**Geekbrains**

**Разработка веб приложения на языке JAVA «Система управления складом шин с аутентификацией пользователей и интеграцией с Excel».**

IT-специалисты:

Разработка веб приложений

Мысаков И.Е

Базан М.А

2025г

**Дипломный проект**

**«Разработка системы управления складом шин с аутентификацией пользователей и интеграцией с Excel»**

**Содержание**

Введение…………………………………………………………...........3

Глава 1. Пред проектная стадия……………………………………...4

* 1. Порядок проведения работ…………………………………….4
  2. Определение требований к приложению……………………6
  3. Выбор стека технологий………………………………………..9

Глава 2. Проектная стадия……………………………………………11

2.1. Проектирование общей архитектуры приложения…………11

2.2. Проектирование web-интерфейса приложения………........21

2.3. Проектирование базы данных…………………………………30

2.4. Проектирование контроллеров………………………………..36

2.5. Проектирование «парсера»…………………………………........58

Заключение……………………………………………………………..66

Список использованной литературы……………………………….67

# **Введение**

# В современном мире, где логистика и управление запасами играют ключевую роль в успехе бизнеса, эффективная организация складского учета является неотъемлемой частью обеспечения стабильной и бесперебойной работы. Особую актуальность приобретают системы, способные автоматизировать процессы учета, сокращая время на обработку данных и минимизируя человеческий фактор. Данная дипломная работа посвящена разработке веб-приложения для управления складом шин, с использованием современных технологий и инструментов разработки. В качестве основного фреймворка для создания веб-приложения выбран Spring Boot, который позволяет быстро и эффективно разрабатывать веб-приложения с использованием Java. Для хранения данных используется реляционная база данных PostgreSQL. Для загрузки данных из Excel-файлов применяется библиотека Apache POI. В работе также реализована система аутентификации пользователей для обеспечения безопасности доступа к административным функциям приложения. Разработанное приложение позволит автоматизировать процессы учета шин на складе, предоставляя пользователям удобный интерфейс для просмотра и поиска информации о складских запасах. Целью работы является демонстрация возможностей современных технологий в решении задач управления складом и практическое применение знаний в области веб-разработки и баз данных.

**Глава 1. Предпроектная стадия**

**1.1. Порядок проведения работ**

В ходе создания проекта были проведены следующие виды работ:

* **Анализ предметной области:** Исследованы текущие процессы управления складом шин, определены требования к новой системе, включая функциональные и нефункциональные (безопасность, производительность, масштабируемость).
* **Проектирование базы данных:** Разработана схема базы данных (ERD), определены таблицы и связи между ними для хранения информации о шинах, складах, пользователях и их взаимодействиях. Определены типы данных, ключи, ограничения целостности.
* **Проектирование архитектуры приложения:** Разработана архитектурная схема приложения, включая слои (представление, сервис, модель, репозиторий) и взаимодействие между ними. Описаны используемые технологии и фреймворки.
* **Реализация модели данных:** Созданы сущности Java для представления данных из базы данных, включая аннотации JPA для отображения в базе данных.
* **Реализация сервисов:** Созданы сервисные классы для обработки бизнес-логики, таких как добавление, удаление, поиск данных о шинах и складах, аутентификация пользователей.
* **Реализация контроллеров:** Разработаны контроллеры для обработки HTTP-запросов от пользователей, включая обработку запросов на загрузку данных из Excel, отображение списков шин и складов, аутентификацию.
* **Реализация механизма загрузки данных:** Разработана функциональность для импорта данных из файлов Excel в базу данных, включая обработку ошибок и валидацию данных.
* **Реализация пользовательского интерфейса:** Разработаны HTML-шаблоны (например, Thymeleaf) для представления данных пользователю, включая страницы для авторизации, отображения списка шин, складов и формы для загрузки данных.
* **Реализация механизма аутентификации:** Реализована система аутентификации пользователей, включая хранение данных о пользователях и проверку паролей.
* **Тестирование:** Проведены unit-тесты для сервисных классов и контроллеров, а также интеграционное тестирование для проверки взаимодействия различных компонентов. Проведено тестирование пользовательского интерфейса.
* **Документирование:** Разработана полная документация к проекту, включая описание архитектуры, базы данных, кода и пользовательского интерфейса.

**1.2. Определение требований к приложению**

В этом разделе мы подробно рассмотрим требования, предъявляемые к разрабатываемому веб-приложению для управления складом шин. Требования разделены на функциональные, описывающие, что система должна уметь делать, и нефункциональные, описывающие, как система должна работать.

**1.2.1. Функциональные требования**

Функциональные требования описывают конкретные действия, которые система должна выполнять для удовлетворения потребностей пользователей. В данном случае, мы выделяем следующие основные функции:

1. **Аутентификация пользователей:**

* Система должна предоставлять возможность пользователям аутентифицироваться с использованием имени пользователя и пароля.
* После успешной аутентификации пользователь должен получить доступ к административной панели управления системой.
* Система должна предоставлять механизм для защиты административных функций от неавторизованного доступа.

1. **Загрузка данных из Excel:**
   * Система должна предоставлять интерфейс для загрузки данных о шинах и складах из Excel-файлов.
   * Система должна автоматически обрабатывать загруженные данные и сохранять их в базе данных.
   * Система должна осуществлять валидацию загруженных данных, проверяя их корректность и целостность.
   * Система должна уведомлять пользователя об ошибках во время загрузки файла.
2. **Просмотр списка шин:**
   * Система должна предоставлять возможность просмотра списка всех шин, хранящихся в базе данных.
   * Список шин должен включать основную информацию о каждой шине, такую как артикул, наименование и другие необходимые сведения.
   * Система должна обеспечивать возможность отображения списка шин как в табличном виде.
3. **Просмотр списка складов:**
   * Система должна предоставлять возможность просмотра списка всех складов, зарегистрированных в базе данных.
   * Список складов должен включать наименование каждого склада и другую необходимую информацию.
   * Система должна обеспечивать возможность отображения списка складов как в табличном виде.
4. **Поиск шин:**
   * Система должна предоставлять возможность поиска шин по артикулу.
   * Система должна предоставлять возможность поиска шин по наименованию.
   * Система должна предоставлять возможность поиска шин по складу.
   * Результаты поиска должны отображаться в удобном для пользователя формате.
5. **Управление данными (базовые операции):**
   * Система должна предоставлять возможность удаления всех данных из базы данных, с последующим сбросом идентификаторов(auto\_increment)
6. **Вывод данных:**
   * Система должна выводить корректные данные по запросу пользователя.
   * В случае каких-либо ошибок выводить понятные сообщения об ошибках.

**1.2.2. Нефункциональные требования**

Нефункциональные требования описывают, как система должна работать, а не что она должна делать. Эти требования влияют на качество, надежность и производительность системы. В данном проекте выделяем следующие нефункциональные требования:

1. **Производительность:**
   * Система должна обеспечивать быстрое время отклика на запросы пользователей.
   * Система должна быстро обрабатывать загрузку данных из Excel-файлов.
   * Время загрузки данных не должно превышать (например) 5 минут.
2. **Надежность:**
   * Система должна быть надежной и стабильной в работе.
   * Система не должна приводить к потере данных.
   * Ошибки в работе системы должны обрабатываться корректно и не приводить к сбоям.
3. **Безопасность:**
   * Система должна обеспечивать защиту от несанкционированного доступа к данным.
   * Пароли пользователей должны храниться в базе данных в хешированном виде.
   * Доступ к административной панели должен быть доступен только авторизованным пользователям.
4. **Масштабируемость:**
   * Система должна быть спроектирована с учетом возможности масштабирования для обработки большого количества данных и пользователей.
5. **Удобство использования:**
   * Система должна иметь интуитивно понятный и простой пользовательский интерфейс.
   * Пользователь должен иметь возможность легко перемещаться по системе и выполнять необходимые действия.
6. **Поддержка:**
   * Система должна быть легко поддерживаемой.
   * Код системы должен быть документирован.

**Заключение:**

Данный раздел предоставляет четкое описание всех требований к разрабатываемому приложению, которые являются основой для дальнейших этапов проектирования, разработки и тестирования системы. Определение функциональных и нефункциональных требований позволит создать систему, которая будет соответствовать потребностям пользователей и обеспечивать надежную и эффективную работу.

**1.3. Выбор стека технологий**

В данном разделе представлено обоснование выбора технологического стека, используемого при разработке веб-приложения для управления складом шин. Выбор технологий производился на основе анализа требований к приложению, их надежности, масштабируемости, а также доступности и простоты использования.

1. **Язык программирования: Java**
   * **Обоснование:** Java является кроссплатформенным, объектно-ориентированным языком программирования, что делает его идеальным выбором для разработки корпоративных приложений. Java обладает развитой экосистемой, большим количеством библиотек и фреймворков, что упрощает и ускоряет процесс разработки. Java также широко используется в индустрии, что обеспечивает высокую востребованность разработанного приложения и возможность его дальнейшей поддержки.
2. **Фреймворк: Spring Boot**
   * **Обоснование:** Spring Boot выбран в качестве основного фреймворка для веб-приложения благодаря своей простоте и удобству в использовании. Spring Boot позволяет быстро создавать готовые к работе приложения, не требуя сложной настройки. Он включает в себя множество встроенных функций и инструментов, таких как встроенный веб-сервер, поддержка REST API, работа с базами данных и многое другое. Spring Boot также обеспечивает высокую производительность и масштабируемость приложений.
3. **База данных: PostgreSQL**
   * **Обоснование:** PostgreSQL является мощной и надежной реляционной базой данных с открытым исходным кодом. PostgreSQL обеспечивает высокую производительность, надежность и поддерживает широкий спектр функций, включая транзакции, сложные запросы и различные типы данных. Она хорошо интегрируется с Java и Spring Boot, что упрощает работу с данными.
4. **Инструмент для работы с Excel: Apache POI**
   * **Обоснование:** Apache POI — это мощная Java библиотека для работы с файлами Microsoft Office, включая Excel. Эта библиотека предоставляет возможность чтения и записи файлов Excel, что необходимо для реализации функции загрузки данных о шинах и складах. Apache POI обеспечивает необходимую функциональность и является проверенным и надежным решением.
5. **Шаблонизатор: Thymeleaf**
   * **Обоснование:** Thymeleaf – это современный серверный Java шаблонизатор, позволяющий создавать HTML-страницы с динамическим контентом. Он хорошо интегрируется со Spring Boot и обеспечивает удобный способ работы с представлениями. Thymeleaf позволяет создавать более читаемый и поддерживаемый код, в отличие от JSP, а также обладает возможностью работы в offline режиме.
6. **Система контроля версий: Git**
   * **Обоснование:** Git — это популярная и широко распространенная система контроля версий. Git позволяет эффективно управлять изменениями в коде, обеспечивая возможность отслеживать историю изменений, координировать работу нескольких разработчиков и легко откатываться к предыдущим версиям в случае необходимости. Git является важным инструментом для командной разработки и обеспечения стабильности проекта.
7. **Среда разработки: IntelliJ IDEA**
   * **Обоснование:** IntelliJ IDEA - это мощная интегрированная среда разработки для Java, которая предоставляет широкий спектр инструментов для эффективной разработки приложений. Она включает в себя редактор кода, отладчик, инструменты для рефакторинга, поддержку систем контроля версий и многое другое. IntelliJ IDEA является популярным выбором среди Java-разработчиков, что делает ее идеальной для данного проекта.

## **Глава 2. Проектная стадия**

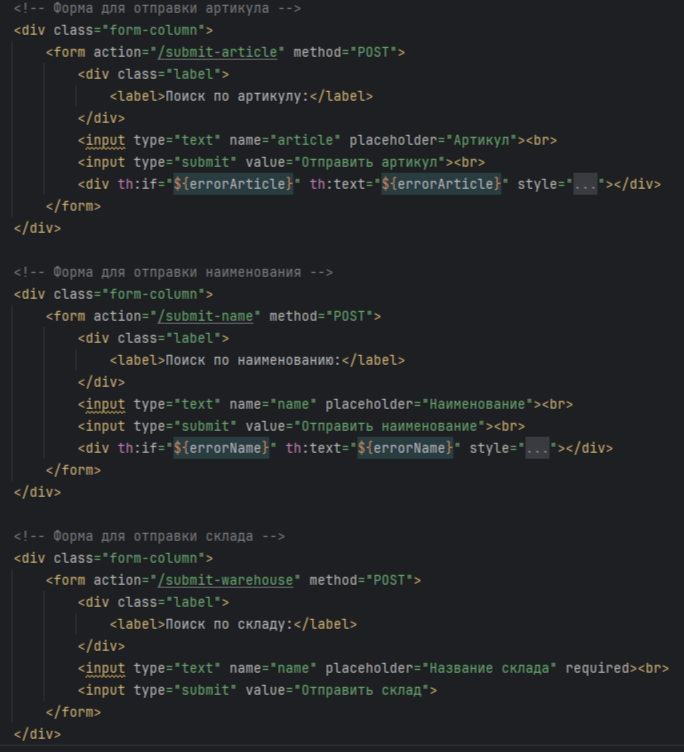
## **2.1 Проектирование общей архитектуры приложения**

Данный раздел посвящен проектированию общей архитектуры веб-приложения для управления складом шин. Архитектура разделена на три основных уровня, что обеспечивает модульность, масштабируемость и поддерживаемость приложения.

**2.1.1. Уровень Представления (Presentation)**

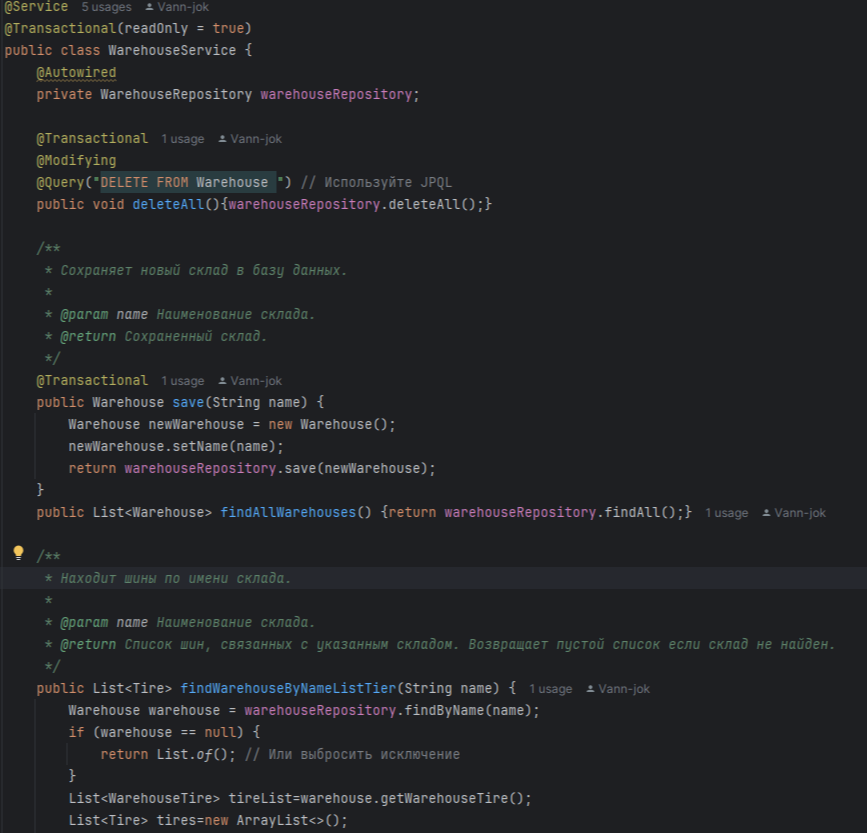
Уровень Представления отвечает за взаимодействие с пользователем. Он включает в себя:

* **Веб-интерфейс:** Создан с использованием Thymeleaf, языка шаблонов, обеспечивающего динамическое отображение данных из базы. Веб-интерфейс предоставляет пользователю интуитивно понятные формы для ввода данных, отображения списка шин и складов, а также для поиска необходимой информации. Используются стандартные HTML-элементы для создания понятных и удобных страниц. Макеты интерфейса, отображенные в приложениях, согласованы с UX-правилами. В основе интерфейса лежит понятная структура навигации.



**2.1.2. Уровень Сервиса (Application)**

Уровень Сервиса отвечает за бизнес-логику приложения. Этот уровень отделен от уровня представления и модели, что повышает гибкость и масштабируемость. Этот уровень содержит:

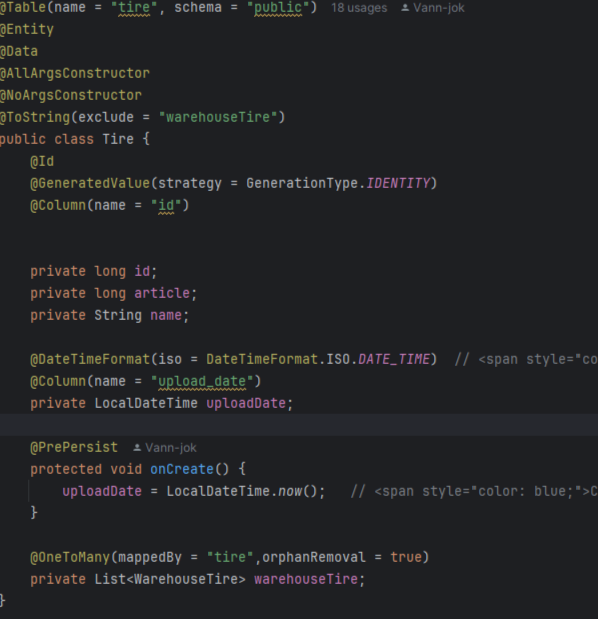
* **Сервисы для работы с данными:** Реализованы сервисные классы для работы со складами (WarehouseService), шинами (TireService) и связью шин со складами (WarehouseTireService). Сервисы обеспечивают все необходимые операции: получение списка, добавление, удаление, поиск данных, а также работу с загруженными данными из Excel. Сервисы реализованы с учетом принципов SOLID и DI (Dependency Injection), что гарантирует гибкость и модульность.
* **Валидация данных:** Сервисы включают валидацию входных данных, предотвращая некорректные операции и обеспечивая целостность данных в базе. Валидация происходит на уровне сервиса, отделяя логику проверки от представления.
* **Логика аутентификации:** Этот уровень содержит сервис (UserService) для аутентификации пользователей. Валидация введенных логина и пароля производится в сервис-слое, скрывая детали реализации от уровня представления.



**2.1.3. Уровень Модели (Data)**

Уровень Модели отвечает за хранение данных и взаимодействие с базой данных.

* **База данных:** Используется PostgreSQL для хранения данных о шинах, складах и пользователях.
* **Репозитории:** Используются репозитории Spring Data JPA для работы с базой данных. Репозитории обеспечивают абстракцию взаимодействия с базой данных, используя JPQL (Java Persistence Query Language).
* **Сущности (Entities):** Определены сущности (Tire, Warehouse, WarehouseTire, User) с необходимыми полями и связями, для представления данных в базе данных. Сущности соответствуют требованиям к модели данных, обеспечивая целостность данных.



**2.1.4. Взаимодействие уровней:**

Уровни взаимодействуют следующим образом:

* Уровень Представления получает запросы от пользователя и передает их в уровень Сервиса.
* Уровень Сервиса выполняет бизнес-логику и взаимодействует с уровнем Модели для получения или сохранения данных.
* Уровень Модели взаимодействует с базой данных для получения или сохранения данных.
* Результаты операций возвращаются обратно в уровень Представления для отображения пользователю.

**2.1.5. Технические решения:**

* **Spring Boot:** Выбран в качестве основной технологии для построения приложения.
* **Thymeleaf:** Используется для создания шаблонов веб-интерфейса.
* **PostgreSQL:** Используется для хранения данных.
* **Apache POI:** Используется для работы с Excel-файлами.

Эта архитектура обеспечивает модульность, масштабируемость, надежность и поддерживаемость, что позволит легко добавлять новые функции и масштабировать приложение по мере развития потребностей пользователя. Подробное описание каждой сущности и сервиса приведено в соответствующих разделах.

## **2.2. Проектирование WEB-интерфейса приложения**

Данный раздел описывает проектирование веб-интерфейса приложения для управления складом шин. Он фокусируется на удобстве использования, навигации и отображении данных пользователю.

**2.2.1. Общая структура веб-интерфейса:**

Веб-интерфейс разделен на несколько основных страниц, каждая из которых отвечает за определенный функционал:

* **Главная страница (Dashboard):** Содержит сводную информацию о складах и шинах: количество единиц на складах, наличие дефицита, общую информацию по шинам и т.д. Представлена в виде удобных таблиц и диаграмм.
* **Страница управления шинами:** Позволяет просматривать, добавлять, редактировать и удалять информацию о шинах. Используется табличная форма для представления списка шин, обеспечивающая сортировку, фильтрацию и пагинацию.
* **Страница управления складами:** Предоставляет возможности управления данными о складах. Позволяет просматривать, добавлять, редактировать и удалять информацию о складах.
* **Страница загрузки данных:** Позволяет загрузить данные о шинах и складах из файла Excel. Формы должны быть понятными и интуитивными, с учетом требований к вводу данных.
* **Страница поиска:** Позволяет искать информацию о шинах и складах по заданным критериям (артикул, наименование, склад). Результаты поиска отображаются в таблице, позволяющей фильтровать и сортировать найденные записи.
* **Страница авторизации:** Для доступа к административной части приложения. Форма авторизации должна быть простой и интуитивной. При ошибочной попытке авторизации должна отображаться соответствующая ошибка.

**2.2.2. Компоненты веб-интерфейса:**

* **Таблицы:** Основной механизм для отображения списков шин, складов и результатов поиска. Таблицы должны быть интуитивно понятными, поддерживать сортировку, фильтрацию, пагинацию и возможность поиска.
* **Формы:** Для ввода, редактирования и добавления данных о шинах и складах. Формы должны быть валидированы для предотвращения некорректного ввода.
* **Диаграммы (опционально):** Для наглядного представления статистических данных, например, количества шин на складах, дефицита и т.д.
* **Ссылки и кнопки:** Для перехода между страницами и выполнения действий (например, загрузка, сохранение, поиск).
* **Индикаторы загрузки (опционально):** Для отображения процесса загрузки данных.
* **Сообщения об ошибках:** Для информативного отображения ошибок, возникших при выполнении операций (например, некорректные данные, ошибки доступа).

**2.2.3. Требования к навигации:**

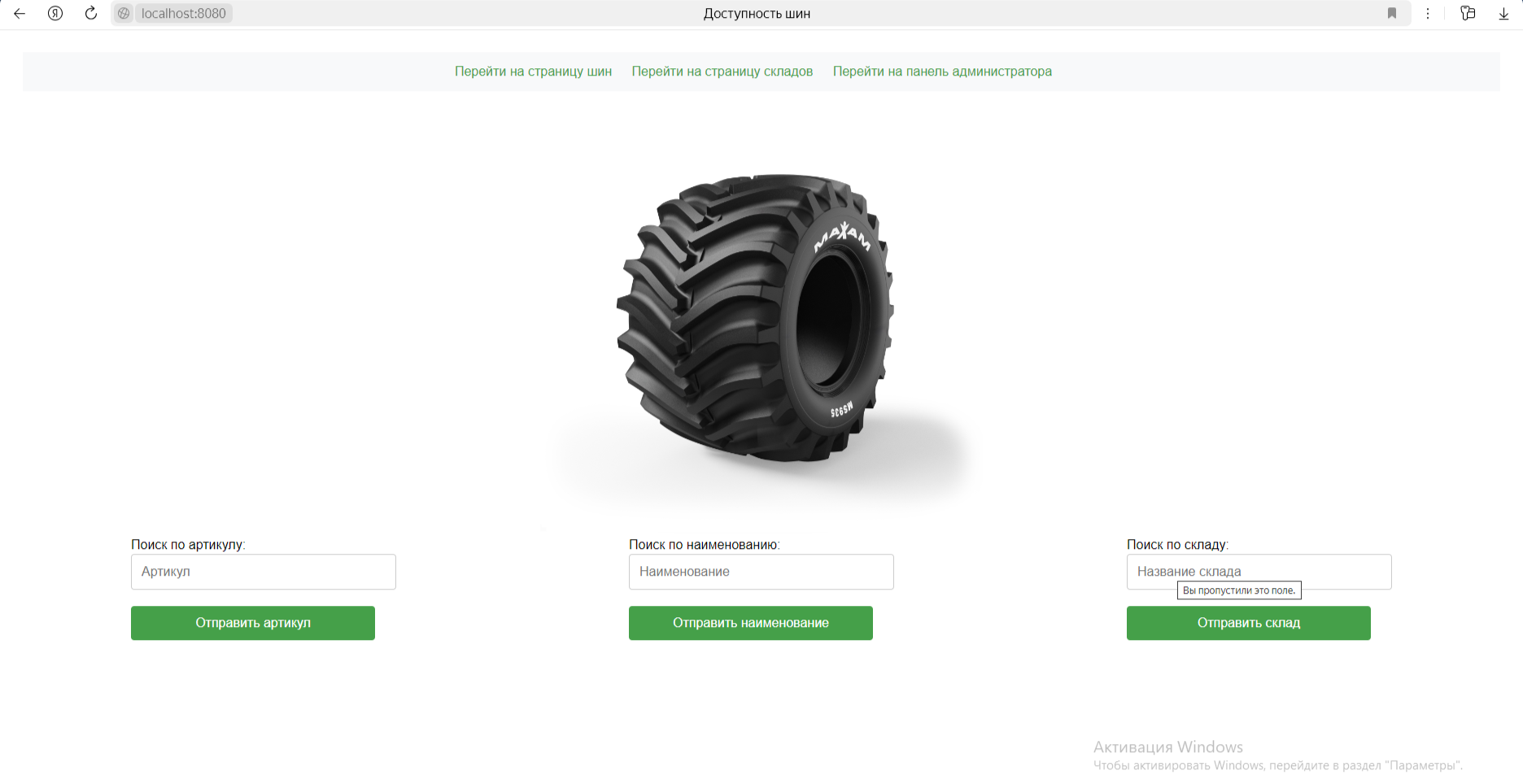
* Четкая и понятная навигация между страницами приложения.
* Использование интуитивно понятных ссылок и кнопок.
* Возможность быстрого возвращения к предыдущей странице.
* Отображение текущего местоположения пользователя в системе.

**2.2.4. Дополнительные требования:**

* **Валидация ввода:** Все поля в формах должны быть валидированы для предотвращения некорректных данных. Например, проверка на наличие обязательных значений, правильный формат данных и т.д.
* **Интуитивная структура:** Структура приложения должна быть интуитивной и простой для использования.
* **Доступность:** Веб-интерфейс должен быть доступен для разных категорий пользователей, включая пользователей с ограниченными возможностями.
* **Респонсивный дизайн:** Веб-интерфейс должен адаптироваться к различным устройствам (компьютеры, планшеты, мобильные телефоны).

**Заключение:**

Веб-интерфейс приложения будет разработан таким образом, чтобы обеспечить удобство, интуитивность и лёгкость использования для пользователей. Применение современных UX-практик обеспечит качественный интерфейс, удовлетворяющий всем требованиям по интуитивности, эффективности и безопасности. Подробное описание компонентов и функций интерфейса приведено в следующих разделах проекта.



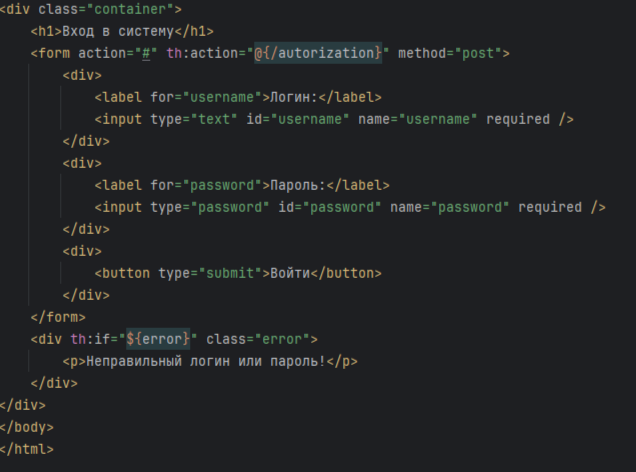
## **2.2. Проектирование WEB-интерфейса приложения (Расширенная версия)**

В данном разделе мы подробно рассмотрим проектирование веб-интерфейса приложения, уделив внимание структуре страниц, элементам управления и логике взаимодействия пользователя с системой. Особое внимание уделено принципам удобства использования (Usability), доступности и адаптивности.

**2.2.1. Архитектура Страниц**

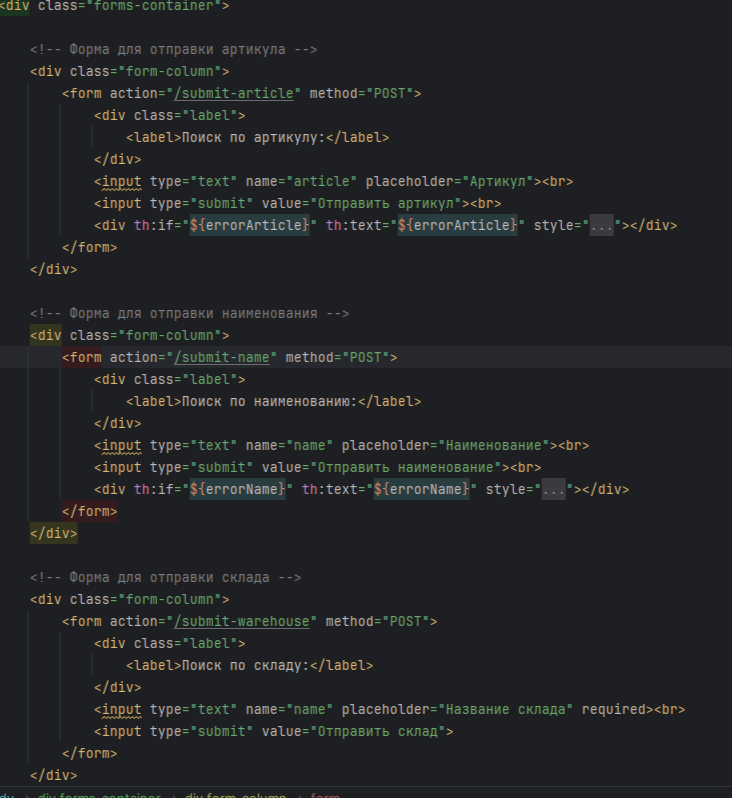
Веб-интерфейс спроектирован как SPA (Single Page Application) с использованием серверного рендеринга Thymeleaf. Это означает, что основная логика приложения обрабатывается на сервере, а страницы обновляются динамически при переходах пользователя. В соответствии с функциональными требованиями, предусмотрены следующие страницы:

1. **Страница авторизации (autorization.html)**:
   * **Назначение**: Предоставляет форму для ввода имени пользователя и пароля.
   * **Структура**:
     + Заголовок страницы “Авторизация”.
     + Форма (<form>) с полями для ввода логина и пароля (<input type="text"> и <input type="password">).
     + Кнопка “Войти” (<button type="submit">).
     + Сообщение об ошибке при неверном вводе.



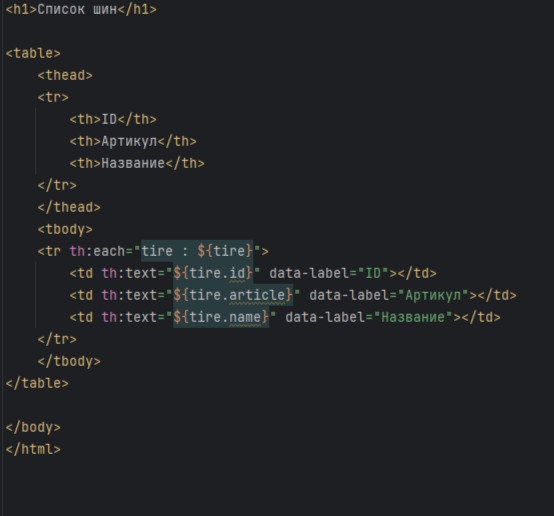
* + **Элементы**:
    - Текстовое поле для ввода логина.
    - Текстовое поле для ввода пароля.
    - Кнопка для отправки формы авторизации.

1. **Главная страница (home) - (home.html)**:
   * **Назначение**: Отображает общую информацию о складах и шинах, а также предоставляет навигацию к другим разделам.
   * **Структура**:
     + Заголовок страницы “Главная”.
     + Сводка по всем складам (количество, наличие дефицита и т.д.).
     + Сводка по всем шинам (количество, общее число)



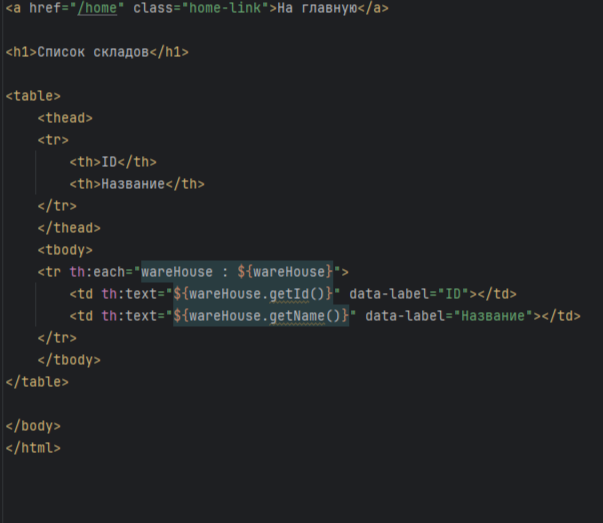
* + - Навигационное меню для доступа к страницам управления шинами, складами, загрузки данных и поиска.
  + **Элементы**:
    - Заголовки разделов.
    - Короткая информация по складам.
    - Короткая информация по шинам.
    - Навигационная панель.

1. **Страница управления шинами (tire-spisok.html)**:



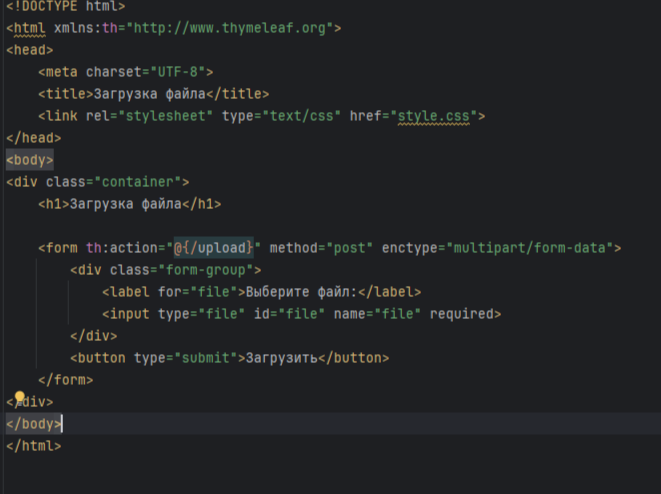
* + **Назначение**: Позволяет просматривать, добавлять, редактировать и удалять информацию о шинах.
  + **Структура**:
    - Заголовок страницы “Управление шинами”.
    - Таблица (<table>) со списком шин.
    - Форма для добавления/редактирования шин.
    - Кнопки “Добавить”, “Редактировать”, “Удалить”.
    - Пагинация для разделения списка на несколько страниц.
  + **Элементы**:
    - Заголовки столбцов таблицы.
    - Ячейки таблицы с информацией о шинах.
    - Поля ввода для добавления/редактирования шин.
    - Кнопки управления данными.

1. **Страница управления складами (warehouse-spisok.html)**:



* + **Назначение**: Позволяет просматривать, добавлять, редактировать и удалять информацию о складах.
  + **Структура**:
    - Заголовок страницы “Управление складами”.
    - Таблица (<table>) со списком складов.
    - Форма для добавления/редактирования складов.
  + **Элементы**:
    - Заголовки столбцов таблицы.
    - Ячейки таблицы с информацией о складах.
    - Поля ввода для добавления/редактирования складов.
    - Кнопки управления данными.

1. **Страница загрузки данных (upload-tire-list.html)**:



* + **Назначение**: Позволяет загрузить данные о шинах и складах из файла Excel.
  + **Структура**:
    - Заголовок страницы “Загрузка данных”.
    - Форма (<form>) с полем для выбора файла (<input type="file">).
    - Кнопка “Загрузить” (<button type="submit">).
    - Информация о поддерживаемых форматах.
    - Индикатор загрузки.
    - Сообщение об успешной загрузке или ошибке.
  + **Элементы**:
    - Поле для выбора файла.
    - Кнопка для отправки файла.
    - Информация об успешной загрузке.

**2.2.2. Элементы управления:**

* **Текстовые поля ввода (<input type="text">)**: Используются для ввода текста (например, наименование, артикул).
* **Поля ввода пароля (<input type="password">)**: Используются для ввода пароля.
* **Числовые поля ввода (<input type="number">)**: Используются для ввода чисел (например, количество).
* **Поля для загрузки файлов (<input type="file">)**: Используются для выбора файла для загрузки.
* **Кнопки (<button type="submit"> или <button>)**: Используются для отправки форм, выполнения действий, переходов по страницам и т.д.
* **Таблицы (<table>)**: Используются для отображения данных в структурированном виде.
* **Ссылки (<a>)**: Используются для навигации между страницами.
* **Сообщения (<span>, <div>, и т.д.)**: Используются для отображения информационных сообщений, ошибок и т.д.

**2.2.3. Логика взаимодействия пользователя:**

1. **Авторизация:**
   * Пользователь переходит на страницу авторизации (autorization.html).
   * Вводит имя пользователя и пароль в форму.
   * Нажимает кнопку “Войти”.
   * Система проверяет учетные данные и, в случае успеха, перенаправляет пользователя на главную страницу (dashboard.html).
   * В случае неудачи, на странице авторизации появляется сообщение об ошибке.
2. **Управление шинами/складами:**
   * Пользователь переходит на страницу управления шинами или складами.
   * Пользователь просматривает список существующих шин или складов.
   * Пользователь может добавить новую шину/склад, заполнив форму, либо редактировать существующую запись, нажав на кнопку “Редактировать”.
   * Для удаления записи, пользователь выбирает запись и нажимает “Удалить”.
   * После выполнения операции, пользователь получает подтверждающее сообщение.
3. **Загрузка данных:**
   * Пользователь переходит на страницу загрузки данных (upload-tire-list.html).
   * Выбирает файл Excel с данными.
   * Нажимает кнопку “Загрузить”.
   * Система обрабатывает файл и сохраняет данные в базе данных, отображая сообщение об успехе или ошибке.
4. **Поиск:**
   * Пользователь переходит на страницу поиска (search.html).
   * Вводит критерии поиска (артикул, наименование, склад).
   * Нажимает кнопку “Найти”.
   * Система отображает результаты поиска в таблице.

**2.2.4. Адаптивность и доступность:**

* Веб-интерфейс должен быть адаптивным, то есть корректно отображаться на разных устройствах (компьютеры, планшеты, мобильные телефоны) с использованием CSS и медиазапросов.
* Все текстовые элементы должны иметь достаточный контраст для людей с нарушением зрения.
* Интерфейс должен быть доступен для навигации с помощью клавиатуры.

**Заключение:**

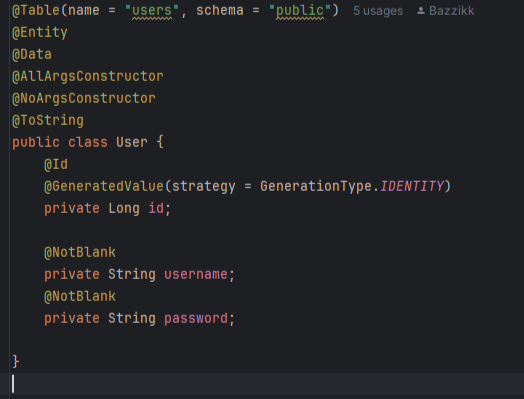
Этот раздел детализирует структуру и функциональность веб-интерфейса, подчеркивая принципы удобства использования, понятности, доступности и адаптивности.

## **2.3. Проектирование базы данных**

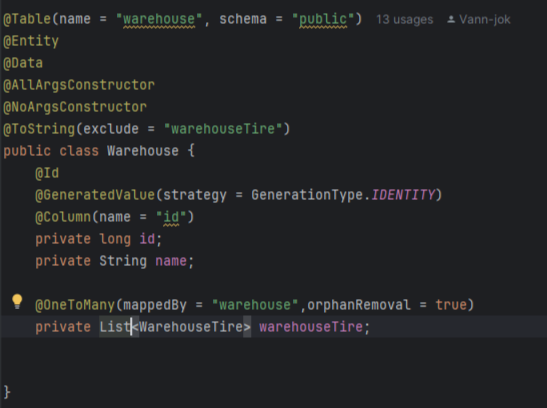
В данном разделе описывается проектирование базы данных для веб-приложения по управлению складом шин. База данных спроектирована с учетом требований к хранению информации о шинах, складах, пользователях и их взаимодействиях, а также с учетом принципов нормализации и производительности.

**2.3.1. Таблицы базы данных:**

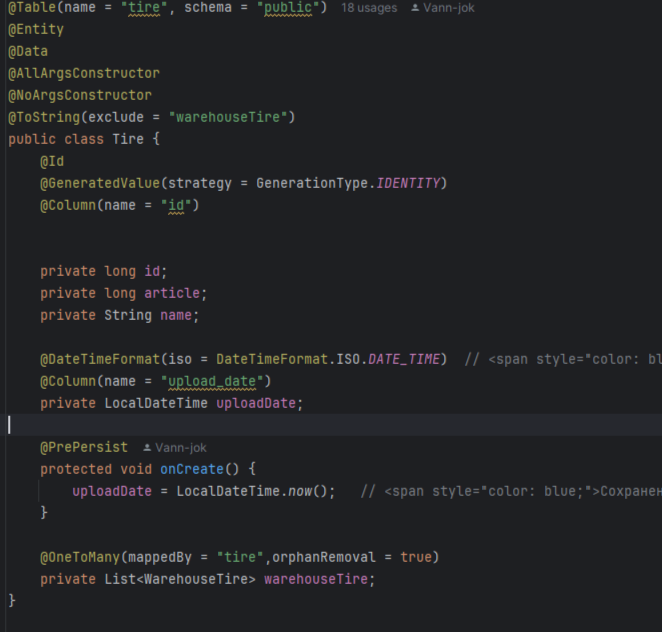
* **user:**



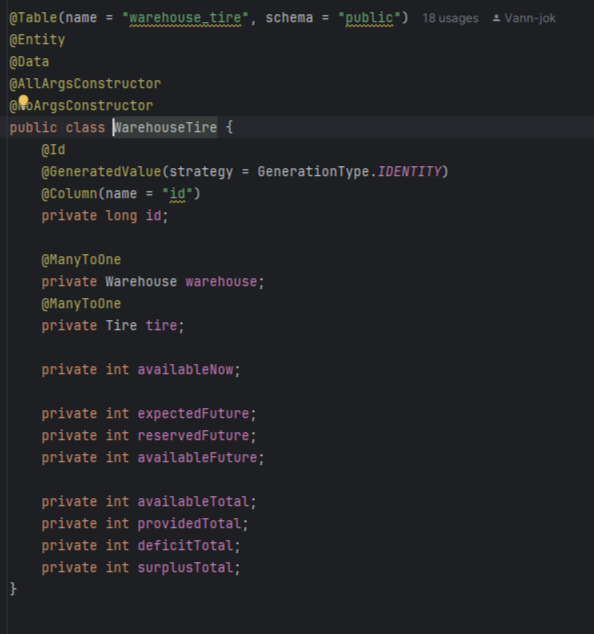
* + id (INT, Primary Key, AUTO\_INCREMENT): Идентификатор пользователя.
  + username (VARCHAR, уникальный): Имя пользователя.
  + password (VARCHAR): Хешированный пароль пользователя (не хранить пароль в открытом виде!).
* **warehouses:**



* + id (INT, Primary Key, AUTO\_INCREMENT): Идентификатор склада.
  + name (VARCHAR): Наименование склада.
  + address (VARCHAR): Адрес склада.
  + WarehouseTire: список колёс.
* **tire:**



* + id (INT, Primary Key, AUTO\_INCREMENT): Идентификатор шины.
  + article (BIGINT, уникальный): Артикул шины.
  + name (VARCHAR): Наименование шины.
  + uploadDate (DATA):Дата.
  + warehouseTire : список складов.
* **warehouse\_tires:**



* + id (INT, Primary Key, AUTO\_INCREMENT): Идентификатор записи о наличии шины на складе.
  + available\_now (INT): Количество шин на складе в текущий момент.
  + expected\_future (INT): Ожидаемое количество шин на складе в будущем.
  + reserved\_future (INT): Зарезервированное количество шин.
  + available\_total (INT): Общее доступное количество шин.
  + provided\_total (INT): Поставленное количество шин.
  + deficit\_total (INT): Общий дефицит шин.
  + surplus\_total (INT): Общий избыток шин.

**2.3.2. Связи между таблицами:**

* warehouses - warehouse\_tires (один ко многим): Связывает склады с наличием шин на них.
* tires - warehouse\_tires (один ко многим): Связывает шины с их наличием на складах.

**2.3.3. Типы данных:**

* Используются подходящие типы данных для каждого поля (INT, BIGINT, VARCHAR, DATE, POINT). Выбор типов данных основывается на анализе ожидаемых данных.

**2.3.4. Ограничения целостности:**

* Уникальные ключи (UNIQUE) для артикулов шин и имён пользователей.
* Внешние ключи (FOREIGN KEY) для обеспечения связей между таблицами.
* Ограничения для полей, чтобы обеспечить целостность и корректность данных. Например, available\_now не может быть отрицательным.

**Заключение:**

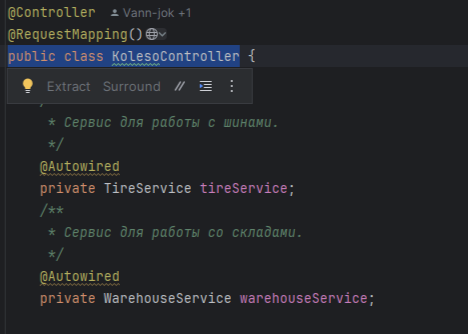
Проектирование базы данных является критически важным этапом разработки приложения. Выбор типов данных, связей и ограничений целостности позволит обеспечить надежность и производительность хранилища информации.

**2.4 Проектирование контроллеров**

**2.4.1 Подробное описание работы public class KolesoController**

KolesoController является контроллером Spring MVC, предназначенным для обработки HTTP-запросов, связанных с отображением списков шин и складов, домашней страницы и реализации функциональности поиска. Этот контроллер выступает посредником между пользовательским интерфейсом и бизнес-логикой приложения, обеспечивая взаимодействие между слоем представления и слоем сервисов.

* 1. **Общая структура и назначение**



* **Аннотация @Controller**: Обозначает класс как контроллер Spring MVC, ответственный за обработку HTTP-запросов.
* **Зависимости (TireService, WarehouseService)**: Внедряются через конструктор для доступа к методам бизнес-логики и аутентификации.

**2. Поля класса**

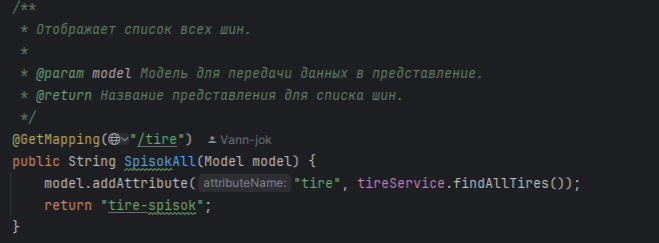
* **private final TireService tireService;**: Сервис для работы с шинами, обеспечивает доступ к методам получения информации о шинах.
* **private final WarehouseService warehouseService;**: Сервис для работы со складами, обеспечивает доступ к методам получения информации о складах.

**3. Конструктор**

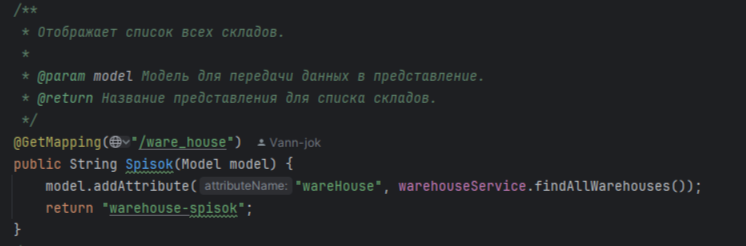
* **public KolesoController(TireService tireService, WarehouseService warehouseService, UserService userService)**:
  + Используется для внедрения зависимостей. Spring автоматически внедряет объекты TireService, WarehouseService и UserService в качестве параметров конструктора. Это позволяет KolesoController использовать их методы.

**4. Методы контроллера**

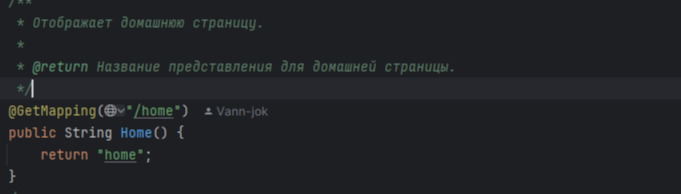
* **@GetMapping("/tire") public String SpisokAll(Model model)**:



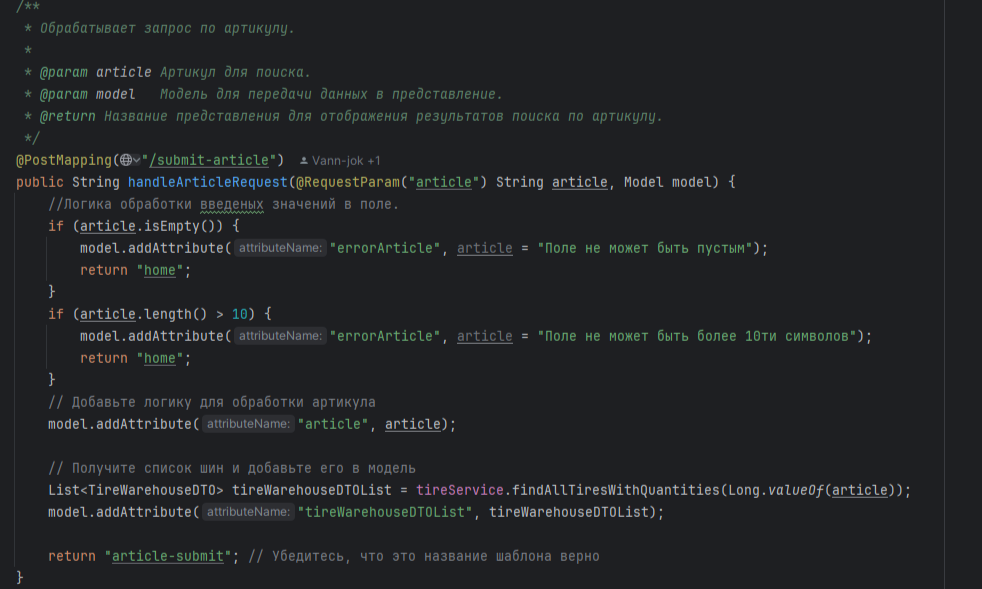
* + **@GetMapping("/tire")**: Аннотация для обработки GET-запросов на /koleso/tire.
  + **Model model**: Параметр для передачи данных в представление.
  + **Логика:**
    1. Вызывает tireService.findAllTires() для получения списка всех шин.
    2. Добавляет список шин в модель с именем "tire" (чтобы использовать его в представлении).
    3. Возвращает имя представления "tire-spisok", которое отобразит список шин.
  + **Предназначение:** Отображает страницу со списком всех шин.



* **@GetMapping("/ware\_house") public String Spisok(Model model)**:
  + **@GetMapping("/ware\_house")**: Аннотация для обработки GET-запросов на /koleso/ware\_house.
  + **Model model**: Параметр для передачи данных в представление.
  + **Логика:**
    1. Вызывает warehouseService.findAllWarehouses() для получения списка всех складов.
    2. Добавляет список складов в модель с именем "wareHouse".
    3. Возвращает имя представления "warehouse-spisok", которое отобразит список складов.



* **@GetMapping("/home") public String Home()**:
  + **@GetMapping("/home")**: Аннотация для обработки GET-запросов на /koleso/home.
  + **Логика:**
    1. Возвращает имя представления "home" для отображения домашней страницы.
  + **Предназначение:** Отображает домашнюю страницу приложения.



* **@PostMapping("/submit-article") public ModelAndView handleArticleRequest(@RequestParam("article") String article, Model model, HttpServletRequest request)**:
  + **@PostMapping("/submit-article")**: Аннотация для обработки POST-запросов на /koleso/submit-article.
  + **@RequestParam("article") String article**: Параметр для получения артикула из формы.
  + **Model model**: Параметр для передачи данных в представление.
  + **HttpServletRequest request**: Используется для проверки аутентификации пользователя.
  + **Логика:**
    1. **Валидация ввода:**
       - Проверяет, не пустое ли поле ввода (article.isEmpty()). Если да, добавляет ошибку в модель и возвращает домашнюю страницу.
       - Проверяет, не превышает ли длина введенного значения 10 символов (article.length() > 10). Если да, добавляет ошибку в модель и возвращает домашнюю страницу.
    2. **Проверка аутентификации:**
       - Вызывает userService.isUserLoggedIn(request) для проверки статуса входа пользователя.
       - Если пользователь не авторизован, добавляет сообщение об ошибке в модель и перенаправляет на страницу авторизации (redirect:/koleso/authorization).
    3. **Поиск шин:**
       - Вызывает метод tireService.findAllTiresWithQuantities(Long.valueOf(article)) для поиска шин по артикулу (преобразование строки в Long).
    4. Добавляет результаты поиска в модель с именем "tireWarehouseDTOList".
    5. Добавляет искомый артикул в модель с именем "article".
    6. Возвращает ModelAndView с именем представления "article-submit".
  + **Предназначение:** Обрабатывает запрос поиска шин по артикулу, проверяет авторизацию пользователя.
* **@PostMapping("/submit-name") public String handleNameRequest(@RequestParam("name") String name, Model model, HttpServletRequest request)**:
  + **@PostMapping("/submit-name")**: Аннотация для обработки POST-запросов на /koleso/submit-name.
  + **@RequestParam("name") String name**: Параметр для получения наименования из формы.
  + **Model model**: Параметр для передачи данных в представление. \* **HttpServletRequest request**: Используется для проверки аутентификации пользователя.
  + **Логика:**
    1. **Валидация ввода:**
       - Проверяет, не пустое ли поле ввода (name.isEmpty()). Если да, добавляет ошибку в модель и возвращает домашнюю страницу.
       - Проверяет, не превышает ли длина введенного значения 12 символов (name.length() > 12). Если да, добавляет ошибку в модель и возвращает домашнюю страницу.
    2. **Проверка аутентификации:**
       - Вызывает userService.isUserLoggedIn(request) для проверки статуса входа пользователя.
       - Если пользователь не авторизован, добавляет сообщение об ошибке в модель и перенаправляет на страницу авторизации (redirect:/koleso/authorization).
    3. **Поиск шин:**
       - Вызывает метод tireService.findAllTiresWithName(name) для поиска шин по наименованию.
       - Добавляет результаты поиска в модель с именем "tireWarehouseDTOList".
    4. Добавляет искомое наименование в модель с именем "name".
    5. Возвращает имя представления "article-submit".
  + **Предназначение:** Обрабатывает запрос поиска шин по наименованию, проверяет авторизацию пользователя.
* **@PostMapping("/submit-warehouse") public String handleWarehouseRequest(@RequestParam("name") String warehouse, Model model, HttpServletRequest request)**:
  + **@PostMapping("/submit-warehouse")**: Аннотация для обработки POST-запросов на /koleso/submit-warehouse.
  + **@RequestParam("name") String warehouse**: Параметр для получения наименования склада из формы.
  + **Model model**: Параметр для передачи данных в представление.
  + **HttpServletRequest request**: Используется для проверки аутентификации пользователя.
  + **Логика:**
    1. **Проверка аутентификации:**
       - Вызывает userService.isUserLoggedIn(request) для проверки статуса входа пользователя.
       - Если пользователь не авторизован, добавляет сообщение об ошибке в модель и перенаправляет на страницу авторизации (redirect:/koleso/authorization).
    2. **Поиск шин:**
       - Вызывает warehouseService.findWarehouseByNameListTier(warehouse) для поиска шин на складе по его наименованию.
    - Добавляет результаты поиска в модель с именем "tire". 3. Добавляет искомое название склада в модель с именем "warehouse". 4. Возвращает имя представления "warehouse-submit".
  + **Предназначение:** Обрабатывает запрос поиска шин по складу, проверяет авторизацию пользователя.
* **@GetMapping("/authorization") public String showAuthorizationPage()**:
  + **@GetMapping("/authorization")**: Аннотация для обработки GET-запросов на /koleso/authorization
  + **Логика:**
    - Возвращает имя представления "authorization" (или login page), которое отобразит страницу авторизации.
  + **Предназначение:** Отображает страницу авторизации.

**5. Взаимодействие с другими компонентами**

* **TireService**: Используется для получения списков всех шин (findAllTires()) и поиска шин по артикулу (findAllTiresWithQuantities(Long article)) и наименованию (findAllTiresWithName(String name)).
* **WarehouseService**: Используется для получения списка всех складов (findAllWarehouses()) и поиска шин по складу (findWarehouseByNameListTier(String name)).
* **UserService**: Используется для проверки аутентификации пользователя isUserLoggedIn(HttpServletRequest request).
* **Представления (HTML-шаблоны)**: Контроллер использует HTML-шаблоны (например, tire-spisok.html, warehouse-spisok.html, home.html, article-submit.html и warehouse-submit.html) для отображения данных пользователю.
* **Model**: Используется для передачи данных (списков шин, складов, сообщений) в представления.
* **ModelAndView**: Используется для динамического возвращения представлений и перенаправлений (redirect:/koleso/authorization).
* **HttpServletRequest**: Используется для проверки аутентификации.

**6. Логика обработки запросов**

1. **GET-запросы (/tire, /ware\_house, /home, /authorization)**:
   * Контроллер вызывает соответствующие методы сервисов для получения данных.
   * Данные добавляются в модель.
   * Возвращается имя представления для отображения пользователю.
2. **POST-запросы (/submit-article, /submit-name, /submit-warehouse)**:
   * Контроллер получает параметры запроса из формы.
   * **Важно!** Контроллер проверяет авторизацию пользователя перед выполнением поиска.
   * Контроллер проводит валидацию полученных данных.
   * Контроллер вызывает соответствующие методы сервисов для поиска.
   * Результаты поиска добавляются в модель.
   * Возвращается ModelAndView с указанием представления и перенаправлением. **7. Особенности реализации**

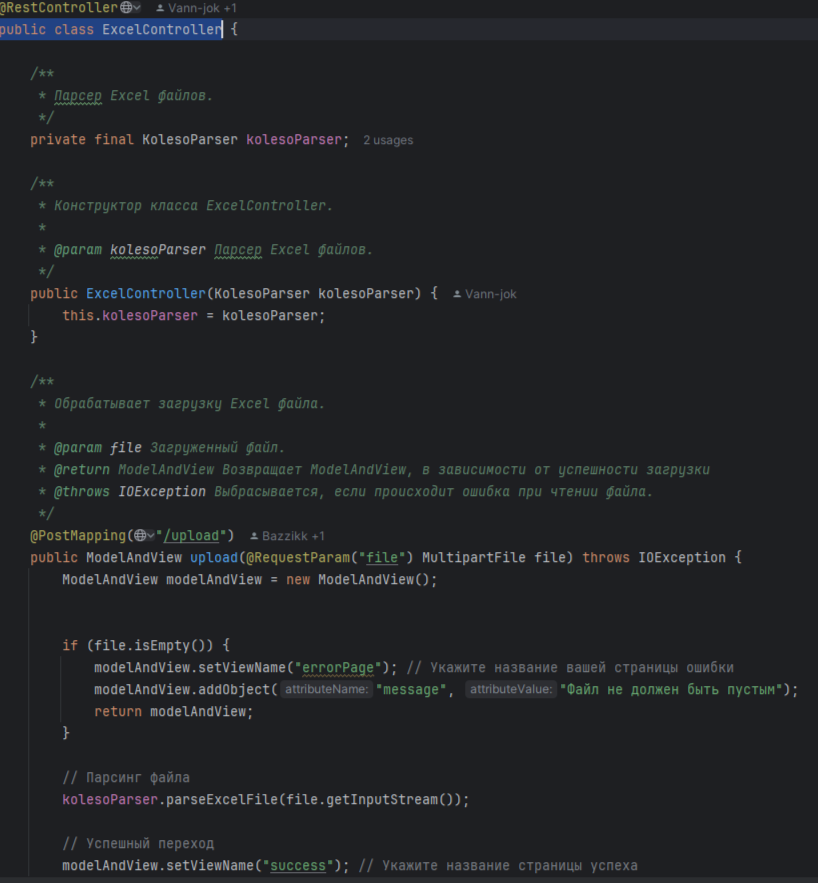
* **Валидация данных:** В handleArticleRequest и handleNameRequest реализована базовая валидация ввода (проверка на пустое значение и на длину значения).
* **Безопасность:** Внедрена проверка авторизации для доступа к страницам поиска, что обеспечивает защиту от несанкционированного доступа к критической функциональности.
* **Использование ModelAndView**: Для динамического перенаправления неавторизованных пользователей, что позволяет улучшить user experience и безопасность приложения.

**Заключение**

KolesoController является ключевым компонентом веб-приложения, обеспечивая обработку HTTP-запросов и взаимодействие с бизнес-логикой для отображения данных пользователю. Контроллер обеспечивает базовую функциональность и необходимый уровень безопасности. Подробное понимание работы этого контроллера позволяет глубже понять архитектуру приложения и его функциональность.

**2.4.2. Подробное описание работы public class ExcelController**

ExcelController представляет собой REST-контроллер в Spring MVC, предназначенный для обработки запросов на загрузку и обработку Excel-файлов. Он использует сервис KolesoParser для парсинга загруженных файлов и возвращает ModelAndView для отображения результатов. Этот контроллер обеспечивает функциональность импорта данных из Excel-файлов в ваше приложение.



**1. Общая структура и назначение**

* **Аннотация @RestController**: Обозначает класс как REST-контроллер Spring MVC. В отличие от @Controller, @RestController автоматически добавляет аннотацию @ResponseBody ко всем методам, что означает, что возвращаемые данные будут сериализованы в JSON (в нашем случае ModelAndView, который Spring MVC обрабатывает как обычный Controller, но при этом все еще является RestController).
* **Зависимость KolesoParser**: Внедряется через конструктор для доступа к методам парсинга Excel-файлов.

**2. Поля класса**

* **private final KolesoParser kolesoParser;**: Поле для хранения объекта KolesoParser, который внедряется через конструктор. final обозначает, что поле будет проинициализировано только один раз.

**3. Конструктор**

* **public ExcelController(KolesoParser kolesoParser)**:
  + Используется для внедрения зависимости KolesoParser. Spring автоматически внедряет объект KolesoParser в качестве параметра конструктора, что позволяет ExcelController использовать его методы.

**4. Метод контроллера**

* **@PostMapping("/upload") public ModelAndView upload(@RequestParam("file") MultipartFile file) throws IOException**:
  + **@PostMapping("/upload")**: Аннотация для обработки POST-запросов на /upload. Это означает, что метод upload будет вызываться, когда пользователь отправит POST-запрос на этот URL.
  + **@RequestParam("file") MultipartFile file**: Аннотация для получения загруженного файла в виде объекта MultipartFile. Spring автоматически конвертирует загруженный файл в объект MultipartFile.
  + **throws IOException**: Указывает, что метод может выбрасывать IOException (например, если возникнет ошибка при чтении файла), и эта ошибка будет обработана Spring MVC.
  + **ModelAndView modelAndView = new ModelAndView();**: Создание объекта ModelAndView, который используется для управления представлением и передачи данных в него.
  + **Логика:**
    1. **Проверка на пустой файл (if (file.isEmpty()))**:
       - Проверяет, был ли выбран файл для загрузки. Если файл не был выбран (пустой), то:
         * Устанавливает имя представления errorPage.
         * Добавляет сообщение об ошибке в модель с ключом “message” (сообщение, что файл не должен быть пустым).
         * Возвращает modelAndView, который Spring MVC использует для отображения страницы ошибки.
    2. **Парсинг файла (kolesoParser.parseExcelFile(file.getInputStream());)**:
       - Если файл не пустой, вызывает метод parseExcelFile сервиса KolesoParser, передавая поток ввода загруженного файла (file.getInputStream()). Это позволяет KolesoParser прочитать данные из Excel-файла и обработать их (например, сохранить в базу данных).
    3. **Успешное завершение (modelAndView.setViewName("success");)**:
       - Если парсинг выполнен успешно, то:
    4. \* Устанавливает имя представления `success`.
    5. \* Добавляет сообщение об успехе в модель с ключом `"message"`.
    6. **Возврат ModelAndView (return modelAndView;)**:
    7. Возвращает ModelAndView, который содержит информацию о том какое представление и с какими данными необходимо отобразить.

**5. Взаимодействие с другими компонентами**

* **KolesoParser**:
  + ExcelController делегирует логику обработки Excel-файла сервису KolesoParser.
  + KolesoParser отвечает за чтение данных из файла, их обработку и сохранение в базу данных.
* **Представления (HTML-шаблоны)**:
  + ExcelController использует представления success.html и errorPage.html (должны быть созданы), для отображения результата пользователю.
* **MultipartFile**:
  + Интерфейс Spring Framework для представления загруженного файла, позволяет получить доступ к байтам загруженного файла.
* **ModelAndView**:
  + Объект Spring MVC, который используется для управления представлением (шаблон) и передачи данных в это представление.

**6. Логика обработки запросов**

1. **Запрос на /upload**:
   * Пользователь загружает Excel-файл через форму и отправляет POST-запрос на /upload.
2. **Обработка ExcelController**:
   * ExcelController получает загруженный файл в виде MultipartFile.
   * Проверяет, был ли выбран файл для загрузки.
   * Если файл не был выбран, то возвращается ModelAndView, который отвечает за отображение страницы ошибки.
   * Если файл не пустой, вызывается метод kolesoParser.parseExcelFile(file.getInputStream()) для парсинга Excel-файла.
3. **Результат обработки**
   * Если парсинг был выполнен успешно, то возвращается ModelAndView, который отвечает за отображение страницы успеха.
   * В обоих случаях Spring MVC отвечает за отрисовку нужной страницы.

**7. Особенности реализации**

* **Обработка исключений**:
  + Метод upload() может выбрасывать IOException, что позволяет Spring MVC обрабатывать ошибки при чтении файла.
* **Разделение ответственности**:
  + ExcelController отвечает только за прием файла и делегирует парсинг KolesoParser. Это обеспечивает разделение ответственности и упрощает поддержку и расширение функциональности.
* **RESTful:**
  + Использование аннотации @RestController позволяет обрабатывать запросы в стиле REST, делая контроллер более простым и предсказуемым.
* **ModelAndView**
  + Позволяет гибко управлять отображением результатов, а также в зависимости от ситуации редиректить пользователей. **8. Дополнительные замечания**
* Для корректной работы ExcelController необходимо реализовать представления success.html и errorPage.html, а также KolesoParser для обработки файлов.
* Следует добавить логику обработки исключений в KolesoParser для предотвращения сбоев в работе приложения.
* Для больших файлов необходимо настроить таймаут и обработку ошибок для корректного импорта данных.

**Заключение**

ExcelController представляет собой REST-контроллер, который обеспечивает загрузку и обработку Excel-файлов в вашем приложении. Он делегирует парсинг KolesoParser и возвращает ModelAndView для отображения результатов пользователю. Этот контроллер является важным звеном в процессе импорта данных, обеспечивая гибкость, надежность и возможность дальнейшего расширения функциональности.

**2.4.3 Подробное описание работы public class UserController**

UserController в вашем приложении отвечает за обработку запросов, связанных с аутентификацией, авторизацией и доступом к защищенным ресурсам. Он является посредником между веб-интерфейсом (например, HTML-формами) и бизнес-логикой аутентификации (UserService).

**1. Общая структура и назначение**

UserController — это контроллер Spring MVC, который обрабатывает HTTP-запросы, связанные с авторизацией пользователя. Он принимает запросы на авторизацию, проверяет корректность введенных данных и, при успехе, предоставляет доступ к защищенным ресурсам. Важно, что контроллер отвечает за логику доступа, не за хранение данных самих пользователей.

**2. Поля класса**

* **private final UserService userService;**: Это поле хранит зависимость к сервису UserService, который содержит всю бизнес-логику, связанную с аутентификацией и авторизацией. Это ключевой компонент, обеспечивающий отделение логики доступа от контроллера.

**3. Конструктор**

* **public UserController(UserService userService)**: Используется для внедрения зависимости UserService. Spring автоматически создаст и внедрит экземпляр UserService в контроллер.

**4. Методы контроллера**

* **@GetMapping("/authorization") public String showAuthorizationForm(Model model)**:
  + **@GetMapping("/authorization")**: Обрабатывает GET-запросы на URL /authorization.
  + **Model model**: Используется для передачи данных в представление.
  + **Логика:**
    1. Возвращает представление authorization, которое, как ожидается, содержит форму для ввода логина и пароля. Важно: Это *только* отображает форму; логика проверки и обработки ввода находится в другом месте. Это делает контроллер *легче читаемым* и *тестируемым*.
  + **Предназначение:** Отображает форму для ввода учетных данных.
* **@PostMapping("/authorization") public ModelAndView authorizeUser(@RequestParam String username, @RequestParam String password, HttpServletRequest request, Model model)**:
  + **@PostMapping("/authorization")**: Обрабатывает POST-запросы на /authorization, обрабатывающие отправленную форму.
  + **@RequestParam String username, @RequestParam String password**: Получает имя пользователя и пароль из формы.
  + **HttpServletRequest request**: Используется для получения информации о сессии пользователя, для последующей работы.
  + **Model model**: Используется для передачи данных в представление.
  + **Логика:**
    1. Вызывает userService.authenticate(username, password) для проверки учетных данных.
    2. Если аутентификация прошла успешно, то:
       - Создает ModelAndView и задает представление redirect:/koleso/home. Это перенаправление на защищенную область (/koleso).
       - Добавляет пользователя в сессию (с помощью request). Это важный шаг для последующей авторизации.
    3. Если аутентификация не удалась, то:
       - Добавляет ошибку в модель.
       - Возвращает представление authorization для отображения ошибки пользователю.
  + **Предназначение:** Проверяет учетные данные пользователя и перенаправляет или отображает ошибку.
* **@GetMapping("/logout") public String logout(HttpServletRequest request)**:
  + **@GetMapping("/logout")**: Обрабатывает GET-запрос на /logout.
  + **HttpServletRequest request**: Используется для работы с сессией.
  + **Логика:**
    1. Выполняет логирование пользователя (удаление сессии или cookie).
    2. Возвращает представление redirect:/koleso/authorization для перенаправления на форму авторизации.
  + **Предназначение:** Выполняет процедуру выхода пользователя из системы.

**5. Взаимодействие с другими компонентами**

* **UserService**: Обрабатывает всю логику проверки учетных данных.
* **ModelAndView**: Используется для перенаправления пользователя на защищенные ресурсы или для отображения сообщения об ошибке.
* **Model**: Используется для передачи данных в представления (сообщения об ошибках, успешной аутентификации).

**6. Логика обработки запросов**

1. Запрос на /authorization (GET): Отображение формы авторизации.
2. Запрос на /authorization (POST): Обработка формы авторизации, проверка учетных данных и перенаправление или отображение ошибки.
3. Запрос на /logout: Выход пользователя из системы.

**7. Особенности реализации**

* **Разделение ответственности**: Контроллер фокусируется на обработке HTTP-запросов и управлении доступом, делегируя бизнес-логику UserService. Это улучшает читаемость и тестируемость кода.
* **Использование ModelAndView**: Обеспечивает гибкость, включая возможность перенаправлений после успешной или неудачной авторизации.
* **Обработка ошибок**: В методе authorizeUser реализована обработка случаев неудачной аутентификации (возвращается authorization с сообщением об ошибке).

**Заключение**

UserController играет важную роль в обеспечении безопасности приложения, обрабатывая запросы на аутентификацию и авторизацию. Отделение логики доступа от контроллера способствует модульности и простоте сопровождения проекта.

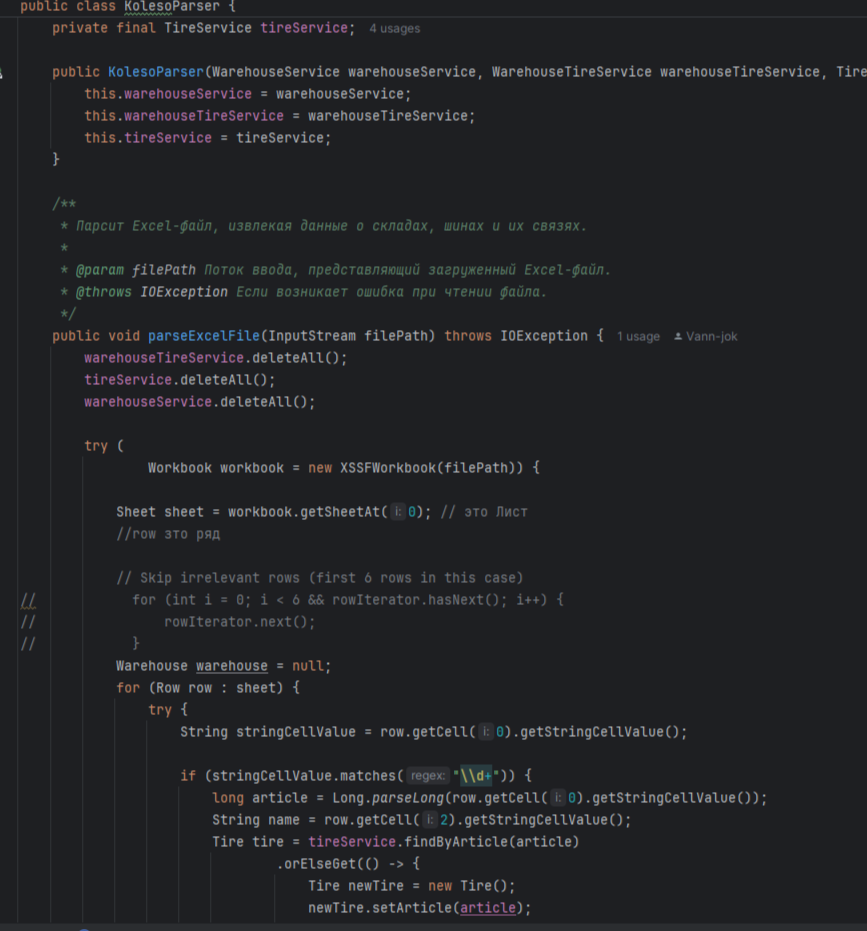
**2.5. Полное и подробное описание работы public class KolesoParser**

KolesoParser — это компонент Spring, предназначенный для парсинга Excel-файлов, содержащих информацию о шинах, складах и их взаимосвязях. Он отвечает за чтение данных из файла, их преобразование в сущности Java (объекты, представляющие данные в вашем приложении) и сохранение этих сущностей в базу данных с использованием соответствующих сервисов.

**1. Общая структура и назначение**

* **@Component**: Аннотация, которая обозначает класс как компонент Spring. Это позволяет Spring управлять жизненным циклом объекта KolesoParser и внедрять его зависимости в другие компоненты приложения.
* **Назначение:** KolesoParser является ключевым звеном в процессе импорта данных из Excel-файлов в ваше приложение. Он извлекает информацию о шинах, складах, их количествах и связях, преобразует ее в объекты Java и сохраняет в базу данных.
* **Зависимости:** KolesoParser зависит от нескольких сервисов (WarehouseService, WarehouseTireService, TireService), что позволяет ему делегировать операции с данными (поиск, сохранение, удаление) другим компонентам приложения, следуя принципам разделения ответственности.

1. **Поля класса**



* **private final WarehouseService warehouseService;**: Сервис, предназначенный для работы со сущностями Warehouse. Используется для сохранения и поиска складов.
* **private final WarehouseTireService warehouseTireService;**: Сервис, предназначенный для работы со связями между шинами и складами. Используется для сохранения информации о наличии шин на складах (сущность WarehouseTire).
* **private final TireService tireService;**: Сервис, предназначенный для работы с сущностями Tire. Используется для сохранения и поиска информации о шинах.
* **private Warehouse warehouse = null;**: Поле для сохранения текущего склада.

**3. Конструктор**

* **public KolesoParser(WarehouseService warehouseService, WarehouseTireService warehouseTireService, TireService tireService)**:
  + Используется для внедрения зависимостей. Spring автоматически предоставляет необходимые экземпляры сервисов, позволяя KolesoParser использовать их методы для доступа к данным.
  + Обеспечивает внедрение зависимостей через конструктор, что повышает тестируемость и гибкость кода.

**4. Методы класса**

* **public void parseExcelFile(InputStream filePath) throws IOException**:
  + **InputStream filePath**: Параметр, представляющий поток ввода Excel-файла.
  + **throws IOException**: Указывает на возможность возникновения исключения ввода/вывода.
  + **Логика:**
    1. **Очистка данных:**
       - Удаляет все существующие данные из таблиц warehouse\_tire, tire и warehouse (для новой загрузки).
    2. **Важно!**: В реальной системе может потребоваться другая логика (например, обновление существующих записей или отслеживание изменений).
    3. **Чтение Excel-файла:**
       - Использует Apache POI для создания объекта Workbook из входного потока.
       - Получает первый лист Excel-файла.
    4. **Итерация по строкам:**
       - Проходит каждую строку на листе.
       - Проверяет, является ли первая ячейка в строке числом (артикулом). Если является, то обрабатывает данные о шинах, в противном случае - обрабатывает данные о складе.
  1. **Обработка строк с данными о шинах:**
     + Извлекаются данные из ячеек: артикул (ячейка 0), наименование (ячейка 2), количество доступных шин (ячейка 6), ожидаемое поступление (ячейка 8), зарезервировано (ячейка 9), доступно в будущем (ячейка 10), доступно всего (ячейка 11), поступило всего (ячейка 12), дефицит всего (ячейка 13), избыток всего (ячейка 14). \* **Важно**: Если ячейка не содержит число, то для всех ячеек со значениями количества (ячейки 6, 8, 9, 10, 11, 12, 13, 14) используется значение по умолчанию 0 (перехват Exception). \* Выполняется поиск шины по артикулу (tireService.findByArticle(article)).
       - Если шина не найдена, то создается новая сущность Tire, сохраняется и используется в дальнейшем.
       - Если шина найдена, то используется эта сущность Tire.
         * Создается сущность WarehouseTire с данными о текущей шине и текущем складе, и заполняется информацией о количестве шин.
       - Сущность сохраняется в базу данных.
* 5. \*\*Обработка строк с данными о складе:\*\*
* \* Проверяется, начинается ли значение первой ячейки с `Склад ООО`. Если да, то это строка с данными о складе.
* \* Извлекается наименование склада из первой ячейки.
* \* Сохраняется склад с помощью `warehouseService.save(name)` и его сущность записывается в поле `private Warehouse warehouse = null;` для использования в следующих строках.

**5. Взаимодействие с другими компонентами**

* **WarehouseService**:
  + Используется для удаления всех складов перед загрузкой данных.
  + Используется для поиска или сохранения сущности Warehouse.
* **WarehouseTireService**:
  + Используется для удаления всех связей между шинами и складами перед загрузкой данных.
  + Используется для сохранения информации о наличии шин на складах в сущностях WarehouseTire.
* **TireService**:
  + Используется для удаления всех шин перед загрузкой данных.
  + Используется для поиска шин по артикулу и сохранения данных о шинах.
* **Apache POI**:
  + Используется для чтения данных из Excel-файлов.
* **InputStream**:
  + Используется для получения доступа к потоку ввода Excel-файла.
* **Row, Cell**: Интерфейсы Apache POI для представления строк и ячеек в Excel-файле.

**6. Логика работы**

1. parseExcelFile() принимает на вход поток данных Excel-файла.
2. Все существующие данные из связанных таблиц очищаются.
3. Создается Workbook из файла для доступа к его содержимому.
4. Происходит итерация по всем строкам первого листа Excel-файла.
5. Для каждой строки, где первая ячейка является числом (артикулом):
   * Извлекаются данные.
   * Ищется шина в базе данных, если не найдена - создается новая и сохраняется.
   * Создается и сохраняется сущность WarehouseTire со всеми полученными данными о наличии шин на складе.
6. Для каждой строки, где первая ячейка начинается с “Склад ООО”:
   * Извлекается название склада и сохраняется.

**7. Особенности реализации**

* **Обработка ошибок:**
  + Используются блоки try-catch для перехвата исключений при преобразовании данных в числовой формат, что позволяет избежать сбоев при парсинге невалидных данных, однако это *некорректная обработка*, так как исключение “проглатывается” (пустой блок catch).
  + Код перехватывает Exception, что не является лучшей практикой, лучше отлавливать конкретные типы исключений.
* **Значения по умолчанию:**
  + Если значение не может быть прочитано из ячейки (или не число), то используются значения по умолчанию (0).
* **Порядок обработки данных:**
  + Сначала удаляются все старые данные, а затем загружаются новые.
* **Работа с сервисами:**
  + KolesoParser делегирует операции с базой данных сервисам (WarehouseService, WarehouseTireService, TireService).

**8. Рекомендации по улучшению**

* **Логирование**: Добавить логирование для отслеживания процесса парсинга, ошибок и важных событий (например, создание или обновление сущностей).
* **Обработка ошибок**: Улучшить обработку ошибок в блоках try-catch. Рассмотреть возможность выбрасывания собственных исключений или логирования ошибок.
* **Управление обновлениями**: Реализовать логику для обновления данных в случае, если они уже существуют (например, с помощью поиска по артикулу и складу).
* **Рефакторинг**: Вынести повторяющийся код для обработки разных типов данных в отдельные методы.
* **Более гибкое определение данных о складе**: Следует рассмотреть возможность более гибкого определения строки со складом, а также возможность работы с более сложной структурой имени склада.
* **Детализация исключений:** Ловить более специфичные исключения, а не просто Exception.
* **Атомарность операций:** Все операции с базами данных должны быть атомарны, т.е. либо выполняются все, либо ни одна, в случае ошибки.
* **Валидация**: Проверять данные на валидность (например, отрицательные значения) прежде чем сохранять в базу данных.
* **Заключение**

KolesoParser является ключевым компонентом вашего приложения, обеспечивая чтение данных из Excel-файлов, их преобразование в объекты Java и сохранение в базу данных. Он делегирует операции с данными сервисам, следуя принципам разделения ответственности. Однако, код требует доработки в части обработки исключений, логирования и более гибкой обработки данных.

**Заключение**

В процессе работы над дипломным проектом был получен ценный опыт в разработке веб-приложений с использованием Java и фреймворка Spring Boot. Результатом работы стало создание системы управления складом шин, которая обеспечивает автоматизацию процессов учета, хранения и поиска данных. В ходе работы были успешно освоены такие технологии, как Spring Boot, PostgreSQL, Apache POI и Thymeleaf, а также получены навыки проектирования баз данных и пользовательских интерфейсов. Проект представляет собой законченное решение, которое может быть использовано для управления складом шин. В дальнейшем планируется углубленное изучение Spring Security для более эффективной защиты данных, а также изучение React для более современного подхода к проектированию веб-интерфейсов. Этот проект является важным этапом в развитии навыков веб-разработки и предоставляет прочную основу для дальнейшего изучения и применения современных технологий в данной области.

**Список литературы**

1. **Spring в действии,** Крейг Уоллс (6-е издание, 2022 год).
2. Учебный материал Гикбрейнс.
3. Статьи и видео-материалы из сети интернет.