**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ФАКУЛЬТЕТ СОЦИОКУЛЬТУРНЫХ КОММУНИКАЦИЙ**

**Кафедра информационных технологий**

­­­

**ИНТЕРНЕТ-ВИТРИНА МАГАЗИНА ПОДУШЕК**

Курсовой проект

Токмаковой Юлии Андреевны

студентки 4 курса,

специальности

«Прикладная информатика»

Научный руководитель:

доцент, к.т.н Царик С. В.

Минск, 2020

ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 4](#_Toc59203899)

[ГЛАВА 1. ОБЗОР ТЕХНОЛОГИЙ И СРЕДСТВ РАЗРАБОТКИ 5](#_Toc59203900)

[1.1 Анализ требований 5](#_Toc59203901)

[1.2 Обзор технологий и средств разработки 5](#_Toc59203902)

[ГЛАВА 2. РЕАЛИЗАЦИЯ СЕРВЕРНОЙ ЧАСТИ 13](#_Toc59203903)

[2.1 Проектирование базы данных 13](#_Toc59203904)

[2.2 Сущность базы данных 15](#_Toc59203905)

[2.3 Разработка клиент-серверного взаимодействия в стиле REST 16](#_Toc59203906)

[2.4 Загрузка, хранение и передача файлов 19](#_Toc59203907)

[2.5 Авторизация 21](#_Toc59203908)

[ГЛАВА 3. РЕАЛИЗАЦИЯ ПОЛЬЗОВАТЕЛЬСКОЙ ЧАСТИ 23](#_Toc59203909)

[3.1 Структура проекта 23](#_Toc59203910)

[3.2 Реализация Интернет-витрины 25](#_Toc59203911)

[3.3 Реализация панели администрирования 27](#_Toc59203912)

[ЗАКЛЮЧЕНИЕ 31](#_Toc59203913)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 32](#_Toc59203914)

[ПРИЛОЖЕНИЕ 33](#_Toc59203915)

РЕФЕРАТ

Курсовой проект содержит: 31 страницу, 10 иллюстраций, 12 листингов, 14 источников, 1 приложение.

Ключевые слова: ВЕБ-ПРИЛОЖЕНИЕ, JAVA, REST API, КОМПОНЕНТ, ANGULAR.

Цель работы – разработка интернет-витрины магазина подушек.

Результатом курсового проекта является веб-приложение, представляющее собой интернет-витрину магазина подушек. Веб-приложение также содержит панель администрирования для добавления, редактирования и удаления информации о продуктах магазина.

РЭФЕРАТ

Курсовая работа змяшчае: 31 старонку, 10 ілюстрацый, 12 лістынгаў, 14 крыніц, 1 прыкладанне.

Ключавыя словы: ВЭБ-ПРЫКЛАДАННЕ, JAVA, REST API, КАМПАНЕНТ, ANGULAR.

Мэта работы – стварэнне інтэрнэт-вітрыны крамы падушак.

Вынікам курсавога праекта з'яўляецца вэб-прыкладанне, якое ўяўляе сабой інтэрнэт-вітрыну крамы падушак. Вэб- прыкладанне таксама змяшчае панэль адміністравання для дадання, рэдагавання і выдалення інфармацыі аб прадуктах крамы.

ABSTRACT

Course work includes: 31 pages, 10 illustrations, 12 listings, 14 sources, 1 application.

Keywords: WEB APPLICATION, JAVA, REST API, COMPONENT, ANGULAR.

The purpose of the work is to develop an online showcase for pillow store.

The result of the course project is a web application that is an online showcase for a pillow store. The web application also contains an administration panel for adding, editing and deleting store product information.

ВВЕДЕНИЕ

Появление сети Интернет навсегда изменило способы ведения бизнеса. Сейчас трудно найти кампанию, не имеющюю своего представления в сети Интерент. Наличие сайта позволяет закрепить узнаваемость бренда, привлечь новых покупателей и увеличить объемы продаж.

С помощью сайта кампания может рассказать о своем уникальном торговом предложении, автоматизировать бизнес-процессы, улучшить качество и повысить скорость обслуживания покупателей. Наличие сайта повышает степень доверия к кампании, влияет на формирование заинтересованности клиентов к продуктам кампании.

Цель курсового проектирования заключается в создании интернет-витрины магазина подушек. Разработанное веб-приложение будет использоваться для привлечения новой аудитории, формирования информационного поля, направленного на узноваемость бренда, повышения имиджа кампании и лояльности клиентов.

В ходе курсового проектирования будет создана база данных для хранения информации о продуктах магазина.

Веб-приложение будет содержать представление каталога продуктов и представление отдельного продукта с его детальным описанием.

Будет реализована панель администрирования для создания, редактирования и удаления информации о продуктах магазина.

Для достижения цели были поставлены следующие задачи:

* Провести обзор инструментальных средств для разработки веб-приложения.
* Разработать REST систему клиент-серверного взаимодействия.
* Разработать систему авторизации пользователя-администратора.
* Спроектировать и подключить к веб-приложению базу данных.
* Разработать пользовательскую часть веб-приложения на базе фреймворка Angular.
* Разработать панель администрирования веб-приложения.

ГЛАВА 1. ОБЗОР ТЕХНОЛОГИЙ И СРЕДСТВ РАЗРАБОТКИ

* 1. Анализ требований

Первым этапом создания веб-приложения является формулировка и анализ требований, которым это веб-приложение должно отвечать.

Требования к системе делятся на функциональные и нефункциональные. Функциональные требования описывают сценарии взаимодействия пользователя с системой, тогда как нефункциональные накладывают дополнительные ограничения на систему, не относящиеся к её поведению.

После изучения предметной области и существующих решений были сформулированы следующие функциональные требования к веб-приложению:

* Просмотр каталога продуктов – пользователь должен иметь возможность просматривать каталог продуктов магазина.
* Просмотр информации об отдельном продукте.
* Авторизация пользователя-администратора ­­­­– необходимо обеспечить контроль доступа к хранимым данным.
* Добавление, редактирование и удаление информации о продуктах – необходимо разработать систему администрирования веб-приложения.

Был определен следующий ряд нефункциональных требований:

* Удобство использования­ ­­– веб-приложение должно быть адаптивным.
* Разграничение доступа – не авторизированный как администратор пользователь не должен иметь доступ к панели администрирования веб-приложения.

Определив требования к системе, можно приступать к выбору инструментальных средств разработки.

* 1. Обзор технологий и средств разработки

Рассмотрим инструменты и технологии, которые позволяют оптимальным образом удовлетворить описанным ранее требованиям к разрабатываемому веб-приложению.

Были выбраны следующие средства разработки для реализации серверной части веб-приложения: язык программирования Java, фреймворки Spring, Spring Boot, Maven, библиотеки Project Lombok и Java JSON Web Token, система управления базами данных PostgreSQL.

**Язык программирования Java**

Java – это строго типизированный объектно-ориентированный язык программирования высокого уровня. Java отличается быстротой, высоким уровнем защиты и надежностью [1]. Язык программирования Java прост в использовании. В нем присутствует механизм сборки мусора, что позволяет автоматически управлять памятью, а также строгая типизация, которая позволяет избежать многих ошибок при написании кода и делает Java удобным средством для написания крупных проектов.

Java является межплатформенным языком. Код Java, написанный на одной операционной системе, можно запустить без изменений на других операционных системах. Код, написанный на языке программирования Java, запускается с помощью виртуальной машины Java (Java Virtual Machine, JVM). JVM обрабатывает байтовый код, после чего процессор обрабатывает код, полученный от JVM. Все виртуальные машины Java работают аналогично, поэтому один и тот же код работает одинаково во всех операционных системах, что и делает Java межплатформенным языком программирования.

Важным преимуществом языка Java является наличие большого количества библиотек с открытым исходным кодом, которые подходят для решения задач любой сложности. В частности, при разработке серверной части веб-приложения будут использованы фреймворки и библиотеки для создания REST-сервисов, контроля доступа, авторизации, взаимодействия с базой данных, внедрения зависимостей, генерации и валидации токенов JWT.

**Spring**

Spring Framework, или Spring – один из самых популярных фреймворков для создания веб-приложений на языке программирования Java. Spring представляет собой коллекцию меньших фреймворков, большинство из которых могут работать независимо друг от друга, однако они обеспечивают большую функциональность при совместном использовании.

Spring предоставляет слой доступа к базам данных и предоставляет следующие возможности:

* Управление ресурсами – автоматическое получение и освобождение ресурсов базы данных.
* Обработка исключений – перевод исключений при доступе к данным в исключения Spring.
* Транзакционность – прозрачные транзакции в операциях с данными.

Spring предлагает широкий спектр классов, предназначенных для поддержки создания веб-приложений. Используя Spring, можно получить максимальную гибкость при выборе способа реализации пользовательского интерфейса для веб-приложения.

**Spring Boot**

Spring Boot используется для упрощения разработки приложений с помощью фреймворка Spring. С помощью Spring Boot можно создать полностью функциональное приложение за максимально короткий срок, требуя при этом минимум усилий от разработчика. Фреймворк Spring Boot имет следующие особенности:

* встроенный веб-сервер (Tomcat, Jetty или Undertow), который автоматически запускается вместе с приложением;
* набор готовых пакетов зависимостей (starter dependencies);
* автоматическая конфигурация как самого Spring, так и некоторых сторонних библиотек;
* возможность следить за состоянием приложения в реальном времени с помощью Spring Boot Actuator;
* не требует использовать XML-конфигурацию.

Используя Spring Framework в качестве основы, Spring Boot упрощает зависимости Spring и запускает приложения прямо из командной строки. Spring Boot неявно упаковывает необходимые сторонние зависимости для каждого типа приложения на основе Spring и предоставляет их разработчику в виде starter-пакетов (spring-boot-starter-web, spring-boot-starter-data-jpa и т.д.).

Starter-пакеты представляют собой набор дескрипторов зависимостей, которые можно включить в разрабатываемое приложение, что позволяет получить универсальное решение для всех, связанных со Spring технологий.

Например, чтобы использовать Spring Data JPA для доступа к базе данных, необходимо просто включить в проект зависимость spring-boot-starter-data-jpa и Spring Boot автоматически установит совместимые драйверы баз данных и библиотеки Hibernate.

При создании веб-приложения необходимо добавить зависимость spring-boot-starter-web, которая подтянет в проект все библиотеки, необходимые для разработки Spring MVC-приложений, таких как spring-webmvc, jackson-json, validation-api и Tomcat.

Таким образом, Spring Boot собирает все общие зависимости и определяет их в одном месте. Файл pom.xml, при использовании Spring Boot, содержит намного меньше строк, чем при использовании его в Spring-приложениях.

Для настройки и запуска Spring Boot приложений требуется следующее:

* Java 8+
* Apache Maven 3.x

**Maven**

Maven – это фреймворк с открытым исходным кодом, который используется для автоматизации сборки проектов на основе описания их структуры в файлах на языке POM (Project Object Model).

POM – это общая, унифицированая модель описания программных модулей, поддерживающая хранение не только аттрибутов отдельных модулей, но и высокоуровневых связей между ними. Также POM содержит информацию об инструментах, необходимых для поддержки жизненного цикла модуля.

В отличие от средства автоматизации сборки Apache Ant, Maven обеспечивает декларативную, а не императивную сборку проекта. В файлах описания проекта содержится его спецификация, а не отдельные команды выполнения. Задачи по обработке файлов, описанные в спецификации, Maven выполняет посредством их обработки последовательностью встроенных и внешних плагинов.

Maven используется для построения и управления проектами, написанными на Java, C#, Ruby, Scala, и других языках.

Проекты, использующие Maven, имеют унифицированную структуру каталогов:

* /src/main/java содержит исходный код программы;
* /src/main/resources содержит дополнительные ресурсы приложения, такие как SQL-скрипты, файлы конфигурации и т. д.;
* /src/test/java содержит исходный код тестовых классов;
* /src/test/resources содержит дополнительные ресурсы приложения, необходимые для тестирования;
* /target содержит результаты сборки, которые включают в себя дистрибутив программы, скомпилированные классы, результаты тестирования.

Информация для сборки проекта, поддерживаемого Maven, содержится в XML-файле pom.xml. При запуске Maven проверяет, содержит ли конфигурационный файл все необходимые данные и все ли данные синтаксически правильно записаны. Минимальная конфигурация включает версию конфигурационного файла, имя проекта, версию проекта, его автора. С помощью pom.xml конфигурируются зависимости от других проектов, список плагинов, реализующих порядок сборки [2].

**Project Lombok**

Lombok – это библиотека, которая добавляет дополнительную функциональность в Java c помощью изменения исходного кода перед Java компиляцией.

При разработке каждого Java-проекта существует необходимость писать стандартные методы доступа (getters и setters), сравнения (equals), вычисления хеш-кода (hashcode), преобразования объекта в строку (toString) и др. Project

Lombok используется для автоматизации этого процесса.

Кроме генерации исходного кода Lombok также позволяет:

* использовать неявные типы переменных (var и val);
* создавать конструкторы классов с разным набором параметров;
* создавать классы-строители;
* создавать неизменяемые классы;
* автоматически освобождать ресурсы;
* проверять аргументы методов на равенство null.

Достаточно включить Lombok в зависимости проекта, после чего его можно использовать. Пример подключения в случае использования Maven представлен в листинге 1.1.

*Листинг 1.1 – Пример подключения библиотеки Lombok*

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<version>0.11.0</version>

</dependency>

Чтобы Lombok автоматически сгенерировал служебные методы, такие как ToString, EqualsAndHashCode, Getter, Setter, RequiredArgsConstructor, необходимо добавить при объявлении класса аннотацию @Data.

**Java** **JSON Web Token**

JSON Web Token (JWT) представляет собой стандартизированный, в некоторых случаях подписанный и/или зашифрованный формат упаковки данных, который используется для безопасной передачи информации между двумя сторонами [3].

JWT определяет особую структуру информации, которая отправляется по сети. Она представлена в двух формах – сериализованной и десериализованной. Сериализованная форма используется для передачи данных. С другой стороны, чтобы читать и записывать информацию в токен, нужна его десериализация.

В несериализованном виде JWT состоит из заголовка и полезной нагрузки, которые являются JSON-объектами. Заголовок используется для описания криптографических функций, которые применяются для подписи и/или шифрования токена. Если JWT подписан и/или зашифрован, в заголовке указывается заявка alg и имя алгоритма шифрования.

Заявка – это часть информации, аналогичная ключу объекта JSON. Заявки бывают служебными и пользовательскими. Служебные заявки являются частью какого-либо стандарта и имеют определенные значения. Наиболее распространенные служебные заявки:

* iss – издатель токена;
* sub – описываемый объект;
* aud – получатели;
* exp – дата истечения срока действия;
* iat – время создания.

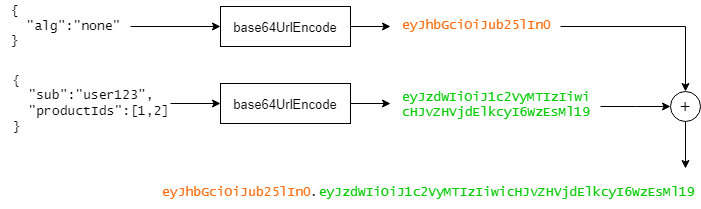
JWT токен также содержит полезную нагрузку – часть токена, в которой размещается необходимая пользовательская информация.

JWT в сериализованной форме представляет собой строку следующего формата:

[ Header ].[ Payload ].[ Signature ]

Заголовок (header) и полезная нагрузка (payload) присутствуют всегда, в то время как подпись (signature) может отсутствовать.

На рисунке 1.1 представлен процесс сериализации неподписанного JWT токена:



*Рисунок 1.1 – Процесс сериализации JWT токена*

Процесс сериализации JWT состоит из кодирования заголовка, полезной нагрузки и подписи, если она есть, с помощью алгоритма base64url.

Библиотека Java JSON Web Token (JJWT) используется для кодирования веб-токенов в приложениях Java. JJWT стремится быть самой простой в использовании и понятной библиотекой для создания и верефикации веб-токенов на JVM и Android [4]. Библиотека представляет собой чистую реализацию Java, основанную исключительно на спецификациях JWT, JWS, JWE, JWK и JWA RFC и с открытым исходным кодом.

**PostgreSQL**

PostgreSQL – это объектно-реляционная система управления базами данных с открытым исходным кодом.

PostgreSQL поддерживает большую часть стандарта SQL и предлагает следующие функции:

* сложные запросы;
* внешние ключи;
* триггеры;
* изменяемые представления;
* транзакционная целостность;
* многоверсионность.

Кроме того, пользователи могут расширять возможности PostgreSQL, например создавая свои типы данных, функции и операторы, процедурные языки.

PostgreSQL предоставляет свободную лицензию. Это значит, что ее можно использовать бесплатно как для личных, так и для коммерческих целей.

Клиентская часть веб-приложения будет реализована с помощью следующих средств разработки: фреймворк Angular, язык программирования TypeScript, препроцессор Sass.

**Angular**

Фреймворк Angular является одной из самых надежных платформ для построения одностраничных веб-приложений. Приложения, разработанные с помощью фреймворка Angular, состоят из компонентов. Компоненты содержат шаблоны HTML в сочетании со специальными объектами и служат для управления отдельными частями страницы.

Работая с компонентами и шаблонами вместо элементов DOM, приложения Angular могут действовать на более высоком уровне абстракции с меньшим объемом кода по сравнению с приложениями, которые используют только JavaScript или библиотеку jQuery. Angular также задает определенные способы организации файлов скриптов на стороне клиента. По соглашению приложения Angular используют общую структуру папок, в которой файлы модуля и скрипта компонента располагаются в папке приложения. Скрипты Angular, используемые при построении, развертывании и тестировании приложения, обычно располагаются в папке более высокого уровня.

**TypeScript**

Приложения Angular разрабатываются на языке TypeScript. TypeScript – это язык с открытым исходным кодом, который основан на JavaScript, одном из наиболее часто используемых в мире языков программирования. TypeScript расширяет возможности JavaScript путем добавления определений статических типов [5].

Типы предоставляют способ описать форму объекта, обеспечивая документируемость кода и позволяя TypeScript проверять правильность работы кода.

Код, написанный на языке программирования TypeScript, преобразуется в код JavaScript с помощью компилятора TypeScript или Babel. Преобразованный код представляет собой чистый, простой код JavaScript, который запускается везде, где выполняется JavaScript: в браузере, на Node.JS или в пользовательских приложениях.

**Препроцессор Sass**

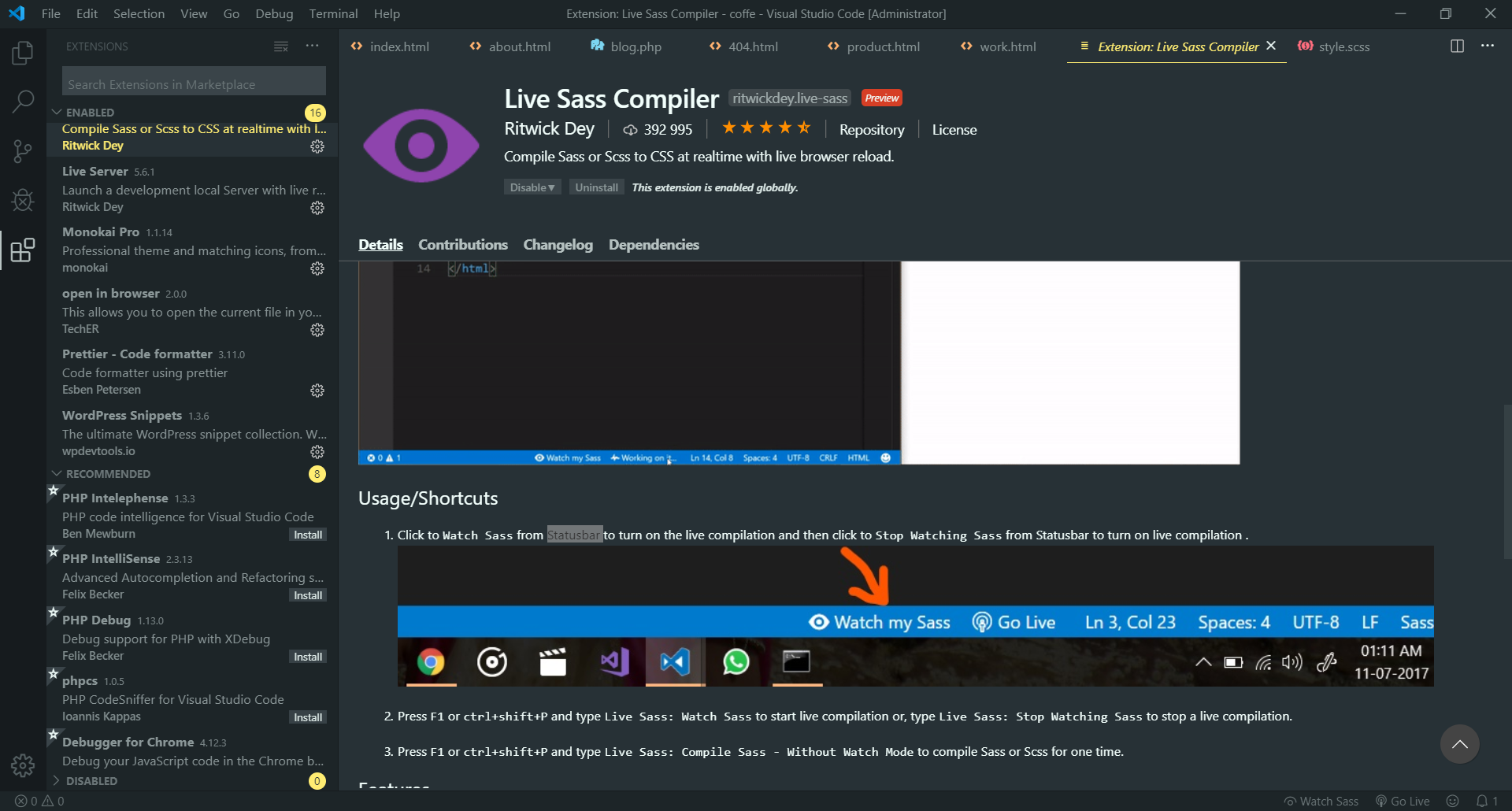
CSS-препроцессор — это программа или инструмент, имеющий свой собственный синтаксис, который затем компилируется в стандартный CSS-код. CSS-препроцессор имеет различные функции, помогающие разработчикам писать CSS-код лучше, быстрее и чище.

Существует несколько CSS-препроцессоров, таких как Sass, Less и Stylus. Все CSS-препроцессоры имеют схожие функции и отличаются лишь синнтаксисом.

Sass является одним из наиболее популярных CSS-препроцессоров. Функции, предостовляемые Sass препроцессором:

* Переменные. Переменные в Sass используются для хранения и переиспользования информации на протяжении написания всех стилей проекта. В переменных можно хранить цвета, стеки шрифтов или любые другие значения CSS. В Sass используется символ $ для создания переменной.
* Вложенность — Sass позволяет вкладывать правила CSS друг в друга, как и в визуальной иерархии HTML. Вложенные правила применяются только для элементов, соответствующих внешним селекторам.
* Фрагментирование — можно создавать фрагменты Sass-файла, содержащие в себе небольшие кусочки кода CSS, и переиспользовать данные фрагменты в других Sass-файлах. Фрагменты позволяют сделать CSS код модульным, и облегчить его обслуживание. Фрагмент – это Sass-файл, имя которого начинается с нижнего подчеркивания, например, \_partial.scss. Нижнее подчеркивание в имени Sass-файла говорит компилятору о том, что это только фрагмент и он не должен компилироваться в CSS. Фрагменты в Sass подключаются при помощи директивы @import.
* Миксины (примеси) – используются для создания набора CSS-свойств. Миксины можно использовать повторно или смешивать с другими значениями. Миксины можно использовать для создания отдельных тем одного макета. Миксины могут принимать аргументы, что позволяет создавать большое разнообразие стилей при помощи небольшого количества миксинов.
* Наследование. Используя директиву @extend можно наследовать наборы свойств CSS от одного селектора другому [8]. Использование наследования позволяет сохранить скомпилированный CSS код чистым и аккуратным.
* Математические операторы. Sass поддерживает математические операторы.
* Функции – можно создавать определения CSS в виде функций для многократного использования [9].

В среде разработки VSCode для компиляции Sass файлов в CSS код в режиме реального времени используется плагин «Live Sass Compiler» [11]. Чтобы установить плагин «Live Sass Compiler», необходимо открыть среду разработки VSCode, нажать комбинацию клавиш ctrl+P и в строке поиска ввести следующую строку: install live-sass. После установки плагина в строке состояния среды разработки появится кнопка «Watch Sass»:



*Рисунок 1.2 – Кнопка для запуска «Live Sass Compiler»*

После нажатия кнопки «Watch Sass» плагин «Live Sass Compiler» будет запущен.

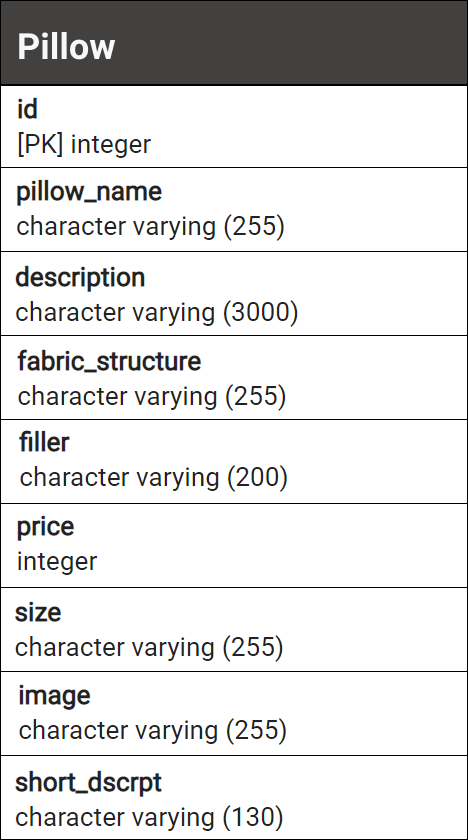
ГЛАВА 2. РЕАЛИЗАЦИЯ СЕРВЕРНОЙ ЧАСТИ

* 1. Проектирование базы данных

База данных – это совокупность массивов и файлов данных, организованная по определённым правилам, предусматривающим стандартные принципы описания, хранения и обработки данных независимо от их вида [7]. Создание и подключение базы данных необходимо для реализации гибкого и расширяемого приложения. База данных позволяет структурировать информацию приложения. С помощью базы данных можно извлекать контент и выводить его в информационных блоках на страницах веб-приложения.

Разрабатываемое веб-приложение использует систему управления базами данных PostgreSQL. База данных была создана с помощью программы pgAdmin. pgAdmin – это открытая платформа администрирования и разработки для PostgreSQL и связанных с ней систем управления базами данных, написанная на Python и jQuery [12].

Для выполнения функциональных требований к разрабатываемому веб-приложению, определенных в главе 1, был выделен класс-сущность базы данных Pillow (подушка). Схема таблицы «pillowTable» представлена на рисунке 2.1.



*Рисунок 2.1 – Схема таблицы «pillowTable»*

Таблица «pillowTable» предназначена для хранения информации о продукте магазина и имеет следующие поля:

* id – идентификационный номер продукта;
* pillow\_name – название подушки;
* description – полное описание;
* short\_dscrpt – краткое описание подушки;
* fabric\_structure – материал чехла подушки;
* filler – материал наполнителя подушки;
* size – размеры подушки;
* price – цена;
* image – путь к файлу изображения подушки.

Подключение к базе данных осуществляется с помощью кода, представленного в листинге 2.1.

*Листинг 2.1 – Код для подключения к базе данных*

spring.datasource.url=jdbc:postgresql://localhost:5432/pillows

spring.datasource.username=postgres

spring.datasource.password=\*\*\*\*

spring.jpa.show-sql=true

spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.PostgreSQLDialect

spring.jpa.hibernate.ddl-auto = update

Свойство spring.datasource.url используется для указания местоположения базы данных. В свойстве содержатся следующие три значения:

* имя драйвера SQL;
* URL-адрес сервера базы данных;
* имя базы данных.

В свойстве spring.datasource.username указывается имя пользователя для доступа к базе данных, а в свойстве spring.datasource.password – пароль.

При указании свойства spring.jpa.show-sql со значением true все запросы, запускаемые Hibernate, будут выводится в консоль, что очень удобно при отладке приложения.

Свойство spring.jpa.properties.hibernate.dialect помогает Hibernate правильно сгенерировать SQL для указанной базы данных.

Свойство spring.jpa.hibernate.ddl-auto со значением update указывает, что каждый раз при запуске сервера будет обновлятся уже существующая база данных, а не создаваться новая.

* 1. Сущность базы данных

Сущность «Pillow» базы данных размещенная в пакете domain в файле Pillow.java, используется для описания продукта магазина. Реализация класса Pillow представлена в листинге 2.2.

*Листинг 2.2 – Реализация класса Pillow*

@Data

@Entity

@Table(name="pillowTable")

public class Pillow {

@Id

@GeneratedValue(strategy=GenerationType.AUTO)

private Integer id;

@Column(name = "pillowName", nullable = false)

private String pillowName;

@Column(name = "shortDscrpt")

private String shortDscrpt;

@Column(name = "description", nullable = false)

private String description;

@Column(name = "fabricStructure")

private String fabricStructure;

@Column(name = "filler", length = 200)

private String filler;

@Column(name = "size")

private String size;

@Column(name = "price", nullable = false)

private Double price;

@Column(name = "image")

private String fileName;

protected Pillow() {}

}

Класс Pillow содержит девять полей, соответствующих столбцам таблицы «Pillow» базы данных. Класс Pillow отмечен аннотацией @Entity, которая представляет класс Entity в Spring Boot. Если класс помечен аннотацией Entity, то это указывает на то, что данный класс является представлением какой-либо сущности базы данных.

Аннотация @Table указывает, какая таблица базы данных сопоставлена с классом Entity, в данном случае таблица «pillowTable» сопоставлена с классом Pillow.

При объявлении поля id используется аннотация @Id. Аннотация @Id используется для представления столбца первичного ключа в таблице базы данных, в данном случае поле id представляет первичный ключ. Аннотация @GeneratedValue используется для определения стратегии генерации уникального идентификатора. В данном случае идентификатор будет генерироваться автоматически исходя из возможностей базы данных.

Поле, помеченное аннотацией @Column, сопоставляется со столбцом таблицы, название которого указанно в скобках.

* 1. Разработка клиент-серверного взаимодействия в стиле REST

Взаимодействие клиента с веб-сервисом приложения основано на использовании архитектурного стиля Representational State Transfer (REST). REST – это стиль архитектуры программного обеспечения для построения распределенных масштабируемых веб-сервисов [13]. Термин REST был введен в 2000 году Роем Филдингом, одним из авторов HTTP-протокола. Системы, поддерживающие REST, называются RESTful-системами.

REST API представляет собой набор конечных точек (веб-адресов), обращаясь к которым с помощью HTTP-запросов, клиент получает в ответ информацию с сервера в формате JSON, XML или HTML.

RESTful-системы имеют следующие свойства:

* производительность – взаимодействие компонентов системы может являться доминирующим фактором производительности и эффективности сети с точки зрения пользователя;
* масштабируемость для обеспечения большого числа компонентов и взаимодействий компонентов.

Рой Филдинг описывает влияние архитектуры REST на масштабируемость следующим образом:

* простота унифицированного интерфейса;
* открытость компонентов к возможным изменениям для удовлетворения изменяющихся потребностей (даже при работающем приложении);
* прозрачность связей между компонентами системы для сервисных служб;
* переносимость компонентов системы путем перемещения программного кода вместе с данными;
* надежность, выражающаяся в устойчивости к отказам на уровне системы при наличии отказов отдельных компонентов, соединений или данных [14].

В RESTful-системах для манипулирования ресурсами используются стандартные HTTP-методы в соответствии со спецификацией протокола:

* GET – используется для получения текущего представления ресурсов;
* POST – используется для создания нового ресурса;
* PATCH используется для полного или частичного обновления существующего ресурса;
* PUT – для создания нового ресурса или обновления существующего. Если ресурс с заданным идентификатором найден, информация о нем должна быть обновлена, в иначе будет создан новый ресурс;
* DELETE используется для удаления существующего ресурса.

Серверу при обработке REST-запросов нет необходимости извлекать контекст приложения, поскольку клиент включает в запрос все необходимые данные, используя для этого заголовки и тело запроса. Такой подход повышает производительность, упрощает дизайн и реализацию серверных компонентов системы.

Разрабатываемая в стиле REST архитектура должна следовать следующим основным принципам:

* Использование клиент-серверной модели – отделение потребности интерфейса клиента от потребностей сервера, хранящего данные, повышает переносимость кода клиентского интерфейса на другие платформы, а упрощение серверной части улучшает масштабируемость.
* Использование стандартных методов протокола HTTP.
* Использование уникальных идентификаторов ресурсов – все ресурсы должны идентифицироваться в запросах, например, с использованием URI (унифицированный идентификатор ресурса).
* Кэширование – использование кэширования способно полностью или частично устранить некоторые клиент-серверные взаимодействия, что повышает производительность и расширяемость системы.
* Отсутствие состояния на стороне сервера – все запросы от клиента должны быть составлены так, чтобы сервер получил всю необходимую информацию для выполнения запроса. Состояние сессии при этом сохраняется на стороне клиента.

Соблюдение вышеперечисленных принципов повышает надежность системы за счет отсутствия необходимости сохранять информацию о состоянии клиента, которая может быть утеряна. Благодаря использованию кэширования также повышается производительность.

Система, разработанная в стиле REST, обладает такими преимуществами, как простота и единообразие интерфейсов, портативность программных компонентов, легкость внесения изменений, а также способность за короткий промежуток времени приспосабливаться к новым требованиям.

HTTP-запросы RESTful-сервиса обрабатываются контроллером. В разрабатываемом веб-сервисе класс-контроллер PillowController размещен в пакете controller в файле PillowController.java. Полный код файла PillowController.java представлен в приложении.

Контроллер отмечается аннотацией @RestController, которая указывает, что в данном классе будет реализована логика обработки клиентских запросов.

Контроллер PillowController реализует CRUD (Create, Read, Update, Delete) операции над объектами класса Pillow. При объявлении класса указана аннотация @RequestMapping("/api/products"), которая определяет базовый uri-адресс операций.

Рассмотрим реализацию CRUD операций на примере операции Delete. В листинге 2.3 представлен код метода, осуществляющего удаление объекта класса Pillow по его идентификационному номеру.

*Листинг 2.2 – Реализация метода для удаления объекта класса Pillow*

@DeleteMapping("/{id}")

public ResponseEntity<?> deletePillow (@PathVariable(value = "id") Integer pillowId)

{

Pillow pillow = pillowRepo.findById(pillowId).get();

final boolean deleted = pillowRepo.delete(pillow);

return deleted

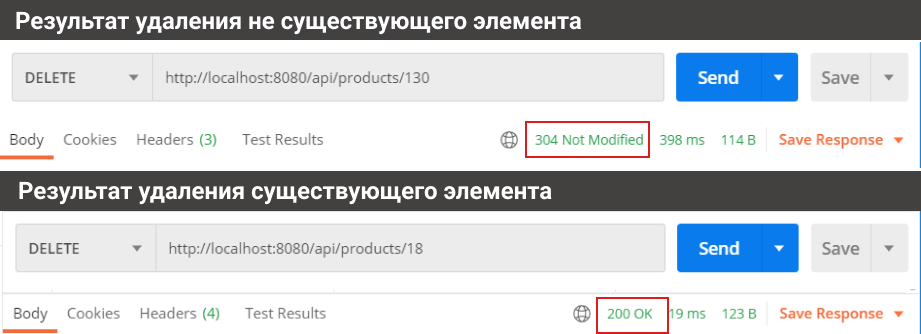
? new ResponseEntity<>(HttpStatus.OK)

: new ResponseEntity<>(HttpStatus.NOT\_MODIFIED);

}

Аннотация @DeleteMapping("/{id}") обозначает, что представленный в листинге 2.2 метод обрабатывает DELETE запросы на адрес /api/products/{id}, где id – это пременная пути, определенная в параметрах метода с помощью аннотации @PathVariable(value = "id"). В теле метода происходит поиск в базе данных элемента с соответствующим идентификационным номером. Если элемент с таким идентификационным номером существует, то произойдет его удаление, после чего метод вернет статус 200 OK. Иначе – база данных останется неизменна, а в качестве возвращаемого параметра будет отправлен статус 304 Not Modified.

Результаты работы метода deletePillow() при удалении существующего и не существующего элементов представлены на рисунке 2.2.



*Рисунок 2.2 – Результаты работы метода deletePillow()*

Для работы непосредственно с базой данных был создан репозиторий и размещен в пакете repo в файле PillowRepo.java. Репозитории относятся к уровню данных приложения и представляют из себя набор методов, с помощью которых сервисы манипулируют информацией из базы данных, абстрагируясь от механизма их хранения. Реализация класса-репозитория, размещенного в файле PillowRepo.java представлена в листинге 2.3.

*Листинг 2.3 – Реализация класса-репозитория*

package com.pillows.springbootpillowsapi.repo;

import com.pillows.springbootpillowsapi.domain.Pillow;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

@Repository

public interface PillowRepo extends JpaRepository<Pillow, Integer>{

}

Класс PillowRepo наследуется от класса JpaRepository в котром реализованы все методы, необходимые для выполения CRUD операций. В угловых скобках указывается имя класса-сущности базы данных (в данном случае это класс Pillow) и тип первичного ключа.

Аннотация @Repository указывает, что данный класс создан для реализации всех CRUD операций, таких как создание, обновление, удаление и чтение из базы данных.

* 1. Загрузка, хранение и передача файлов

Согласно требованиям к веб-приложению, определенным в первой главе, необходимо реализовать возможность загрузки файлов-изображений продуктов. В базе данных хранятся лишь пути к файлам изобрображений в виде строковых данных. Сами файлы необходимо хранить на сервере в качестве статических ресурсов. По умолчанию Spring Boot предоставляет статическое содержимое из каталога /static. Чтобы изменить это поведение необходимо добавить класс-конфигуратор, наследуемый от класса WebMvcConfigurer и переопределить в нем метод addResourceHandlers(). Пример переопределения метода addResourceHandlers()представлен в листинге 2.4.

*Листинг 2.4 – Пример переопределения метода addResourceHandlers()*

@Configuration

public class MvcConfig implements WebMvcConfigurer {

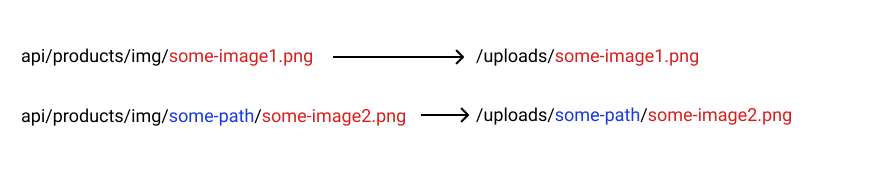
@Override

public void addResourceHandlers(ResourceHandlerRegistry registry) { registry.addResourceHandler("api/products/img/\*\*")

.addResourceLocations("file:uploads/"); }

}

Метод addResourceHandler() позволяет определить по какому URL-адресу изображение будет доступно клиенту. С помощью метода addResourceLocations() определяется фактическое расположение источника данных. На рисунке 2.3 представлена схема сопоставления URL-адреса изображения и его фактического расположения на сервере.



*Рисунок 2.3 – Схема сопостовления URL-адресов и фактических адресов файлов изображений*

Загрузка файлов изображений на сервер осуществляется в классе PillowController с помощью методов createPillow() и updatePillow() (при добавлении и обновлении данных о продукте соответственно). Рассмотрим реализацию загрузки файла на примере метода createPillow():

*Листинг 2.5 – Пример реализации загрузки файла изображения на сервер*

@PostMapping("")

public Pillow createPillow

(@RequestParam("pillowName") String pillowName,

…

@RequestParam("file") MultipartFile file,

HttpServletRequest request

) throws IOException

{

Pillow pillow = new Pillow(…);

if (file != null && !file.getOriginalFilename().isEmpty())

{

String uploadDirectory = System.getProperty("user.dir") +

"/uploads";

File uploadDir = new File(uploadDirectory);

if (!uploadDir.exists()){ uploadDir.mkdir(); }

String uuidFile = UUID.randomUUID().toString();

String resultFileName = uuidFile + '.' +

file.getOriginalFilename();

file.transferTo(new File(uploadDirectory + "/" +

resultFileName));

pillow.setFileName(resultFileName);

}

…

}

С помощью метода System.getProperty("user.dir") определяем текущий каталог и находим в нем каталог uploads, это и будет фактическим расположением всех загружаемых на сервер файлов. Если такого каталога еще не существует, то создается новый с помощью метода uploadDir.mkdir().

Далее необходимо сгенерировать уникальное имя загружаемого файла, чтобы избежать коллизий при сохранении. Генерация осуществляется с помощью метода UUID.randomUUID() который генерирует уникальный 128-битный идентификатор, причем вероятность сгенерировать один и тот же идентификатор дважды так мала, что ее можно считать нулевой. На основе этого идентификатора генерируется уникальное имя файла.

С помощью метода file.transferTo() происходит перемещение файла в каталог сервера uploads, а чтобы в последующем клиент мог получить доступ к загруженному изображению, путь к файлу сохраняется в базе данных в качестве строки.

* 1. Авторизация

Авторизация необходима для ограничения прав доступа к POST, PUT и DELETE HTTP-запросам пользователей, не являющихся администраторами. Авторизироваться пользователь может, отправив POST-запрос на адрес api/login. При этом необходимо передать логин и пароль администратора. Если логин и пароль окажутся верными, то с помощью метода generateJWTToken() будет сгенерирован JWT токен и отправлен пользователю в качестве возврачаемого параметра. Реализация генерации токена в методе generateJWTToken() представлена в листинге 2.6.

*Листинг 2.6 – Генерация JWT токена*

private Map<String, String> generateJWTToken(String name) {

long timestamp = System.currentTimeMillis();

long TOKEN\_VALIDITY = 2 \* 60 \* 60 \* 1000;

String token = Jwts.builder().signWith(SignatureAlgorithm.HS256, API\_SECRET\_KEY)

.setIssuedAt(new Date(timestamp))

.setExpiration(new Date(timestamp + TOKEN\_VALIDITY))

.claim("userName", name)

.compact();

Map<String, String> map = new HashMap<>();

map.put("token", token);

return map;

}

Токен генерируется с помощью метода Jwts.builder() библиотеки JJWT. При генерации токена с помощью методов setIssuedAt() и setExpiration() указывается дата создания токена и время, по истечению которого, токен станет недействительным соответственно.

В качестве полезной нагрузки с помощью метода claim() указывается имя авторизовываемого пользователя.

Был создан фильтр, чтобы ограничить неавторизованным пользователям доступ к API. В качестве фильтра используется класс AuthFilter, который наследует класс GenericFilterBean и переопределяет метод doFilter(). В переопределенном методе doFilter() происходит проверка токена на валидность с помощью метода Jwts.parser() библиотеки JJWT. Если токен верный, то происходит дальнейшее выполнение запроса, иначе – возвращается статус 403 Forbidden.

Код класса AuthFilter, реализующего проверку токена представлен в приложении.

Пример регистрации фильтра представлен в листинге 2.7.

*Листинг 2.7 – Пример регистрации фильтра*

@Bean

public FilterRegistrationBean<AuthFilter> filterRegistrationBean() {

FilterRegistrationBean<AuthFilter> registrationBean = new

FilterRegistrationBean<>();

AuthFilter authFilter = new AuthFilter();

registrationBean.setFilter(authFilter);

registrationBean.addUrlPatterns("/api/products/\*");

return registrationBean;

}

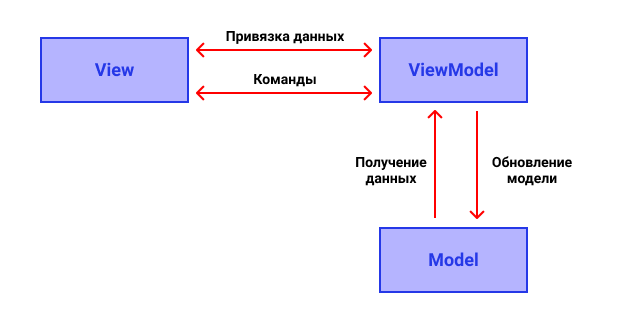
В примере, представленном в листинге 2.7, создается объект класса FilterRegistrationBean, с помощью которого происходит регистрация ранее созданного фильтра AuthFilter. Далее с помощью метода addUrlPatterns() определяется URL-адрес действия фильтра. Таким образом все запросы, URL-адрес которых начинается со строки «/api/products» автоматически будут перенаправлены для фильтрации.

ГЛАВА 3. РЕАЛИЗАЦИЯ ПОЛЬЗОВАТЕЛЬСКОЙ ЧАСТИ

* 1. Структура проекта

При разработке веб-приложения был использован архитектурный паттерн проектирования MVVM (Model-View-ViewModel). Использование паттерна MVVM позволяет отделить логику приложения от визуальной составляющей, тем самым создать слабо связанную, легко расширяемую структуру приложения.

Паттерн MVVM состоит из трех компонентов: модели (Model), модели представления (ViewModel) и представления (View). Схема связей между компонентами паттерна MVVM представлена на рисунке 3.1.



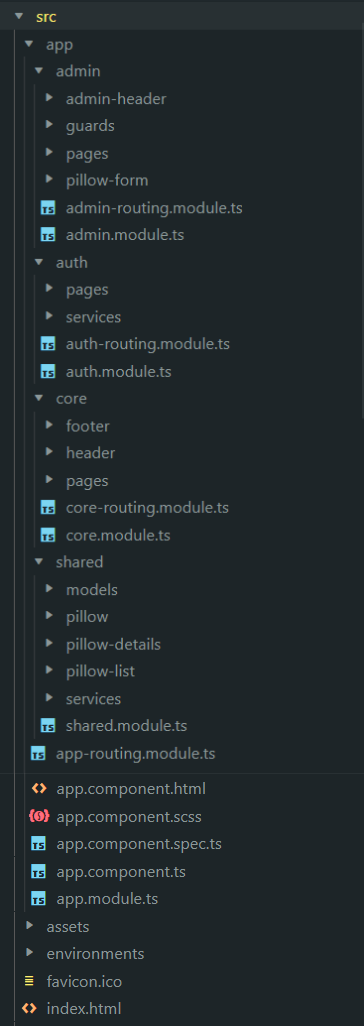
*Рисунок 3.1 – Схема связей между компонентами паттерна MVVM*

Модель описывает используемые в приложении данные. Представление или View определяет визуальный интерфейс, с помощью которого пользователь взаимодействует с приложением. Модель представления (ViewModel) связывает модель и представление через механизм привязки данных. Модель представления также содержит логику по получению данных из модели, которые потом передаются в представление. Также модель представления определяет логику по обновлению данных в модели.

Структура проекта представляет собой систему папок, включающую в себя следующие структурные единицы:

* Файлы компонентов, где компонент – это часть интерфейса приложения с собственной логикой. Компонент состоит из трех частей, каждая из которых распологается в отдельном файле:
  + .htm – в этом файле содержится шаблон компонента;
  + .scss – файл для стилей компонента;
  + .js – файл контроллера компонента в котором содержится вся логика поведения и управления данными, такая как порядок выполнения запросов к серверу, логика фильтров, обработчики событий.
* Файлы моделей.
* Файлы сервисов, использующихся для обработки и предоставления данных компонентам.

Файловая структура проекта представлена на рисунке 3.2.



*Рисунок 3.2 – Структура проекта*

Приложения Angular имеют модульную структуру. Модули служат изолирующей логической объединяющей структурой для компонентов и сервисов. Разрабатываемое приложение разделено на четыре модуля:

* модуль авторизации – в нем содержаться компоненты и сервисы, связанные со страницей авторизации пользователя;
* модуль администрирования – содержит в себе все необходимые компоненты для реализации панели администрирования;
* модуль, содержащий компоненты и сервисы, обеспечивающие вывод списка продуктов, а также информации об отдельном продукте;
* модуль общих компонентов и сервисов, которые переиспользуются в других модулях.

Файлы модулей расположены в каталогах auth, admin, core и shared соответственно.

Разделение приложения на модули способствует организации «ленивой загрузки» файлов компонентов. «Ленивая загрузка» – это техника, при которой часть веб-страницы загружается в более поздний момент времени, когда эта часть действительно необходима [6]. Таким образом, повышается скорость первоначальной загрузки приложения.

* 1. Реализация Интернет-витрины

Первым этапом разработки пользовательской части веб-приложения стала реализация Интернет-витрины. Все компоненты и сервисы, необходимые для работы Интернет-витрины, содержатся в модуле core, также для вывода информации о продуктах используются некоторые общие компоненты, определенные в модуле shared, такие как:

* PillowComponent – отвечает за вывод карточки продукта;
* PillowListComponent – осуществляет вывод списка карточек продуктов;
* PillowDetailsComponent – используется для вывода подробной информации о продукте;
* PillowService – сервис, содержащий методы для отправки и обработки запросов на сервер.

На главной странице Интернет-витрины представлен список продуктов магазина. Вывод списка продуктов осуществляется в компоненте PillowListComponent следующим образом: при инициализации компонента в методе ngOnInit() происходит вызов метода сервиса getPillowsList(), в котором осуществляется GET запрос к разработанному ранее API. Для взаимодействия с сервером и отправки запросов по протоколу http применяется класс HttpClient, который определяет ряд методов для отправки таких запросов, как GET, POST, PUT, DELETE.

Методы класса HttpClient после выполнения запроса возвращают объект Observable<any>, который определен в библиотеке RxJS ("Reactive Extensions"). Библиотека RxJS не является частью Angular, однако широко используется при взаимодействии с сервером по http. В библиотеке RxJS реализован паттерн "асинхронный наблюдатель" (asynchronous observable), поэтому выполнение запроса к серверу с помощью класса HttpClient выполняются в асинхронном режиме.

Реализация вывода списка продуктов в компоненте PillowListComponent представлена в листинге 3.1.

*Листинг 3.1 – Реализация вывода списка продуктов*

// метод контроллера компонента:

ngOnInit(): void {

this.pillowService.getPillowsList().subscribe((data) => {

this.pillowService.pillowsList = data;

}, err => console.error(err));

}

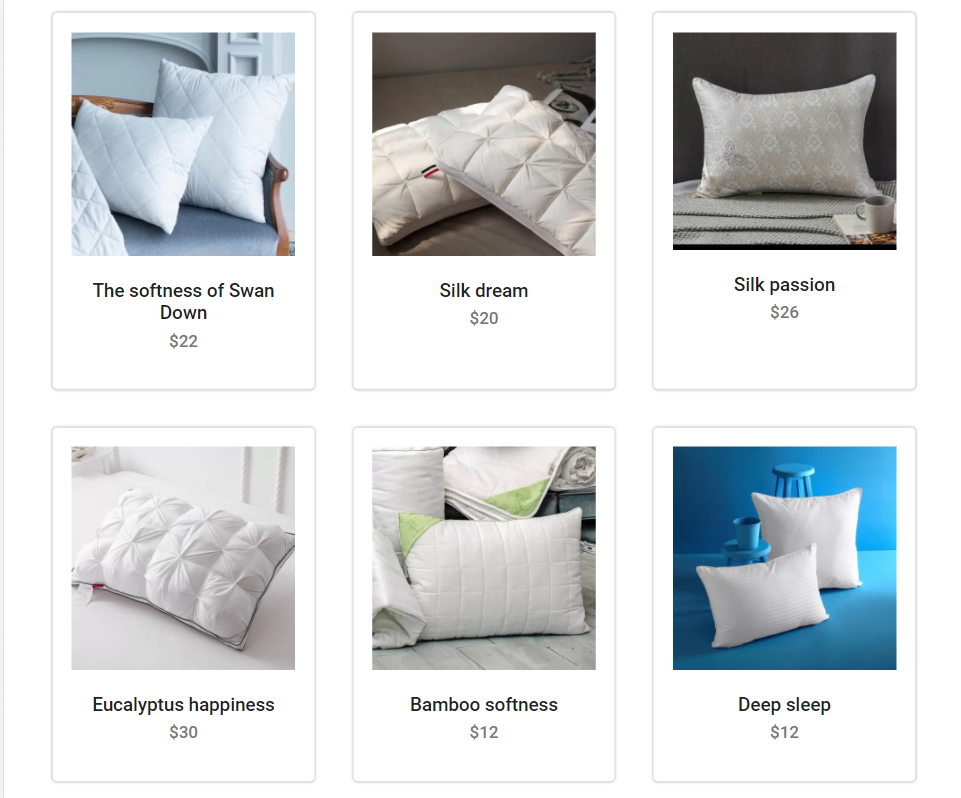
//метод сервиса:

getPillowsList(): Observable<Pillow[]> {

return this.http.get<Pillow[]>(this.baseUrl);

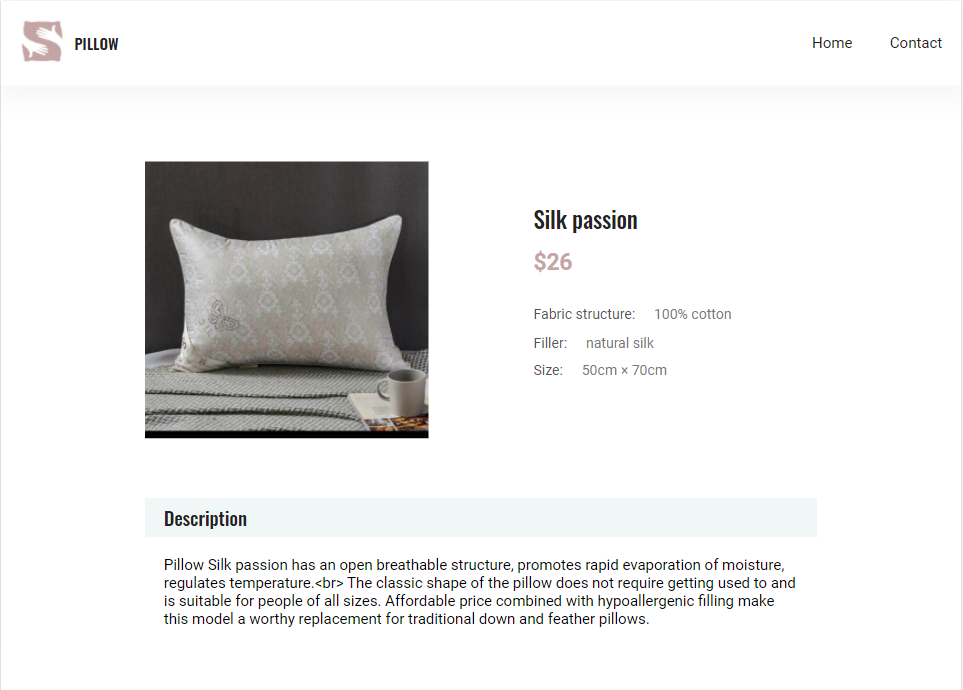
}

Результат выполнения кода из листинга 3.1 представлен на рисунке 3.3.



*Рисунок 3.3 – Список продуктов Интернет-витрины*

Каждый элемент списка продуктов содержит ссылку на страницу с подробным описанием продукта. Вывод информации о продукте осуществляется с помощью компонента PillowDetailsComponent, расположенного в модуле shared. Фрагмент страницы, содержащей детальную информацию о продукте представлен на рисунке 3.4.



*Рисунок 3.4 – Фрагмент страницы с деттальной информацией о продукте*

При инициализации компонента PillowDetailsComponent происходит вызов метода сервиса getPillow(), которому передается идентификационный номер продукта. Метод getPillow() осуществляет GET запрос к API и возвращает данные о продукте. Если продукт с таким идентификационным номером не был найден, то происходит перенаправление на страницу ошибки.

* 1. Реализация панели администрирования

Панель администрирования – панель управления сайтом, с которой может работать любой пользователь, не имеющий навыков программирования [10].

Осуществить вход в панель администрирования можно, набрав в адресной строке браузера после имени домена слово admin. При осуществлении первой попытки входа в панель администрирования произойдет переадресация на страницу авторизации пользователя. Страница авторизации содержит форму для ввода логина и пароля администратора. Если введенные данные окажутся верными, пользователь будет переадресован на главную страницу панели администрирования, иначе на экране появится сообщение об ошибке.

Чтобы ограничить доступ к панели администрирования неавторизированным пользователям был создан класс AuthGuard, который реализует интерфейс CanActivate. Интерфейс CanActivate позволяет управлять доступом к ресурсу при маршрутизации. В классе AuthGuard переопределяется метод canActivate(), реализация которого представлена в листинге 3.2.

*Листинг 3.2 – Реализация метода canActivate()*

canActivate(

next: ActivatedRouteSnapshot,

state: RouterStateSnapshot): Observable<boolean > | boolean {

const token = localStorage.getItem('token');

if (token) { return true; }

this.router.navigate(['/login']);

return false;

}

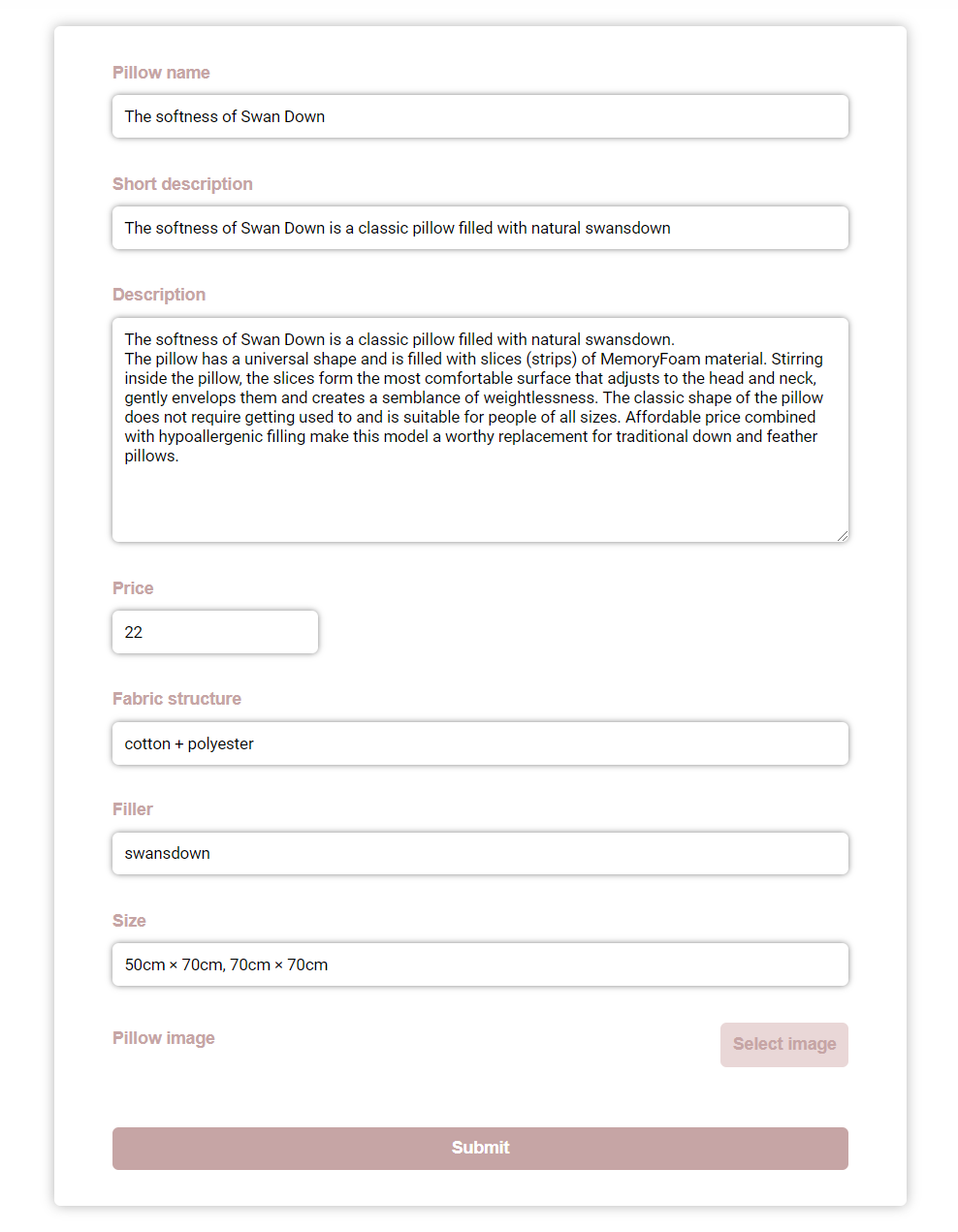
Переопределенный метод canActivate() получает два параметра – объекты ActivatedRouteSnapshot и RouterStateSnapshot, которые содержат информацию о запросе. Объект ActivatedRouteSnapshot содержит различную информацию из запроса, в том числе параметры маршрута и строки запроса. В теле метода происходит проверка токена, который генерируется при успешной авторизации пользователя. Если токен существует, то происходит перенаправление на главную старницу панели администрирования, иначе – загружается страница авторизации пользователя.

С помощью панели администрирования можно осуществлять добавление, редактирование и удаление информации о продуктах магазина. Добавить информацию о новом продукте можно нажав на кнопку «Add new item», расположенную в верхней части главной страницы панелели администрирования. После нажатия на кнопку откроется страница добавления нового продукта. На странице добавления нового продукта расположена форма, содержащая следующие поля:

* Pillow name – для ввода названия продукта.
* Short description – краткое описание.
* Description – подробное описание.
* Price – цена продукта.
* Fabric structure – материал чехла.
* Filler – материал наполнителя.
* Size – размеры продукта.

Форма обеспечивает валидацию введенных пользователем данных по следующим критериям: заполнение таких полей, как «Pillow name», «Short description», «Description» и «Price» является обязательным. Валидация полей формы реализована с помощью встроенного класса Angular Validators.

Форма добавления нового продукта представлена на рисунке 3.5.



*Рисунок 3.5 – Форма для добавления информации о новом продукте*

При нажатии кнопки «Submit» происходит вызов метода submit() класса-контроллера компонента PillowFormComponent. Реализация метода submit() представлена в листинге 3.3.

*Листинг 3.3 – Реализация метода submit()*

submit() {

if (!this.form.valid) { return; }

const fd = new FormData();

fd.append('pillowName', this.pillowName.value);

fd.append('shortDscrpt', this.shortDscrpt.value);

fd.append('description', this.description.value);

fd.append('fabricStructure', this.fabricStructure.value);

fd.append('filler', this.filler.value);

fd.append('size', this.size.value);

fd.append('price', this.price.value);

if (this.selectedFile) {

fd.append('file', this.selectedFile, this.selectedFile.name);

}

this.pillowService.createPillow(fd).subscribe((data) => {

this.router.navigate(['/admin']);

}, (err) => {

if (err.error) {

if (err.error.status == 403) {

this.authService.logOut();

}

}

});

}

В методе submit() происходит проверка введенных данных. Если все данные валидны, то создается объект класса FormData, в который записывается вся необходимая информация для создания нового продукта. Далее вызывается метод createPillow() сервиса PillowService, который осуществляет POST запрос к API. После завершения операции добавления информации о новом продукте происходит перенаправление на главную страницу панели администрирования. Функция редактирования информации о продукте реализована аналогичным образом.

Удалить информацию о продукте можно выбрав соответствующий продукт из списка и нажав на кнопку «Delete». После чего произойдет вызов метода deletePillow() сервиса PillowService, который осуществит DELETE запрос к API. Код сервиса PillowService представлен в приложении. После обработки запроса пользователь будет перенаправлен на главную страницу панели администрирования.

ЗАКЛЮЧЕНИЕ

Результатом курсового проекта является веб-приложение, представляющее собой интернет-витрину магазина подушек. Веб-приложение разработано с помощью следующих технологий и языков программирования: языки программирования Java 8, TypeScript; фреймворки Spring Boot, Maven и Angular; система управления базами данных PostgreSQL.

В ходе выполнения курсового проекта были проделаны следующие этапы работы:

* Проведен анализ требований к веб-приложению.
* Проведен обзор инструментальных средств и технологий разработки.
* Разработана REST система клиент-серверного взаимодействия.
* Разработана система авторизации пользователя-администратора.
* Спроектирована и подключена база данных, содержащая информацию о продуктах магазина.
* Разработана пользовательская часть веб-приложения на базе фреймворка Angular.
* Разработана панель администрирования веб-приложения, реализующая функции добавления, редактирования и удаления информации о продуктах.

Разработанное веб-приложение содержит:

* Главную страницу, на которой представлен каталог продуктов магазина.
* Страницу, содержащую подробное описание продукта.
* Панель администрирования, реализующую функции добавления, редактирования и удаления информации о продуктах.

Разработанное веб-приложение может быть использовано как площадка для информирования и формирования заинтересованности клиентов к продуктам магазина.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Что такое технология Java и каково ее применение? [Электронный ресурс] – Режим доступа: https://www.java.com/ru/download/help/whatis\_java.html – Дата доступа: 12.12.2020.

2. Introduction to the POM [Электронный ресурс] – Режим доступа: <http://maven.apache.org/guides/introduction/introduction-to-the-pom.html> – Дата доступа: 12.12.2020.

3. JSON Web Token (JWT) [Электронный ресурс] – Режим доступа: https://tools.ietf.org/html/rfc7519– Дата доступа: 14.12.2020.

4. Java JWT: JSON Web Token for Java and Android [Электронный ресурс] – Режим доступа: https://github.com/jwtk/jjwt – Дата доступа: 14.12.2020.

5. What is TypeScript? [Электронный ресурс] – Режим доступа: https://www.typescriptlang.org – Дата доступа: 14.12.2020.

6. Как правильно реализовать ленивую загрузку модулей в Angular 8 [Электронный ресурс] – Режим доступа: https://sales-generator.ru/blog/adaptivnyy-dizayn-sayta/ – Дата доступа: 14.12.2020.

7. Patches – база данных [Электронный ресурс] – Режим доступа: https://oracle-patches.com/db/3517-база-данных-определение – Дата доступа: 03.05.2020.

8. Основы Sass [Электронный ресурс] – Режим доступа: <https://sass-scss.ru/guide/> – Дата доступа: 15.12.2020.

9. Развёрнутое руководство по Sass/SCSS [Электронный ресурс] – Режим доступа: https://tproger.ru/translations/complete-sass-guide/#sass-conclusion – Дата доступа: 15.12.2020.

10. Что такое админка сайта [Электронный ресурс] – Режим доступа: https://semantica.in/blog/chto-takoe-adminka-sajta.html – Дата доступа: 10.05.2020.

11. Live Sass Compiler [Электронный ресурс] – Режим доступа: https://marketplace.visualstudio.com/items?itemName=ritwickdey.live-sass – Дата доступа: 15.12.2020.

12. pgAdmin PostgreSQL Tools [Электронный ресурс] – Режим доступа: https://www.pgadmin.org – Дата доступа: 16.12.2020.

13. Архитектура REST [Электронный ресурс] – Режим доступа: https://habr.com/ru/post/38730/ – Дата доступа: 16.12.2020.

14. Representational State Transfer (REST) [Электронный ресурс] – Режим доступа: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest\_arch\_style.htm – Дата доступа: 18.12.2020.

ПРИЛОЖЕНИЕ

Код файла PillowController.java

package com.pillows.springbootpillowsapi;

import java.io.File;

import java.io.IOException;

import java.util.\*;

import com.pillows.springbootpillowsapi.domain.Pillow;

import com.pillows.springbootpillowsapi.repo.PillowRepo;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

import org.springframework.web.multipart.MultipartFile;

import javax.servlet.http.HttpServletRequest;

@RestController

@RequestMapping("/api/products")

public class PillowController {

@Autowired

private PillowRepo pillowRepo;

@GetMapping("")

public List<Pillow> getPillows ()

{

List<Pillow> pillowsList = pillowRepo.findAll();

return pillowsList;

}

@GetMapping("/{id}")

public Pillow getPillowById (@PathVariable(value = "id") Integer pillowId)

{

Pillow pillow = pillowRepo.findById(pillowId).get();

return pillow;

}

@PostMapping("")

public Pillow createPillow (@RequestParam("pillowName") String pillowName,

@RequestParam("shortDscrpt") String shortDscrpt,

@RequestParam("description") String description,

@RequestParam("cloth") String cloth,

@RequestParam("fabricStructure") String

fabricStructure,

@RequestParam("filler") String filler,

@RequestParam("size") String size,

@RequestParam("price") Double price,

@RequestParam("file") MultipartFile file,

HttpServletRequest request) throws IOException {

Pillow pillow = new Pillow(null, pillowName, shortDscrpt, description, cloth, fabricStructure, filler, size, price, null);

if (file != null && !file.getOriginalFilename().isEmpty()) {

String uploadDirectory = System.getProperty("user.dir") + "/uploads";

File uploadDir = new File(uploadDirectory);

if (!uploadDir.exists()) {

uploadDir.mkdir();

}

String uuidFile = UUID.randomUUID().toString();

String resultFileName = uuidFile + '.' + file.getOriginalFilename();

file.transferTo(new File(uploadDirectory + "/" + resultFileName));

pillow.setFileName(resultFileName);

}

Pillow savedPillow = pillowRepo.save(pillow);

return savedPillow;

}

@PutMapping("/{id}")

public ResponseEntity<Pillow> updatePillow (@PathVariable(value = "id")

Integer pillowId,

@RequestParam("pillowName") String pillowName,

@RequestParam("shortDscrpt") String shortDscrpt,

@RequestParam("description") String description,

@RequestParam("fabricStructure") String fabricStructure,

@RequestParam("filler") String filler,

@RequestParam("size") String size,

@RequestParam("price") String price,

@RequestParam(value = "file", required = false) MultipartFile file,

HttpServletRequest request) throws IOException {

Pillow pillow = pillowRepo.findById(pillowId).get();

pillow.setPillowName(pillowName);

pillow.setShortDscrpt(shortDscrpt);

pillow.setDescription(description);

pillow.setFabricStructure(fabricStructure);

pillow.setFiller(filler);

pillow.setSize(size);

pillow.setPrice(Double.parseDouble((price)));

if (file != null && !file.getOriginalFilename().isEmpty()) {

String uploadDirectory = System.getProperty("user.dir") + "/uploads";

File uploadDir = new File(uploadDirectory);

if (!uploadDir.exists()) {

uploadDir.mkdir();

}

String uuidFile = UUID.randomUUID().toString();

String resultFileName = uuidFile + '.' + file.getOriginalFilename();

file.transferTo(new File(uploadDirectory + "/" + resultFileName));

pillow.setFileName(resultFileName);

}

final Pillow newPillow = pillowRepo.save(pillow);

return ResponseEntity.ok(newPillow);

}

@DeleteMapping("/{id}")

public ResponseEntity<?> deletePillow (@PathVariable(value = "id") Integer pillowId)

{

if (pillowRepo.findById(pillowId).orElse(null) == null) {

return new ResponseEntity<>(HttpStatus.NOT\_MODIFIED);

}

Pillow pillow = pillowRepo.findById(pillowId).get();

pillowRepo.delete(pillow);

return new ResponseEntity<>(HttpStatus.OK);

}

}

Код файла AuthFilter.java

package com.pillows.springbootpillowsapi.filters;

import io.jsonwebtoken.Claims;

import io.jsonwebtoken.Jwts;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.http.HttpStatus;

import org.springframework.web.filter.GenericFilterBean;

import javax.servlet.FilterChain;

import javax.servlet.ServletException;

import javax.servlet.ServletRequest;

import javax.servlet.ServletResponse;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import java.io.IOException;

import java.util.Objects;

public class AuthFilter extends GenericFilterBean {

@Override

public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws IOException, ServletException {

HttpServletRequest httpRequest = (HttpServletRequest) servletRequest;

HttpServletResponse httpResponse = (HttpServletResponse) servletResponse;

String authHeader = httpRequest.getHeader("Authorization");

if(Objects.equals(httpRequest.getMethod(), "GET")) {

filterChain.doFilter(servletRequest, servletResponse);

return;

}

if(authHeader != null) {

String[] authHeaderArr = authHeader.split("Bearer ");

if(authHeaderArr.length > 1 && authHeaderArr[1] != null) {

String token = authHeaderArr[1];

try {

Claims claims = Jwts.parser().setSigningKey("expensetrackerapikey")

.parseClaimsJws(token).getBody();

httpRequest.setAttribute("userName", claims.get("userName").toString());

}catch (Exception e) {

httpResponse.sendError(HttpStatus.FORBIDDEN.value(), "invalid/expired token");

return;

}

} else {

httpResponse.sendError(HttpStatus.FORBIDDEN.value(), "Authorization token must be Bearer [token]");

return;

}

} else {

httpResponse.sendError(HttpStatus.FORBIDDEN.value(), "Authorization token must be provided");

return;

}

filterChain.doFilter(servletRequest, servletResponse);

}

}

Код файла pillow.service.ts

import { Injectable } from '@angular/core';

import { environment } from 'src/environments/environment';

import { HttpClient, HttpHeaders } from '@angular/common/http';

import { Observable } from 'rxjs';

import { Pillow } from 'src/app/shared/models/pillow.model';

import { AuthService } from 'src/app/auth/services/auth.service';

@Injectable({

providedIn: 'root'

})

export class PillowService {

public baseUrl = environment.apiUrl;

public pillowsList: Pillow[];

constructor(private http: HttpClient, private authService: AuthService) { }

getPillowsList(): Observable<Pillow[]> {

return this.http.get<Pillow[]>(this.baseUrl);

}

getPillow(id: string): Observable<Pillow> {

return this.http.get<Pillow>(`${this.baseUrl}/${id}`);

}

deletePillow(id: number): Observable<{}> {

const headers = new HttpHeaders({ 'authorization': `Bearer ${this.authService.getToken()}` });

return this.http.delete(`${this.baseUrl}/${id}`, { headers: headers });

}

updatePillow(fd: FormData, id: number): any {

const headers = new HttpHeaders({ 'authorization': `Bearer ${this.authService.getToken()}` });

return this.http.put<Pillow>(`${this.baseUrl}/${id}`, fd, { headers: headers });

}

createPillow(fd: FormData): any {

const headers = new HttpHeaders({ 'authorization': `Bearer ${this.authService.getToken()}` });

return this.http.post<Pillow>(`${this.baseUrl}`, fd, { headers: headers });

}

}