмотренными и использованными людьми без специальных навыков программирования. В то время, когда Интернет был в его начальной стадии развития, HTML стал стандартным языком для создания веб-страниц, так как он был простым, легким в использовании и позволял веб-дизайнерам создавать веб-страницы, которые могли быть понятными людям со всего мира.

HTML (Hypertext Markup Language) - это язык разметки, используемый для структурирования и отображения веб-страниц и их контента. HTML состоит из различных элементов, которые используются для вкладывания и оборачивания различных частей контента, чтобы задать ему определенное отображение или поведение. Элементы могут быть сложными или простыми в зависимости от требуемой функциональности веб-страницы. Например, элемент <a> используется для создания ссылок, элемент <img> для отображения изображений, а элемент <p> для оборачивания абзацев текста. HTML также позволяет создавать более сложные структуры, такие как таблицы, списки и другие элементы, которые помогают организовать и отобразить контент более эффективно. В целом, HTML предоставляет гибкую систему для создания веб-страниц, которая может быть адаптирована под различные устройства и браузеры.

2.3. Механизмы парсинга HTML

Парсинг HTML - это процесс анализа и извлечения информации из HTML документа. Существует несколько механизмов парсинга HTML, каждый с своими плюсами и минусами.

DOM-базированный парсинг

Описание: DOM (Document Object Model) - это кросс-платформенный и языконезависимый интерфейс, который представляет HTML-документы в виде дерева объектов. Парсеры, использующие DOM, читают HTML и создают из него "живое" дерево DOM, с которым затем можно взаимодействовать программно.

Плюсы:

1. Интуитивно понятная структура, отражающая вложенность элементов HTML.
2. Возможность изменения структуры документа на лету.
3. Широкая поддержка во многих языках программирования.

Минусы:

1. Высокое потребление памяти, поскольку весь HTML-документ загружается в память.
2. Относительно медленная скорость обработки из-за необходимости создания полной модели документа.

SAX-базированный парсинг

Описание: SAX (Simple API for XML) - это событийно-ориентированный парсер, который читает HTML-документ последовательно и сообщает о найденных элементах через возникающие события.

Плюсы:

1. Низкое потребление памяти, так как документ читается последовательно.
2. Быстрая скорость обработки, подходит для анализа больших документов.

Минусы:

1. Отсутствие возможности изменять документ в процессе чтения.
2. Меньшая интуитивность по сравнению с DOM, так как требует обработки событий.

Парсинг с использованием регулярных выражений

Описание: Регулярные выражения могут использоваться для поиска и извлечения информации из HTML-кода по заданным шаблонам.

Плюсы:

1. Гибкость в поиске и извлечении специфических данных.
2. Высокая скорость работы на небольших фрагментах текста.

Минусы:

1. Сложность в написании и поддержке (регулярные выражения могут быть запутанными).
2. Не подходит для сложных HTML-документов с нестандартной вложенностью элементов.

CSS селекторы

Описание: Современные библиотеки, такие как BeautifulSoup или jQuery, используют CSS селекторы для выборки элементов из HTML-документа.

Плюсы:

1. Выразительный и понятный синтаксис.
2. Возможность точечного выбора элементов без загрузки всего документа в память.

Минусы:

1. Зависит от корректности HTML-кода.
2. Может быть неэффективен при обработке очень больших документов.

Выбор оптимального механизма для парсинга HTML обычно зависит от задачи, объема данных и требуемой скорости обработки. В данном проекте использован парсинг с регулярными выражениями, который обладает следующими особенностями в контексте данного дипломного проекта.

Так как дизайн каждой страницы сайта, откуда будет парситься контент, одинаков и шаблонизирован - а это значит, что каждая HTML страница построена по единому шаблону, предсказуема и структурирована, то поиск контента для скачивания лучше выполнять с использованием регулярных выражений.

Однако, у такого подхода так же есть ограничения, для работы с более сложными или изменчивыми HTML-документами, этот метод может оказаться ненадежным и приводить к ошибкам. В таких ситуациях рекомендуется использовать специализированные HTML-парсеры и библиотеки, которые предназначены для работы с HTML и могут обрабатывать его сложную и динамичную природу.

2.4. Общая архитектура приложения

Архитектура программного обеспечения – это его скелет, определяющий стабильность, эффективность и долговечность проекта. Рассмотрим общую архитектуру проекта, которая была выбрана в соответствии из потребностей современного рынка.

Выбор серверной платформы – это решение, которое сопряжено с ответственностью за будущую поддержку и развитие приложения. Spring Framework в этом контексте представляет собой универсальный инструмент, способный обеспечить надежную основу для любой бизнес-логики. Почему Spring? Он является де-факто стандартом в мире Java-разработки, предлагая глубокую интеграцию с миром Java EE(Java Enterprise Edition), превосходную модель безопасности и транзакционного управления, а также готовые решения для работы с базами данных и интеграции с внешними сервисами.

Плюсы Spring:

1. Широкая функциональность: Spring предоставляет широкий спектр модулей для различных задач, включая Spring MVC для веб-приложений, Spring Data для работы с базами данных, Spring Security для безопасности и многие другие.
2. Инверсия управления (IoC) и внедрение зависимостей (DI): Эти ключевые концепции упрощают управление жизненным циклом объектов и конфигурацию приложения, что облегчает тестирование и поддержку кода.
3. Аспектно-ориентированное программирование (AOP): Позволяет разработчикам определять общие задачи, такие как логирование или транзакционное управление, отдельно от основного бизнес-логики.

4. Мощное транзакционное управление: Упрощает работу с транзакциями, обеспечивая декларативную поддержку транзакций через аннотации.
5. Интеграция с множеством технологий: Spring легко интегрируется с ведущими технологиями для работы с базами данных, ORM-фреймворками, серверами приложений и многими другими.

Минусы Spring:

1. Высокий порог входа: Обширный набор возможностей может быть сложен для новичков.
2. Сложность конфигурации: Несмотря на упрощение с помощью Java-конфигурации и аннотаций, настройка Spring-приложения все еще может быть сложной.
3. Производительность: В некоторых случаях использование Spring может привести к небольшому снижению производительности из-за абстракций и автоматической обработки.

Так же необходимо выбрать фреймворк для фронтенд части проекта. В данном проекте я решил использовать для этого Vaadin.

Vaadin – это не просто фреймворк, это мост между миром серверных вычислений и веб-технологий. Он позволяет разработчикам, привычным к стандартам Java, в полной мере проявлять свои навыки в создании богатых интерактивных веб-интерфейсов без необходимости погружения в глубины JavaScript и HTML. Vaadin автоматически управляет коммуникациями между браузером и сервером, позволяя разработчикам сосредоточиться на дизайне пользовательского интерфейса и логике, не заботясь о нюансах низкоуровневой веб-разработки.

В Vaadin каждый компонент UI существует как на стороне сервера, так и на клиенте, обеспечивая плавное и эффективное взаимодействие. Это уникальность Vaadin: он делает современные веб-приложения доступными для традиционных Java-разработчиков, устраняя барьеры между серверным кодом и веб-фронтендом.

Однако, Vaadin не лишен недостатков. Из-за серверно-центричной модели, интерфейсы могут страдать от задержек при слабом сетевом соединении, а сервер может испытывать повышенную нагрузку из-за обработки всех действий пользовательского интерфейса. Рассмотрим основные плюсы и минусы данного фреймворка.

Плюсы Vaadin:

1. Простота разработки: Разработчики могут создавать интерфейсы, используя только Java, без необходимости знания HTML, CSS или JavaScript.
2. Серверно-центричная архитектура: Логика приложения находится на сервере, что облегчает разработку и обеспечивает лучший контроль над приложением.
3. Компонентный подход: Богатый набор готовых к использованию компонентов UI ускоряет процесс разработки.
4. Отзывчивый дизайн: Vaadin поддерживает создание отзывчивых приложений, которые хорошо работают на различных устройствах и размерах экрана.
5. Расширяемость и настраиваемость: Можно легко создавать пользовательские компоненты и темы.

Минусы Vaadin:

1. Привязка к серверу: Такой подход может привести к задержкам в интерактивности приложения при медленных интернет-соединениях.

2. Масштабируемость: Поскольку бизнес-логика обрабатывается на сервере, это может создавать дополнительную нагрузку на сервер, особенно при большом количестве пользователей. 3. Ограниченная поддержка SEO: Поскольку Vaadin-приложения обычно рендерятся на стороне сервера, их содержимое не всегда легко индексируется поисковыми системами, что может затруднить SEO-оптимизацию. 4. Зависимость от фреймворка: Код приложения тесно связан с фреймворком Vaadin, что может затруднить переход на другие технологии в будущем. 5. Обновления и совместимость: Переход на новую версию Vaadin может потребовать значительных изменений в коде из-за изменений в API и поведении компонентов.

Vaadin подходит для разработки бизнес-приложений, где важна быстрая разработка и не требуется высокая масштабируемость. Он позволяет создавать полнофункциональные веб-приложения, предлагая привлекательный и функциональный пользовательский интерфейс без необходимости вникать в детали клиентской стороны. Однако для проектов с высокими требованиями к производительности, масштабируемости или SEO может потребоваться рассмотрение альтернативных подходов.

Также огромную роль играет выбор базы данных, тк она будет отвечать за хранение и обработку информации. Поэтому для данного проекта я выбрал PostgreSQL .

PostgreSQL – это продвинутая объектно-реляционная система управления базами данных (СУБД), которая с годами завоевала репутацию надежного и устойчивого решения для самых различных задач хранения и обработки данных. Выбор PostgreSQL в качестве основной СУБД для нашего проекта был продиктован не только ее мощными возможностями, но и требованиями к безопасности, расширяемости и соответствию стандартам SQL.

Плюсы PostgreSQL:

1. Надежность и безопасность: PostgreSQL имеет многолетнюю историю использования в крупных и требовательных к безопасности проектах.
2. Соответствие стандартам: Она тщательно следует стандартам SQL и часто внедряет новшества раньше других СУБД.
3. Расширяемость: PostgreSQL позволяет разработчикам создавать свои собственные типы данных, функции и даже языки.
4. Мощные инструменты для работы с данными: Поддержка транзакций, многоверсионности, сложных запросов, полнотекстового поиска и геопространственных данных.

Минусы PostgreSQL:

1. Сложность управления и настройки: Несмотря на свою мощь, PostgreSQL может быть сложной в настройке и оптимизации для новичков в области управления базами данных.
2. Производительность под высокими нагрузками: Хотя PostgreSQL отлично справляется с большим объемом данных, под определенными высокими нагрузками ее производительность может быть ниже, чем у некоторых специализированных СУБД.

Интеграция PostgreSQL в проект

В нашем проекте PostgreSQL играет роль центрального хранилища данных, где сосредоточены все настройки проектов и данные пользователей. Интеграция с Spring позволяет нам эффективно работать с базой данных, используя Spring Data JPA для удобного маппинга объектов Java в таблицы базы данных и обратно. Это означает, что разработчики могут работать с базой данных на более высоком уровне абстракции, не заботясь о SQL-запросах и тонкостях обращения к данным.

Spring Data JPA в сочетании с Hibernate обеспечивает мощный слой абстракции над базой данных, позволяя проектировать и изменять схему базы данных в соответствии с моделью домена приложения. Транзакционность и консистентность данных гарантируются благодаря тщательно продуманным механизмам управления транзакциями, предоставляемыми Spring Framework.

Таким образом, интеграция PostgreSQL в проект обеспечивает надежную и масштабируемую платформу для всех операций с данными, поддерживая высокий уровень производительности и гибкости, необходимый для современных приложений.

Клиентское приложение, представляющее собой HTML-страницу с интерфейсом пользователя, который он открывает в браузере, реализуется с применением самых передовых технологий фронтенд-разработки на Vaadin. Оно позволяет пользователю взаимодействовать с сайтом через браузер, обеспечивая управление проектами и их настройку. Серверная сторона, сформированная на базе надежного и масштабируемого Spring, обращается к базе данных PostgreSQL, где хранится вся необходимая информация о пользователях и их проектах. Сервер не только управляет данными, но и генерирует динамические веб-страницы, которые пользователь открывает в своем браузере. Как только пользователь инициирует запуск проекта, сервер отправляет REST запрос клиентскому приложению. Этот запрос запускает парсинг HTML-страниц и скачивание контента.

В эпоху цифровой трансформации и веб-инженерии, симбиоз клиент-серверных архитектур и баз данных занимает центральное место в создании современных веб-приложений. Изящество таких систем обусловлено взаимодействием трех ключевых компонентов: клиентского приложения, сервера и базы данных. Такой подход к архитектуре – выбор между мощью Spring и простотой Vaadin – позволяет выполнить весь проект на Java и сделать качественный продукт без использования фреймворков для фронтенд разработки.

2.5. Основные классы приложения и их взаимодействие

В мире современной разработки веб-приложений, где пользовательский интерфейс и взаимодействие с пользователем занимают центральное место, Spring и Vaadin выступают в роли мощного дуэта. Они формируют прочную структуру, позволяющую разработчикам создавать надежные и масштабируемые приложения, следуя принципам MVC (Model-View-Controller).

Классы модели

В основе нашего веб-приложения, спроектированного по принципу MVC, лежат классы модели, которые являются стержнем бизнес-логики и управления данными.

Класс User становится центральным элементом в управлении данными пользователя. Он несет ответственность за хранение всей информации о пользователях, такой как их имена, контактная информация и учетные данные безопасности.

Класс Project является хранилищем проектов. Он организует и обеспечивает доступ к информации о различных проектах, находящихся в работе или завершенных.

Класс PostData занимается трансформацией данных для REST запросов в стандартизированный формат, обеспечивая их готовность к передаче между сервером и клиентом.

Репозитории и сервисы

Каждый из этих классов модели поддерживается соответствующими репозиториями и сервисами. Репозитории осуществляют непосредственное взаимодействие с базой данных, используя Spring Data JPA для упрощения CRUD операций. Сервисы служат прослойкой между репозиториями и остальной частью приложения, предоставляя бизнес-логику и транзакционное управление.

Генерация фронтенда

Перейдем к классам, отвечающим за генерацию фронтенда. Vaadin играет ключевую роль, создавая надежный и интерактивный пользовательский интерфейс. Классы, находящиеся в папке views, являются определением визуальной стороны приложения. Они определяют то, как элементы будут отображаться пользователю, и управляют взаимодействием.

Запуск приложения

И, наконец, класс Application — основной класс фреймворка Spring. Он несет ответственность за запуск нашего приложения, связывая все вышеописанные компоненты в одно целое, готовое к работе.

Каждый из этих элементов сыграл свою роль в создании веб-приложения, которое не только эффективно и безопасно обрабатывает данные, но и предоставляет пользователям удобный и приятный интерфейс.

2.6. Сущность User

В Spring Framework, когда говорится о проектировании сущности, то обычно имеется в виду создание класса, который будет описывать определенный объект, действие или персону. В нашем случае представлять пользователя и его данные в базе данных будет класс User. Для упрощения работы и доступа к базе данных используется JPA (Java Persistence API).

Код класса User представляет из себя следующее:

```
@Entity
@Table(name = "application_user")
@DynamicUpdate
public class User extends AbstractEntity {
    private String username;
    private String name;
    @JsonIgnore
    private String hashedPassword;
    private boolean banned;
    @Email
    private String email;
    @Enumerated(EnumType.STRING)
    @ElementCollection(fetch = FetchType.EAGER)
    private Set<Role> roles;
}
```

Рассмотрим аннотации и их назначение:

@Entity

Аннотация @Entity сообщает JPA, что данный класс является сущностью, то есть его экземпляры должны быть управляемыми и сохраняемыми в базе данных.

@Table

Аннотация @Table используется для указания таблицы в базе данных, к которой будет привязана сущность. В данном случае, name = "application_user" означает, что данные экземпляры User будут храниться в таблице application_user.

@DynamicUpdate

Аннотация @DynamicUpdate (свойство Hibernate), позволяет обновлять только те поля в базе данных, которые действительно изменились, что может повысить производительность.

@Enumerated and @ElementCollection

Эти аннотации используются для хранения коллекции перечислений Role в виде строки (поскольку используется EnumType.STRING). @Enumerated(EnumType.STRING) указывает JPA хранить перечисление в виде строк. @ElementCollection указывает, что мы имеем коллекцию элементов (в данном случае ролей), и fetch = FetchType.EAGER означает, что роли будут загружены сразу при загрузке пользователя из базы данных.

Поля класса

username - это поле представляет уникальное имя пользователя или так называемый login.

name - полное имя пользователя (имя и фамилия для отображения в профиле).

hashedPassword - помечено @JsonIgnore, чтобы не включать хешированный пароль в JSON-ответы. Поле хранит хешированный пароль пользователя.

banned - флаг, указывающий, заблокирован ли пользователь.

email - аннотация @Email валидирует, что строка соответствует формату электронной почты.

roles - в данном поле хранятся роли пользователей сайта: ADMIN или USER.

2.7. Сущность Project

Теперь рассмотрим сущность Project, которая описывает проекты пользователей.

```
@Entity
@Table(name = "project")
@DynamicUpdate
public class Project {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Version
    private int version;
    @Column(nullable = false)
    private String title;
    @Column(nullable = false)
    private String description;
    @OneToMany(mappedBy = "project", cascade = CascadeType.ALL, orphanRemoval = true, fetch = FetchType.EAGER)
    private Set<Article> articles = new HashSet<>();
    @OneToMany(mappedBy = "project", cascade = CascadeType.ALL, orphanRemoval = true, fetch = FetchType.EAGER)
    private Set<Variants> variants = new HashSet<>();
    @ManyToOne
    @JoinColumn(name = "user_id", nullable = false)
    private User user;
    @Column
    private String jsonData;
}
```

Вот так выглядит класс Project, теперь подробнее рассмотрим новые аннотации, используемые для реализации его функционала.

@Id и @GeneratedValue

Аннотация @Id обозначает первичный ключ сущности, а @GeneratedValue указывает стратегию генерации идентификаторов. Здесь выбрана стратегия GenerationType.IDENTITY, что означает, что уникальный идентификатор будет автоматически генерироваться базой данных.

@Version

Аннотация @Version используется для оптимистичной блокировки. Поле version будет автоматически увеличиваться при каждом обновлении сущности, что помогает предотвратить одновременные конфликтующие обновления.

@Column

Аннотация @Column используется для указания характеристик столбца, связанного с полем сущности. nullable = false говорит о том, что поля title и description не могут быть пустыми.

@OneToMany

Эта аннотация указывает на однонаправленную связь один ко многим между Project и Article и Variants. Свойство mappedBy обозначает владельца отношения. В данном случае, Project является обратной стороной отношения, а Article и Variants содержат ссылку project. cascade = CascadeType.ALL означает, что все операции каскадируются от Project к Article и Variants, а orphanRemoval = true говорит о том, что любые сиротские статьи или варианты будут автоматически удалены. fetch = FetchType.EAGER означает, что связанные статьи и варианты будут загружены сразу же при загрузке проекта.

@ManyToOne и @JoinColumn

Аннотация @ManyToOne описывает многонаправленную связь многие к одному между Project и User. @JoinColumn определяет столбец, который будет использоваться для соединения с таблицей пользователя. В данном случае, user_id является внешним ключом, который ссылается на пользователя, владеющего проектом, и не может быть пустым (nullable = false).

Поля класса

id - это поле представляет уникальный идентификатор проекта в базе данных. Оно автоматически генерируется при создании нового экземпляра проекта. Тип Long выбран для обеспечения достаточного пространства для хранения больших значений идентификаторов.

version - поле version используется для контроля одновременности. Оно помогает управлять конкурентным доступом и поддерживать целостность данных при одновременных изменениях. Поле типа int автоматически увеличивается каждый раз, когда проект обновляется.

title - Атрибут title хранит заголовок или название проекта. Строковый тип String используется для хранения текстовых данных. Атрибут помечен как nullable = false, что обозначает обязательность этого поля при сохранении проекта в базе данных.

description - Поле description содержит описание проекта. Также является строковым типом и не может быть пустым, что обеспечивает наличие описания для каждого проекта.

articles - articles представляет собой коллекцию статей, связанных с проектом. Использование Set<Article> обеспечивает уникальность статей и позволяет удобно управлять ими в рамках проекта. Каскадирование операций и автоматическое удаление сирот имеют цель облегчить управление жизненным циклом связанных объектов.

variants - Поле variants аналогично articles, но хранит множество вариантов, связанных с проектом. Оно также управляется с помощью аннотаций @OneToMany, предоставляя те же возможности для каскадных операций и обработки сирот. Сирота в контексте базы данных означает, что есть связанные объекты, которые больше не связаны с их родительским объектом. Например, если есть связь "один ко многим" между объектами Project и Article, и удаляется проект, все связанные с ним статьи становятся "сиротами".

user - Атрибут user представляет владельца или создателя проекта. Он связан с пользовательской сущностью через внешний ключ user_id. Эта связь позволяет отслеживать, какой пользователь ответственен за проект.

jsonData - Поле jsonData хранит данные для парсинга HTML-страницы в формате JSON.

3. Практическая часть

3.1. Настройка проекта в Vaadin Starter и его создание

Vaadin является фреймворком для создания веб-приложений на Java. Vaadin Starter — это онлайн-инструмент, который позволяет быстро настроить и скачать шаблон проекта для разработки. Он облегчает начало работы, предоставляя быстрый и удобный способ создания каркаса проекта, который можно далее развивать и настраивать под свои нужды. Для начала работы нужно перейти на сайт https://start.vaadin.com и зарегистрироваться. Потом необходимо будет создать новый проект и начать его конфигурацию.

Выберем версию Vaadin 24.

Имя проекта: Parser.

Имя пакета: gb.

Тип проекта: Maven.

Spring Boot: да.

Java версия: 21.

Теперь выберем тему оформления (светлая, темная или системная) и создадим в конструкторе необходимые страницы для фронтента (about-view, login, projects-view, forgot-password, register, banned-view, admin-view). В конструкторе можно сразу создать нужный дизайн странички со всеми кнопками и полями для ввода данных. После этого можно скачивать проект в виде ZIP-архива. После распаковки архива его можно импортировать в IDE. В данном случае в IntelliJ IDEA, в которой будет вестись дальнейшая разработка проекта. После запуска приложения при помощи класса Application оно будет доступно по адресу http://localhost:8080.

Теперь можно начать разработку, добавляя новые представления, компоненты и бизнес-логику.

3.2. Создание compose.yaml файла для базы данных

Для того, чтобы подключить базу данных, лучше всего воспользоваться приложением Docker Desktop, который обеспечивает простой способ управления контейнерами. Для этого создадим конфигурационный файл compose.yaml в директории проекта со следующими данными:

```
version: '3.8'

services:
  database:
    image: 'postgres:14-alpine'
    container_name: 'parserdatabase'
    environment:
      - 'POSTGRES_DB=parserdb'
      - 'POSTGRES_PASSWORD=password'
      - 'POSTGRES_USER=admin'
    ports:
      - '8081:5432'
    volumes:
      - './init.sql:/docker-entrypoint-initdb.d/init.sql'
```

В данном файле конфигурации описывается следующая логика для создания docker контейнера:

Имеется сервис с именем database, который использует образ postgres:14-alpine. Сервис будет иметь имя контейнера parserdatabase.

В разделе environment определены переменные окружения для базы данных, такие как имя базы данных (POSTGRES_DB=parserdb), пароль пользователя базы данных (POSTGRES_PASSWORD=password) и имя пользователя базы данных (POSTGRES_USER=admin).

Порт 5432 в контейнере базы данных будет проброшен на порт 8081 хост-системы.

Также определено, что файл init.sql из текущего каталога будет скопирован в контейнер и запущен при инициализации базы данных. Это поможет добавить в базу данных пользователей (user и admin).

Для запуска данного файла необходимо будет установить Docker Desktop и плагин для среды разработки, который интегрирует взаимодействие с Docker. После запуска данного файла в среде разработки автоматически будет скачан нужный образ и создана база данных с данными, которые будут в файле init.sql.

Теперь в проекте присутствует база данных.

3.3. Настройка application.properties

Файл application.properties обычно используется для хранения и настройки параметров и свойств приложения. Он может содержать различные настройки, такие как:

1. Пути к файлам логов, базе данных и другим ресурсам
2. Параметры аутентификации и авторизации
3. Настройки сервера, такие как порт и протокол
4. Настройки кэширования, базы данных или других внешних сервисов
5. Пользовательские настройки, такие как язык или тема оформления

В нашем проекте файл application.properties выглядит так:

```
server.port=${PORT:8080}
logging.level.org.atmosphere = warn
spring.mustache.check-template-location = false

vaadin.launch-browser=true
# PostgreSQL configuration.
spring.datasource.url = jdbc:postgresql://localhost:8081/parserdb
spring.datasource.username = admin
spring.datasource.password = password
spring.jpa.hibernate.ddl-auto = update
vaadin.allowed-packages = com.vaadin,org.vaadin,dev.hilla,gb
spring.jpa.defer-datasource-initialization = true
spring.sql.init.mode = always
```

В данном файле application.properties находятся следующие настройки:

server.port=${PORT:8080} - устанавливает порт сервера на значение переменной 8080.

logging.level.org.atmosphere = warn - устанавливает уровень логирования для пакета org.atmosphere на уровень warn.

spring.mustache.check-template-location = false - отключает проверку наличия файлов шаблонов Mustache.

vaadin.launch-browser=true - запуску браузера при запуске приложения.

spring.datasource.url = jdbc:postgresql://localhost:8081/parserdb - устанавливает URL подключения к базе данных PostgreSQL.

spring.datasource.username = admin - устанавливает имя пользователя для подключения к базе данных.

spring.datasource.password = password - устанавливает пароль для подключения к базе данных.

spring.jpa.hibernate.ddl-auto = update - устанавливает режим автоматической инициализации схемы базы данных, что означает, что схема будет обновляться при каждом запуске приложения.

vaadin.allowed-packages = com.vaadin,org.vaadin,dev.hilla,gb - устанавливает список разрешённых пакетов для Vaadin.

spring.jpa.defer-datasource-initialization = true - отключает инициализацию источника данных при запуске приложения.

spring.sql.init.mode = always - устанавливает режим инициализации базы данных на always, что означает, что скрипты инициализации будут выполнены при каждом запуске приложения.

После тонкой настройки файлов application.properties и compose.yaml можно приступать к реализации бизнес логики проекта.

3.4. Тип пользователя и права доступа

Для работы с ролями доступа в Vaadin приложениях часто используется интеграция с фреймворками безопасности, такими как Spring Security. Это позволяет более гибко настраивать права доступа, аутентификацию и авторизацию. В Vaadin приложениях реализация доступа к страницам на основе ролей играет важную роль в обеспечении безопасности. Для этого можно использовать аннотации @RolesAllowed и @AnonymousAllowed.

@AnonymousAllowed:

@AnonymousAllowed - аннотация, позволяющая доступ к странице анонимным пользователям, т.е. пользователям без аутентификации.

@RolesAllowed:

@RolesAllowed({"USER", "ADMIN"}) - аннотация, которая устанавливает требуемые роли для доступа к странице. Если пользователь не имеет хотя бы одной из указанных ролей, ему будет отказано в доступе.

Например:

```
@Route("admin")
@RolesAllowed({"USER", "ADMIN"})
public class AdminView extends VerticalLayout {
    // ...
}
```

Часто требуется комбинировать различные типы доступа. Например, страница может быть доступна анонимным пользователям, но требовать определенные роли для выполнения определенных действий. Это можно достичь путем комбинирования аннотаций.

Доступ к страницам обычно реализуется с помощью аннотации @Route и класса BeforeEnterEvent. Важно понимать, что в приложении может быть несколько роутеров, каждый из которых может иметь свою логику доступа.

Вот пример, как может выглядеть страничка About с логикой доступа на основе ролей:

```
@PageTitle("About")

@Route(value = "about-view", layout = MainLayout.class)

@RolesAllowed({"USER", "ADMIN"})

@AnonymousAllowed

@Uses(Icon.class)

public class AboutView extends Composite<VerticalLayout> implements BeforeEnterObserver {

//

}
```

Рассмотрим подробно аннотации, которые не рассматривались ранее:

@PageTitle("About") - Эта аннотация устанавливает заголовок страницы, который будет отображаться во вкладке браузера или в навигационной панели. Здесь у страницы устанавливается заголовок "About".

@Route(value = "about-view", layout = MainLayout.class) - Аннотация @Route устанавливает путь к данной странице и указывает, какой макет (layout) использовать для размещения содержимого страницы. В данном случае, страница будет доступна по пути "about-view" и будет использовать макет MainLayout.

@Uses(Icon.class) - Эта аннотация указывает на то, что данная страница использует компонент Icon, возможно для отображения иконок или других графических элементов.

public class AboutView extends Composite<VerticalLayout> implements BeforeEnterObserver - Этот класс AboutView является композитом, который содержит VerticalLayout, и реализует интерфейс BeforeEnterObserver. Это означает, что класс отслеживает событие перед входом на страницу и может выполнять определенные действия при этом событии.

В случае со страницей AboutView перед входом на страничку будет выполнен метод **beforeEnter**, который определен в следующем методе:

```
public void beforeEnter(BeforeEnterEvent event) {
    if (authenticatedUser.get().isPresent()) {
        User user = authenticatedUser.get().get();
        if (user.isBanned()) { event.forwardTo("banned-view"); }
    }
}
```

Рассмотрим подробно, что делает данный метод:

beforeEnter - это метод, который реализует интерфейс BeforeEnterObserver. Он вызывается перед входом на страницу, и его задача - проверить, есть ли у пользователя необходимые права доступа к странице.

if (authenticatedUser.get().isPresent()) - здесь проверяется, аутентифицирован ли пользователь. **authenticatedUser** - это объект, который содержит информацию о зарегистрированном пользователе. Метод **get()** возвращает опциональный объект User, который может содержать информацию о пользователе, если он аутентифицирован. Метод **isPresent()** проверяет, содержит ли опциональный объект значение. Если пользователь аутентифицирован, то выполняется следующий блок кода.

User user = authenticatedUser.get().get(); - здесь из опционального объекта **User** извлекается объект **User**. Если пользователь не аутентифицирован, то будет выброшено исключение **NoSuchElementException**.

if (user.isBanned()) - здесь проверяется, заблокирован ли пользователь. Метод **isBanned()** возвращает булево значение, указывающее, заблокирован ли пользователь. Если пользователь заблокирован, то его перенаправит на страницу **banned-view**.

BeforeEnterObserver может быть реализован любым классом, который используется в качестве страницы в Vaadin приложении. Это позволяет легко добавлять логику доступа к страницам в разные части приложения.

В целом, настройка доступа в Vaadin приложениях важна для обеспечения безопасности и контроля над функциональностью, которая доступна для различных категорий пользователей. Правильное использование аннотаций и интерфейсов позволяет создать надежную систему управления доступом, обеспечивая безопасность и удобство использования приложения для всех пользователей.

3.5. Обзор страницы Login View

Рассмотрим класс **LoginView**, который определяет вид страницы входа в систему для веб-приложения, созданного с использованием фреймворка Vaadin. Рассмотрим его составные части и функциональность.

```
public class LoginView extends VerticalLayout implements BeforeEnterObserver {

    private final AuthenticatedUser authenticatedUser;

    private UserRepository userRepository;

    private LoginForm login = new LoginForm();

    // Другой код }
```

В данном коде:

private final AuthenticatedUser authenticatedUser; - поле для хранения информации об аутентифицированном пользователе.

private UserRepository userRepository; - поле для работы с репозиторием пользователей, предположительно для доступа к данным пользователей в базе данных.

private LoginForm login = new LoginForm(); - создание формы входа, которая будет использоваться для аутентификации пользователей.

Далее выполняется следующий код:

```
public LoginView(UserRepository userRepository, AuthenticatedUser
authenticatedUser) {
        this.userRepository = userRepository;
        this.authenticatedUser = authenticatedUser;
        addClassName("login-view");
        setSizeFull();
        setAlignItems(Alignment.CENTER);
        setJustifyContentMode(JustifyContentMode.CENTER);
        login.setAction("login");
    // Другой код
```

В конструкторе **LoginView** происходит инициализация и настройка элементов пользовательского интерфейса.

this.userRepository = userRepository; - Эта строка присваивает значение переменной **userRepository**, которая является полем класса **LoginView**. Здесь происходит передача экземпляра класса **UserRepository** в конструктор **LoginView**, чтобы он мог быть использован внутри этого класса.

this.authenticatedUser = authenticatedUser; - Эта строка также присваивает значение переменной **authenticatedUser**, которая является полем класса **LoginView**. Здесь происходит передача экземпляра класса **AuthenticatedUser** в конструктор **LoginView**, чтобы он мог быть использован внутри этого класса.

addClassName("login-view"); - Эта строка добавляет класс CSS **login-view** к виду **LoginView**. Это позволяет применить определенные стили к этому виду, например, для визуального оформления.

setSizeFull(); - Эта строка устанавливает размер виджета **LoginView** в полный размер, чтобы он занимал всю доступную область.

setAlignItems(Alignment.CENTER); - Эта строка выравнивает элементы внутри виджета LoginView по вертикали по центру.

setJustifyContentMode(JustifyContentMode.CENTER); - Эта строка выравнивает элементы внутри виджета LoginView по горизонтали по центру.

login.setAction("login"); - Эта строка создает форму входа login. В данном случае, действие установлено на "login", что означает, что данные формы будут отправлены на маршрут с именем "login".

login.addForgotPasswordListener(e -> { ... }); - Эта строка добавляет обработчик события "забыли пароль". Когда пользователь нажимает на кнопку "Забыли пароль", этот обработчик будет вызван, и внутри него будет выполнено перенаправление на страницу восстановления пароля.

Теперь необходимо добавить кнопки и обработчики событий:

```
login.addForgotPasswordListener(e -> {
        UI currentUI = UI.getCurrent();
        if (currentUI != null) {currentUI.navigate("forgot-password"); }
});
```

Эта строка добавляет обработчик события "забыли пароль" к форме входа login. Когда пользователь нажимает на кнопку "Забыли пароль", этот обработчик перенаправляет пользователя на страницу восстановления пароля с помощью currentUI.navigate("forgot-password");.

Аналогично для кнопки **registerButton** происходит перенаправление на страницу **register**:

```
Button registerButton = new Button("Register", e -> {
        UI currentUI = UI.getCurrent();
        if (currentUI != null) {currentUI.navigate("register");}
});
```

Для кнопки **aboutButton** происходит перенаправление на **about-view**. И после этого вся форма для логина собирается вместе строчкой, в которой происходит добавление заголовка “**Welcome to Parser!**”:

```
add(new H1("Welcome to Parser!"), login, registerButton, aboutButton);
```

Также для странички **LoginView** имеется **BeforeEnterEvent**, который проверяет, авторизован ли пользователь и перенаправляет его на определенные страницы в зависимости от его роли, если это так.

3.6. Получение IP клиента и обмен REST сообщениями

REST (Representational State Transfer) - это архитектурный стиль взаимодействия компонентов распределенного приложения в сети. REST сообщения используются для построения масштабируемых веб-сервисов благодаря их простоте и возможности использования стандартных HTTP-методов, таких как GET, POST, PUT и DELETE.

Суть REST в том, что веб-сервисы и их клиенты обмениваются сообщениями, содержащими данные в понятном для обеих сторон формате (чаще всего JSON или XML). Каждый ресурс в REST-сервисе идентифицируется своим URI (Uniform Resource Identifier) и может быть манипулирован с использованием стандартных методов протокола HTTP.

Как происходит обмен REST сообщениями?

Обмен REST сообщениями осуществляется через HTTP-запросы и ответы. Клиент отправляет запрос на сервер, указывая метод HTTP, URI ресурса, заголовки и, при необходимости, тело сообщения. Сервер обрабатывает запрос, выполняет соответствующие действия и отправляет ответ, содержащий статус операции и, при необходимости, данные.

Выбор IP для обмена REST сообщениями

Выбор IP-адреса для взаимодействия с REST-сервисом — ключевой момент, влияющий на доступность и безопасность сервиса. Рассмотрим пример кода на Java, который позволяет выбирать между локальным и публичным IP для обмена REST сообщениями:

```
RadioButtonGroup chooseIP = new RadioButtonGroup();
chooseIP.setItems("Local IP: " + localIp, "Public IP: " + publicIp);
chooseIP.setValue("Local IP: " + localIp);
```

В данном коде используется **localIp** :

```
String localIp = InetAddress.getLocalHost().getHostAddress();
```

и метод **getPublicIp**:

```
public String getPublicIp() {
    try {
        URL url = new URL("https://api.ipify.org");
        URLConnection connection = url.openConnection();
        BufferedReader in = new BufferedReader(new
InputStreamReader(connection.getInputStream()));
        String ip = in.readLine();
        in.close();
        return ip;
    } // обработка исключений
}
```

В этом коде используется компонент RadioButtonGroup для выбора между локальным и публичным IP-адресами. Пользователь выбирает один из предложенных вариантов, и на основе этого выбора формируется базовый URL для REST-сообщений. Событие addValueChangeListener обрабатывает изменение значения и обновляет переменную ipForREST, которая хранит текущий выбранный IP-адрес с указанным портом.

Использование локального IP-адреса обычно предпочтительно в среде разработки или тестирования, когда сервисы общаются внутри одной закрытой сети. Публичный IP-адрес необходим для обращения к сервисам, развернутым в облаке или на серверах, доступных из интернета.

Такой подход позволяет гибко управлять настройками подключения и адаптировать сервисы для работы в различных сетевых условиях, обеспечивая их доступность и надежность.

Теперь можно рассмотреть метод для отправки тестового POST сообщения для клиента. Сперва создадим кнопку и привяжем действие:

```
Button sendTestPostButton = new Button("Send test POST");
sendTestPostButton.addClickListener(e -> {
    sendTestPost(ipForREST);
});
```

Метод **sendTestPost** представляет собой функцию, которая используется для отправки POST запроса по заданному URL-адресу. Код метода:

```
private void sendTestPost(String ipForREST) {
    try {
        URL url = new URL(ipForREST + "/hello");
            HttpURLConnection connection = (HttpURLConnection) url.openConnection();
        connection.setRequestMethod("POST");
        connection.setDoOutput(true);
        int responseCode = connection.getResponseCode();
        if (responseCode == HttpURLConnection.HTTP_OK) {
            Notification.show("Request sent successfully to: " + ipForREST, 5000, Notification.Position.MIDDLE);
        } else {
            Notification.show("Failed to send request to: " + ipForREST, 5000, Notification.Position.MIDDLE);
        }
    } // обработка исключений
```

Формирование URL и соединения:

В методе создается объект URL на основе переданного ipForREST и пути /hello. Затем устанавливается соединение с этим URL с помощью HttpURLConnection.

Настройка запроса:

Устанавливается метод запроса как "POST" с помощью connection.setRequestMethod("POST"), указывая серверу, что мы хотим создать новый ресурс. Устанавливается флаг connection.setDoOutput(true), чтобы разрешить вывод данных в тело запроса.

Отправка запроса и обработка ответа:

После настройки запроса, выполняется отправка запроса на сервер с помощью connection.getResponseCode(). Если ответ сервера имеет статус HTTP_OK (код 200), то выводится уведомление об успешной отправке запроса. В случае ошибки (например, если сервер вернул другой статус), выводится уведомление о неудачной отправке запроса.

Обработка исключений:

В случае возникновения исключения типа ConnectException, которое указывает на отказ соединения, выводится сообщение об ошибке и уведомление. Если возникает любое другое исключение типа IOException, оно просто печатается в стандартный поток ошибок.

Этот метод позволяет асинхронно отправлять POST запросы на указанный URL и обрабатывать ответы от сервера. Обработка исключений помогает корректно реагировать на проблемы с соединением или обменом данными. Важно учитывать, что данный метод отправляет только запрос, но не обрабатывает ответы или данные, которые могут быть возвращены сервером в результате запроса.

3.7. Клиентская часть приложения и рабочая логика

Клиентская часть написана на Java версии 8, что обеспечивает ее запуск на любом ПК с последней версией JVM. Графический интерфейс создан при помощи библиотеки Java Swing.

Интерфейс программы представляет из себя окно для ввода порта, с которого будут считываться и отправляться REST сообщения, кнопка запуска сервера, кнопка остановки сервера и кнопка завершения программы. Код выглядит таким образом:

```
JPanel panel = new JPanel(new GridLayout(4, 1));

panel.add(portField);

panel.add(startButton);

panel.add(stopButton);

panel.add(exitButton);
```

Создание кнопки, запускающей сервер выполняется следующим образом:

```
startButton = new JButton("Start client");
```

Далее в блоке try-catch выполняется следующий код:

```
server = HttpServer.create(new InetSocketAddress(port), 0);

server.createContext("/project", new MyHandler());

server.setExecutor(null);
```

Теперь необходимо создать поток, отвечающий за работу сервера:

```
new Thread(() -> {
    server.start();
}).start();
```

И переключить возможность нажатия кнопок:

```
startButton.setEnabled(false);
stopButton.setEnabled(true);
```

После того как сервер запущен, он может работать с REST-сообщениями. Это делается классом **MyHandler**, который реализует интерфейс HttpHandler из библиотеки com.sun.net.httpserver. Этот класс используется для обработки HTTP-запросов на сервере.

В методе **handle**(HttpExchange exchange) проверяется тип HTTP-запроса. Если запрос является POST-запросом, то вызывается метод handlePostRequest(HttpExchange exchange). В противном случае, вызывается метод sendDefaultResponse(HttpExchange exchange), который отправляет стандартное сообщение ответа на сервер. Метод **handle**:

```
@Override
    public void handle(HttpExchange exchange) throws IOException {
        if ("POST".equals(exchange.getRequestMethod())) {
            handlePostRequest(exchange);
        } else {sendDefaultResponse(exchange);}
    }
```

В методе **handlePostRequest**(HttpExchange exchange) считывается тело запроса, которое затем анализируется для определения того, какая операция должна быть выполнена. Если запрос содержит параметр "update", вызывается метод **processUpdateRequest**(String requestData), который обрабатывает запрос с параметром "update". Если запрос содержит параметр "pause", вызывается метод processPauseRequest(String requestData), который обрабатывает запрос с параметром "pause". Если запрос содержит параметр "run", вызывается метод **processRunRequest**(String requestData), который обрабатывает запрос с параметром "run". Если запрос содержит параметр "report", вызывается метод **processReportRequest**(String requestData), который обрабатывает запрос с параметром "report".

Рассмотрим следующий код:

```
String query = exchange.getRequestURI().getQuery();
    if (query != null && query.contains("update")) {
        sendResponse(exchange, "Project updated successfully!");
        processUpdateRequest(requestBody.toString());
    }
```

Этот код проверяет, содержит ли строка запроса (**query**), извлеченная из URI запроса в объекте HttpExchange, параметр "**update**". Если параметр "**update**" присутствует в строке запроса и не равен null, то выполняются следующие действия:

Вызывается метод **sendResponse**(exchange, "Project updated successfully!"), который отправляет ответ на запрос с сообщением "Project updated successfully!". **На этом этапе создаются необходимые файлы для запуска проекта на компьютере клиента.**

Это файлы:

Articles.txt - список всех артикулов проекта.

Downloaded.txt - пока пустой файл, в него будут записаны скачанные артикулы.

Extension.txt - в этом файле хранится расширение, с котоорым будут сохраняться файлы.

MissedArticles.txt - артикулы, для которых не удалось ничего скачать.

Need to download.txt - те артикулы, которые сейчас находятся в очереди на скачивание. Изначально идентичен **Articles.txt.**

Parameters.txt - параметры для парсера.В этом файле содержится информация по какому тексту искать ссылку.

Variants.txt - список всех сайтов, по которым нужно произвести поиск. Варианты сайтов нужны для парсинга по разным регионам, тк контент на таких сайтах может отличаться.

Затем вызывается метод **processUpdateRequest**(requestBody.toString()), который обрабатывает запрос, используя тело запроса, преобразованное в строку.

Аналогично выполнены методы для **pause, run и report**. Однако, методы **run** и **pause** стоит рассмотреть более подробно, тк они содержат важную рабочую логику, связанную с остановкой и возобновлением исполнения скачивания файлов.

```
if (query != null && query.contains("run")) {
        if (DownloadFiles.isRunning) {
            sendResponse(exchange, "Project is already running!");
        } else {
            sendResponse(exchange, "Project started successfully!");
            new Thread(() -> {
                processRunRequest(requestBody.toString());
            }).start();
        }
    }
```

В этом коде идет предварительная проверка переменной **isRunning** в классе **DownloadFiles**. Переменная **isRunning** представляет из себя:

```
public static volatile boolean isRunning = false;
```

Использование свойств volatile и static вместе придаст переменной **isRunning** следующие свойства:

1. Атомарность обновления: Ключевое слово volatile гарантирует, что изменения значения переменной isRunning будут видны для всех потоков, работающих с этой переменной. Без ключевого слова volatile, значения переменной могут быть кэшированы в локальных потоках, что может привести к некорректному поведению при многопоточной обработке.

2. Общий доступ: Ключевое слово static означает, что переменная isRunning принадлежит классу, а не конкретному экземпляру класса. Это позволяет использовать переменную isRunning напрямую через имя класса, например MyClass.isRunning. Статическая переменная доступна для всех экземпляров класса и может быть доступна без необходимости создания экземпляра класса.
3. Сохранение значений между вызовами методов: Статическая переменная сохраняет свои значения между вызовами методов и может использоваться для обмена информацией между различными методами класса. Ключевое слово volatile гарантирует, что значение переменной будет обновляться атомарно, что предотвращает одновременное обновление значения в разных потоках.

Таким образом, использование свойств volatile и static вместе в объявлении переменной isRunning гарантирует, что значение переменной будет обновляться атомарно, будет доступно для всех потоков и сохраняться между вызовами методов. Это полезно в ситуациях, когда необходимо обеспечить общий доступ к переменной, сохранить общее состояние и предотвратить одновременное обновление значения в разных потоках. Изменение этой переменной позволяет останавливать цикл выполнения программы.

Метод **processRunRequest** получает из запроса необходимые данные для запуска нужного проекта и запускает метод **run** в классе **DownloadFiles**. Основная часть метода выглядит так:

```
JSONObject jsonObject = new JSONObject(requestData);

String userAgent = jsonObject.getString("userAgent");

String title = jsonObject.getString("title");

DownloadFiles.run(title, userAgent);
```

После запуска метода **run** в классе **DownloadFiles** переменной **isRunning** присваивается значение **true** и это дает возможность прервать ее выполнение следующим кодом при помощи переменной **shouldStop** :

```
if (shouldStop) {
  break;
}
```

Переменная **shouldStop** в классе **DownloadFiles**:

```
public static volatile boolean shouldStop = false;
```

Теперь рассмотрим каким образом выполнена рабочая логика. Сперва создаются следующие списки и коллекция:

```
Map<Integer, String> articlesMap = readArticlesFromFile(title);

List<String> variants = readVariantsFromFile(title);

List<Integer> keysToRemove = new ArrayList<>();
```

В данном случае метод **readArticlesFromFile(title)** считывает артикулы из файла по передаваемому наименованию проекта. Аналогично делается и в методе **readVariantsFromFile(title)**. Список **keysToRemove** будет использован для хранения ключей тех артикулов, которые успешно скачались. Эти артикулы потом будут удалены из очереди на скачивание, если пользователь захочет остановить выполнение программы.

После этого идет выполнение методов, которые считывают параметры для парсинга и расширение файла для сохранения. Методам передается название проекта для поиска файлов:

```
readParametersFromFile(title);

readFileExtensionFromFile(title);
```

Метод **readFileExtensionFromFile(String title)** выглядит так:

```
public static void readFileExtensionFromFile(String title) {
    try {
        FileReader fileReader = new FileReader(title + "/Extension.txt");
        BufferedReader bufferedReader = new BufferedReader(fileReader);
        fileExtension = bufferedReader.readLine();
        bufferedReader.close();
        fileReader.close();
    } // Обработка исключений
}
```

После этого запускается цикл по артикулам, а внутри него запускается вложенный цикл по вариантам сайтов для парсинга. Циклы не показаны для простоты восприятия:

String articleValue = article.getValue();

Integer articleKey = article.getKey();

String url = variant + articleValue;

String imageLink = getImageLink(url);

downloadImage(imageLink, articleValue);

keysToRemove.add(articleKey);

String articleValue = article.getValue(); и **Integer articleKey = article.getKey();** получают текущие артикул и вариант сайта.

String url = variant + articleValue; - получаем URL.

String imageLink = getImageLink(url); - находим ссылку.

downloadImage(imageLink, articleValue); - скачиваем файл.

Это было краткое описание логики, которая стоит за скачиванием файлов. Полный код можно найти в Приложении.

4. Полное описание возможностей приложения

Приложение Parser построено по клиент-серверной архитектуре с использованием фреймворков Spring и Vaadin. Сервер подключен к базе данных на PostgreSQL, где хранятся данные пользователей и их проектов. Для начала работы пользователю необходимо зайти в личный кабинет, скачать клиент, запустить его и приступить к работе над проектом в браузере. После регистрации пользователь попадает на страницу **About**, где ему предлагается прочитать о создании его первого проекта и запустить его, скачав несколько файлов. На странице **Projects** пользователь может использовать вкладки **Download desktop client, Create and select project, Modify project, Run project**.

Вкладка **Download desktop client** позволяет:

1. Скачать десктопный клиент.
2. Посмотреть свой локальный и публичный апйпи.
3. Выбрать айпи для тестирования клиента.
4. Протестировать работоспособность клиента, отправив POST запрос.

Вкладка **Create and select project** позволяет:

1. Создать проект кнопкой **Create project**, введя название и описание проекта.
2. Обновить проект кнопкой **Update project**.
3. Открыть проект кнопкой **Open project**.
4. Удалить проект кнопкой **Delete project**.
5. Просмотреть список всех проектов и их описание в таблице.
6. Посмотреть список всех артикулов проекта.
7. Добавить артикулы к проекту.
8. Удалить все артикулы у проекта.

9. Посмотреть список всех вариантов сайтов (локализаций) для проекта.
10. Добавить варианты сайтов к проекту.
11. Удалить все варианты сайтов у проекта.

Вкладка **Create and select project** позволяет:

1. Ввести параметры для поиска ссылок на скачиваемые файлы.
2. Вычислить смещение для параметров.
3. Обновить параметры.
4. Выбрать сколько артикулов протестировать.
5. Протестировать выбранное количество артикулов, попытавшись найти ссылку на скачиваемый файл.
6. Текстовое окно с результатами поиска ссылки для тестового артикула (артикулов).
7. Текстовое окно, в котором можно просмотреть исходный HTML-код странички, адрес которой нужно ввести в поле enter **URL to to view HTML code** и нажать на кнопку **View HTML code**.

Вкладка **Run project** позволяет:

1. Выбрать айпи для тестирования клиента и дальнейшей работы.
2. Протестировать работоспособность клиента, отправив POST запрос.
3. Ввести расширение, с которым будут сохраняться файлы.
4. Кнопка **Update** - для обновления параметров проекта.
5. Кнопка **Run / Resume** - для запуска и возобновления проекта
6. Кнопка **Pause** - для приостановки выполнения проекта.
7. Кнопка **Report** для показа отчета.

Пользователи, которые имеют роль администратора, могут зайти страницу **About** и на страницу **Administration**, на которой можно выполнять следующий функционал:

1) Просматривать текущих пользователей приложения и список их проектов.
2) Изменять имя, логин, электронную почту и статус бана.
3) Удалять пользователей.
4) Выполнять поиск и сортировку в таблице пользователей.

5. Недостатки архитектуры, фреймворков и приложения

Большинство недостатков вытекает из фреймворка **Vaadin**, который был выбран для фронтенд части приложения ввиду своей легкости в создании фронтенд части. Тк основной особенностью данного фреймворка является написание бэкенда и фронтенда в одном классе (который представляет собой одну страничку), то для больших проектов код получается раздутым и плохо читаемым.

В **Vaadin** сложно создавать и отображать динамические структуры данных для фронтенда, из-за чего реализация конструктора операций (где можно самому создавать свои шаблоны для выполнения операций и менять их порядок для каждого проекта) будет вызывать серьезные сложности.

Так же фреймворк **Vaadin** может потреблять больше ресурсов, таких как память и процессорное время, по сравнению с другими фреймворками. Это может быть особенно заметно при работе с большими данными или сложными пользовательскими интерфейсами.

Когда к серверу подключается много пользователей, то нагрузка на него серьезно возрастает, тк все сессии должны храниться в памяти сервера, что может вызвать переполнение оперативной памяти и, как следствие, неработоспособность сервера.

6. Возможные улучшения приложения

Приложение возможно улучшить, добавив следующие функции:

1. Генератор шаблонов для проектов с реализацией разного количества операций и их порядка, тк разные сайты требуют разного подхода. Например, в текущем варианте проекта нет возможности перехода по ссылкам и поиске на сайте по артикулу.

2. Добавление других способов для парсинга контента, например по тэгам HTML-файла.

3. Платная подписка на приложение с различными опциями (оплата за количество скачанных файлов или безлимитная подписка).

4. Двухфакторная аутентификация OAuth 2.0.

5. Переход на другой фронтенд фреймворк для улучшения производительности сервера.

7. Как запускать проект

После скачивания основного проекта по ссылке:

https://github.com/Minyosha/Parser

Ссылка на проект с десктопным клиентом:

https://github.com/Minyosha/Parser_Desktop_Client

Необходимо открыть проект в IntelliJ IDEA и запустить файл:

compose.yaml.

Далее можно запускать созданный контейнер в Docker Desktop и потом запускать сам сервер, запустив класс Application.

После первого запуска и инициализации базы данных необходимо в файле **application.properties** закомментировать строку:

spring.sql.init.mode = true

Это необходимо сделать, чтобы спринг не пытался повторно заполнить базу данных значениями при запуске сервера, что вызывает ошибку.

После запуска сервера браузер откроется на странице **Login**, где нужно будет ввести логин и пароль. Используйте логин и пароль **user** для того, чтобы зайти в качестве пользователя (или можете зарегистрировать новый аккаунт) или логин и пароль **admin**, чтобы зайти в качестве администратора.

После успешной авторизации откроется страничка **About**, где можно почитать больше о проекте.

8. Экономическая эффективность проекта

Для упрощения примем, что программа скачивает файл за 5 секунд. Программа не ошибается и работает с одной и той же скоростью. Допустим, что скачать 1 файл программой будет стоить 10 рублей (с учетом всех расходов на подготовку проекта, организацию рабочего места и зарплату оператора, который может запускать несколько проектов на скачивание сразу или работать над другими проектами). Оператор имеет месячную зарплату от 30000 рублей. Также работодатель несет дополнительные расходы в виде налогов и организации рабочего места оператора, обычно это 100-150% от зарплаты оператора. Рабочий день оператора 8 часов. Оператор работает 5 дней в неделю. На поиск и скачивание 1 файла оператору в среднем требуется от 1 до 10 минут. Если приблизительно прикинуть, то экономический расчет для скачивания 10000 файлов оператором и программой будет приблизительно такой:

Скачивание с помощью программы:

Время на скачивание 10000 файлов: 10000 * 5 секунд = 50000 секунд

Затраты на скачивание 10000 файлов: 10000 * 10 рублей = 100000 рублей

Количество дней, необходимых для выполнения задачи:

Время на скачивание 10000 файлов вручную: 10000 * (1-10) * 60 секунд / 8 часов = 31,25-312,5 дней

Время на скачивание 10000 файлов с помощью программы: 50000 секунд / 8 часов = 1,74 дня

Вилка затрат:

Минимальные затраты на скачивание 10000 файлов вручную: 100000 рублей + 62500 рублей + 125000 рублей = 287500 рублей

Максимальные затраты на скачивание 10000 файлов вручную: 100000 рублей + 62500 рублей + 187500 рублей = 450000 рублей

Средние затраты на скачивание 10000 файлов вручную: (287500 рублей + 450000 рублей) / 2 = 358750 рублей

Затраты на скачивание 10000 файлов с помощью программы = 100000 рублей

Вывод:

Программа оказывается значительно эффективнее и дешевле.

8. Заключение

Представленный дипломный проект является значимым вкладом в упрощение и ускорение процесса наполнения интернет-магазинов контентом. Реализованный с применением современных технологий Spring и Vaadin, он предлагает удобный интерфейс для взаимодействия пользователей с системой, а также автоматизирует рутинные операции по скачиванию файлов с веб-сайтов.

Неоспоримым преимуществом системы является ее способность обеспечивать парсинг и загрузку файлов, что значительно ускоряет подготовку необходимого материала. Архитектурное решение, включающее в себя клиент-серверное взаимодействие и базу данных, гарантирует организованность и безопасность процесса обработки информации. Дифференцированный подход к ролям пользователей позволяет администраторам эффективно управлять системой, в то время как обычные пользователи могут концентрироваться на работе над своими проектами.

Однако, несмотря на перечисленные достоинства, проект имеет ряд ограничений, обусловленных выбором фреймворка Vaadin. Проблемы с масштабируемостью, читаемостью кода и производительностью при работе с большим объёмом данных или высокой нагрузкой на сервер требуют дальнейшего рассмотрения и оптимизации.

В перспективе, развитие проекта может быть направлено на усовершенствование функциональности, включая генератор шаблонов для проектов, расширение методов парсинга, введение платных подписок, улучшение безопасности и переход на более гибкие фреймворки для фронтенда. Эти улучшения позволят повысить эффективность работы с системой, а также расширить потенциал её применения в различных бизнес-сценариях.

9. Список используемой литературы

1. Шилдт Герберт Java полное руководство / Герберт Шилдт. – М : ООО "И.Д. Вильяме",, 2015. – 1376 с.
2. Браун Э. Изучаем JavaScript. Руководство по созданию современных веб-сайтов / Э. Браун. – Москва : Альфа-книга, 2017. – 368 с.
3. Vaadin Cookbook URL: https://cookbook.vaadin.com/? (дата обращения: 25.04.2024).
4. Baeldung URL: https://www.baeldung.com/ (дата обращения: 21.04.2024).
5. Spring URL: https://spring.io/guides (дата обращения: 21.04.2024).
6. Моргунов Е. П. PostgreSQL. Основы языка SQL: учебное пособие / Е. П. Моргунов. – СПб. : БХВ-Петербург, 2018. – 336 с. – ISBN 978-5-9775-4022-3

10. Приложения

1. https://github.com/Minyosha/Parser
2. https://github.com/Minyosha/Parser_Desktop_Client