|  |  |  |
| --- | --- | --- |
|  |  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |

Институт Информационных технологий

Кафедра Инструментального и прикладного программного обеспечения

**ПРАКТИЧЕСКАЯ РАБОТА № 6**

по дисциплине «Разработка серверных частей интернет-ресурсов»

**Тема практической работы:** Чистая архитектура, MVC. Создание приложения с использованием фреймворка Spring

|  |  |
| --- | --- |
| Студент группы ИКБО-32-21 | Быченков А.К. |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись студента)

|  |  |
| --- | --- |
| **Руководитель практической работы** | преподаватель Волков М.Ю. |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись руководителя)

Работа представлена «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2023 г.

Допущен к работе «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2023 г.

Москва 2023

**Цель работы:** получить навыки проектирования и разработки приложений с использованием фреймворка Spring.

**Задание**

Студенту предлагается реализовать бизнес логику ко второй практике по Java. Нужно добавить к существующему приложению реализацию «корзины».

Пользователь должен иметь такие возможности:

1. Добавить товар в корзину.
2. Удалить товар из корзины.
3. Изменить количество товара в корзине.
4. Посмотреть всю корзину.
5. Оформить заказ и очистить корзину.

При добавлении товара в корзину должна быть проверка, что товар есть в наличии. Так же обработать случай, что товар добавили в корзину, после товар закончился на складе и оформить заказ невозможно.

Сервис должен с помощью docker-compose. Должно быть использовано паттерны проектирования: Чистая архитектура, MVC.

**Ход работы**

В соответствии с принципами Чистой архитектуры была разработана структура приложения для обеспечения независимости отдельных слоев приложения от внешней инфраструктуры (фреймворков, различных баз данных, API и т.д), рисунок 1.

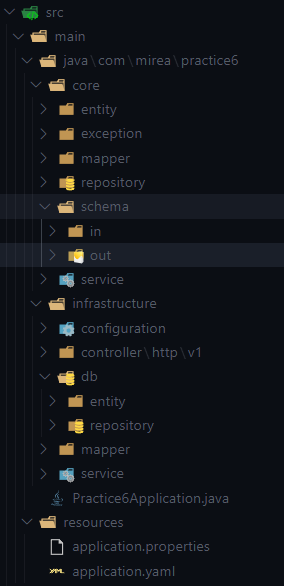
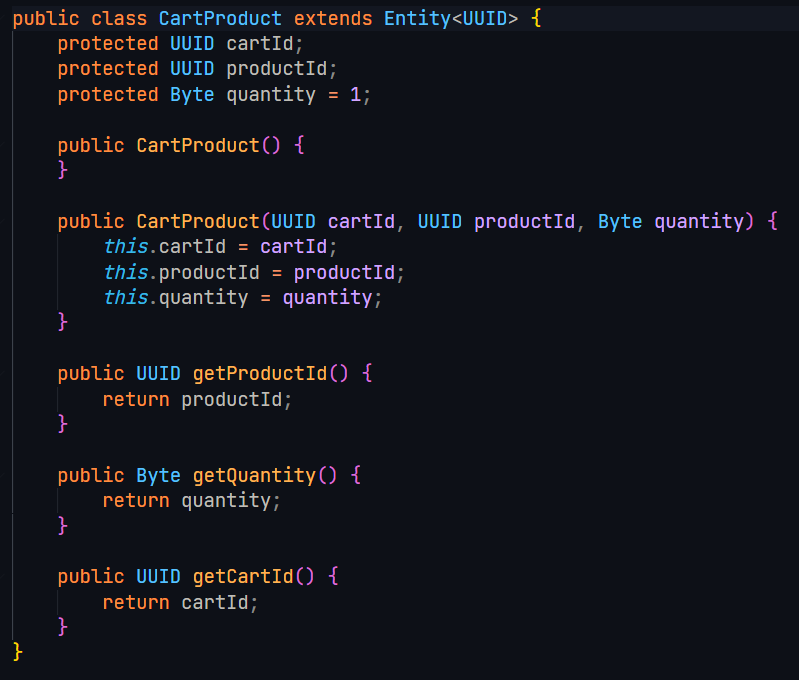
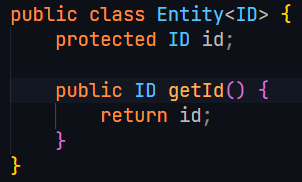
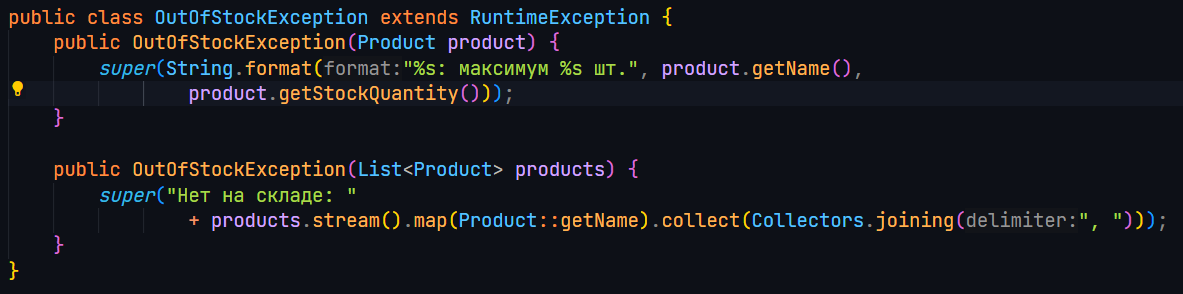
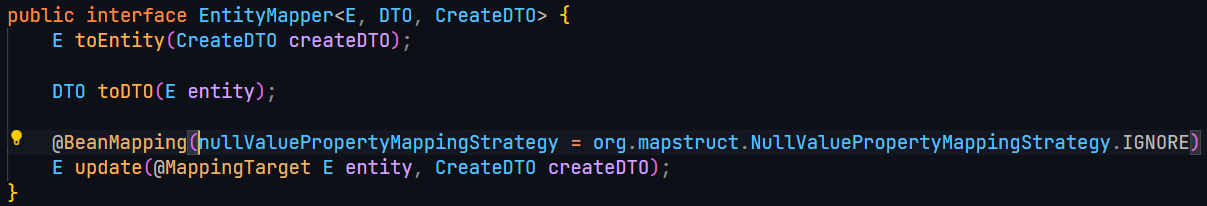
****

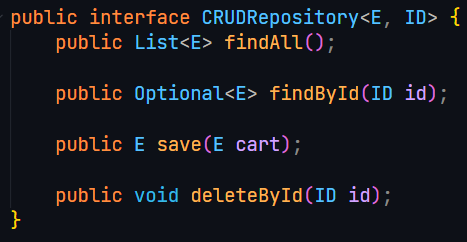
Рисунок 1 – Структура приложения

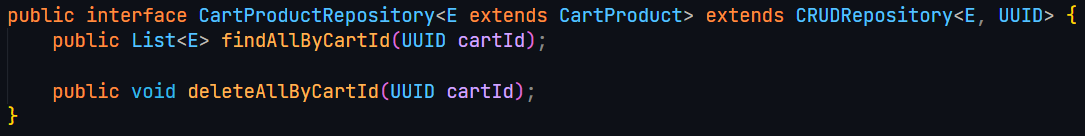


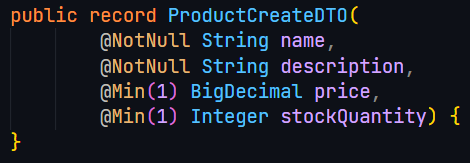


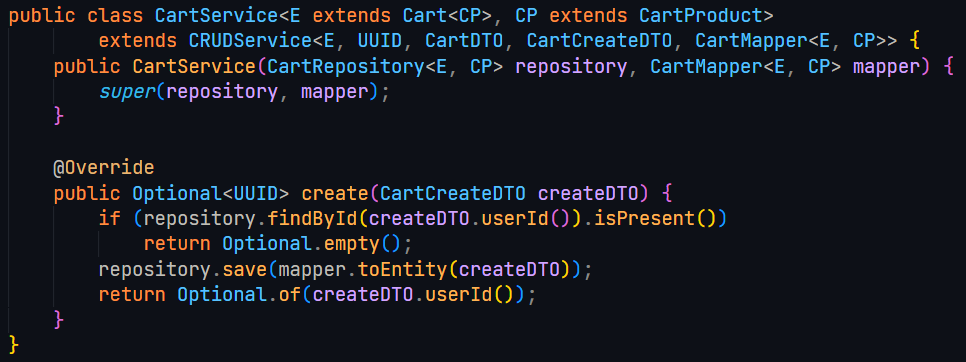






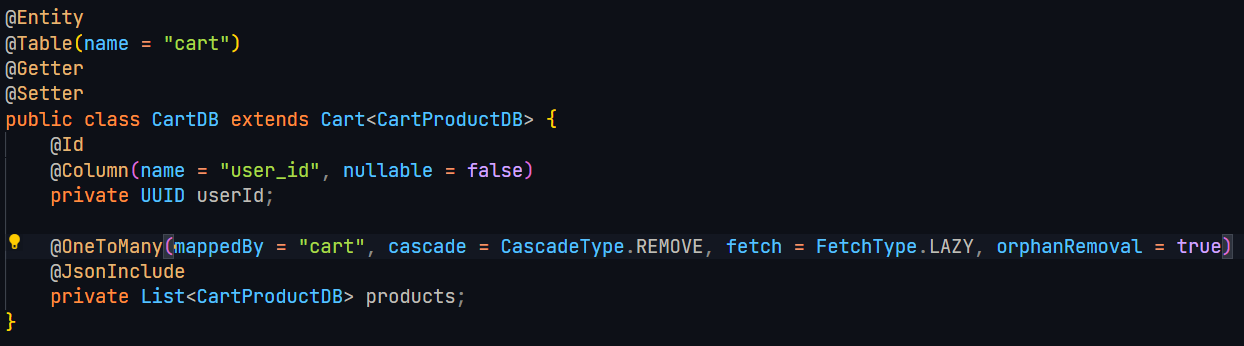


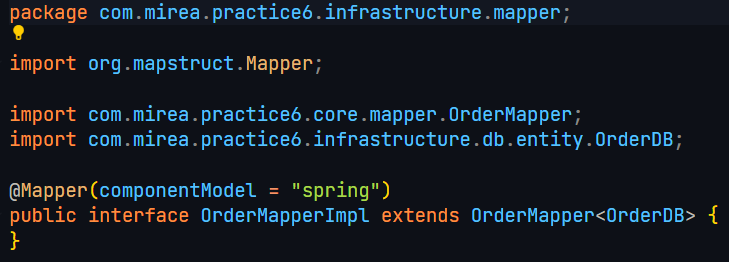




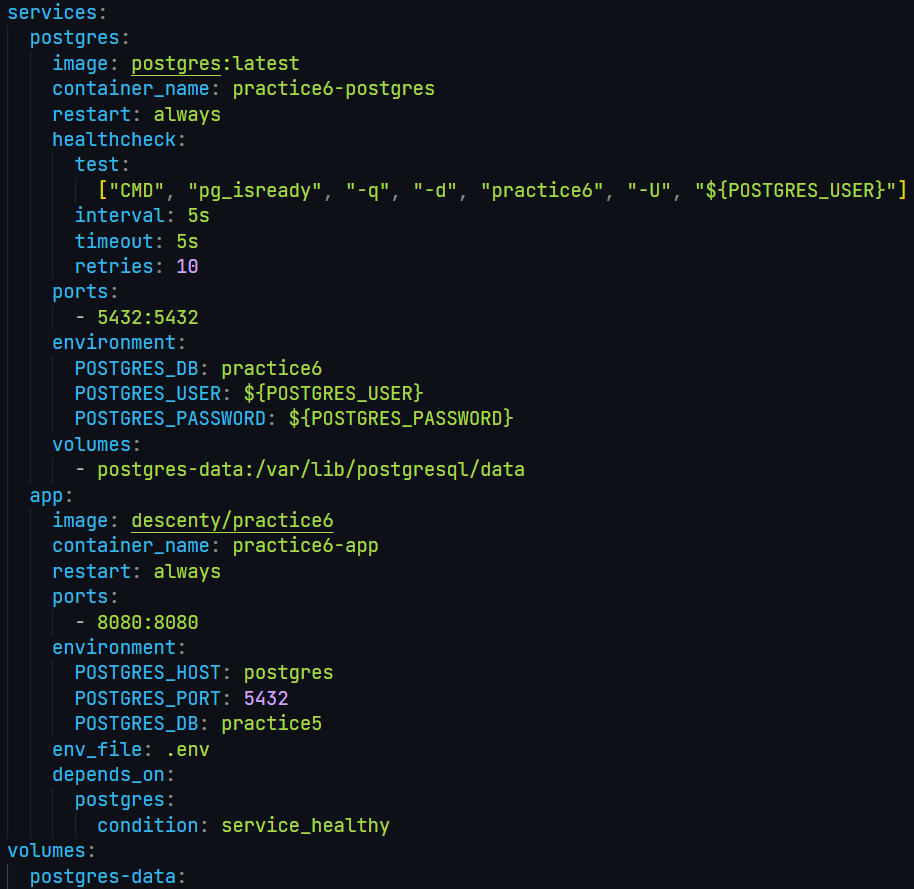












Пример ответа приложения для получения корзины пользователя показан на рисунке X.

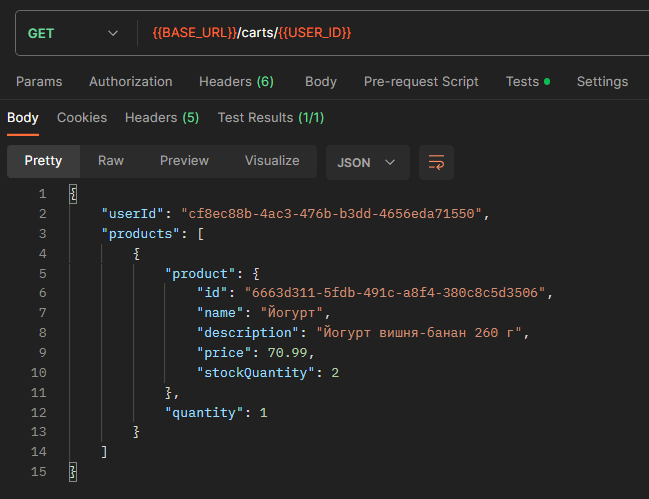


Рисунок X – Получение корзины пользователя

Было проведено автоматизированное тестирование API. Написан тестовый сценарий, который приложение успешно проходит, рисунок Y.

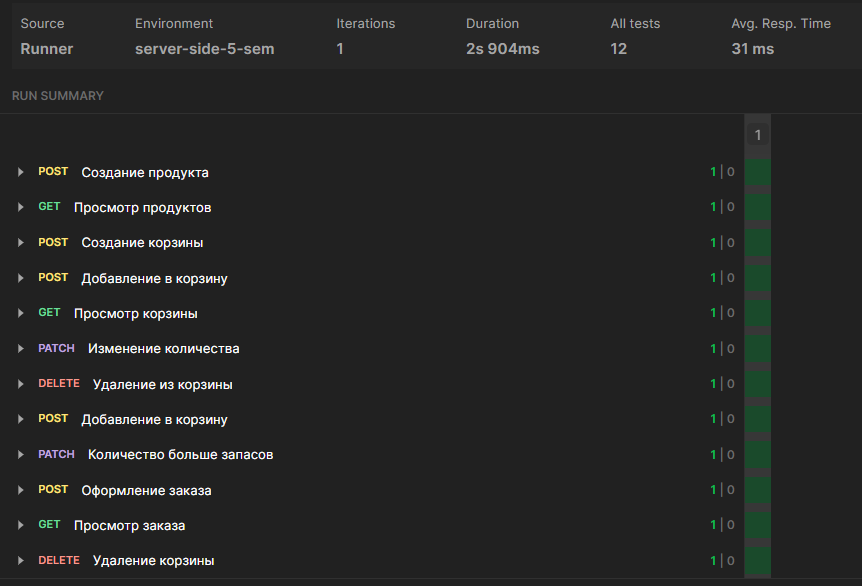


Рисунок Y – Пройденный сценарий использования

**Вывод**

В ходе выполнения практической работы были получены навыки создания Spring-приложений, используя паттерны «Чистая архитектура» и MVC.

**Ответы на вопросы**

1. **Аннотация Service**

Когда класс помечен аннотацией @Service, Spring IoC (Inversion of Control) контейнер обнаруживает этот класс во время компоновки (контекста приложения) и автоматически создает экземпляр этого класса, инъектирует его в другие компоненты и управляет его жизненным циклом.

Аннотация @Service показывает, что класс представляет собой сервис для реализации бизнес-логики. По сути, аннотация не отличается от Component, однако она помогает указать смысловую нагрузку используемого класса.

1. **Паттерн MVC**

Cхема разделения данных приложения и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо. Модель предоставляет данные и реагирует на команды контроллера, изменяя своё состояние.

Под Моделью, обычно понимается часть, содержащая в себе функциональную бизнес-логику приложения. Модель должна быть полностью независима от остальных частей продукта. Модельный слой ничего не должен знать об элементах дизайна, и каким образом он будет отображаться. Достигается результат, позволяющий менять представление данных, то как они отображаются, не трогая саму Модель. Модель обладает следующими признаками:

* Модель — это бизнес-логика приложения;
* Модель обладает знаниями о себе самой и не знает о контроллерах и представлениях;
* Для некоторых проектов модель — это просто слой данных (DAO, база данных, XML-файл);
* Для других проектов модель — это менеджер базы данных, набор объектов или просто логика приложения;

В обязанности Представления входит отображение данных, полученных от Модели. Однако, представление не может напрямую влиять на модель. Можно говорить, что представление обладает доступом «только на чтение» к данным. Представление обладает следующими признаками:

* В представлении реализуется отображение данных, которые получаются от модели любым способом;
* В некоторых случаях, представление может иметь код, который реализует некоторую бизнес-логику.

Контроллер обеспечивает «связь» между пользователем и системой. Контролирует и направляет данные от пользователя к системе и наоборот. Использует модель и представление для реализации необходимого действия.

Контроллер перехватывает событие извне и в соответствии с заложенной в него логикой, реагирует на это событие изменяя Mодель, посредством вызова соответствующего метода. После изменения Модель использует событие о том что она изменилась, и все подписанные на это события Представления, получив его, обращаются к Модели за обновленными данными, после чего их и отображают.

1. **Паттерн «Чистая архитектура»**

Ключевой принцип Clean Architecture - разделение приложения на уровни, каждый из которых выполняет свои задачи и управляет своей ответственностью. Обычно такое разделение выглядит следующим образом:

* Уровень представления
* Уровень приложения
* Уровень домена
* Уровень инфраструктуры

Уровень представления отвечает за взаимодействие с пользователем и обработку запросов. Уровень приложения выполняет бизнес-логику и координирует работу между уровнями представления и домена. Уровень домена содержит бизнес-логику и компоненты, отвечающие за работу с данными. Уровень инфраструктуры занимается поддержкой структур приложения и связью с внешними системами (например, базами данных, API и т.д.).

Одна из важных частей этого принципа - зависимости внутри уровней. В каждом уровне приложения (например, Presentation, Domain или Data) важно тщательно контролировать зависимости между компонентами. Например, компоненты внутри слоя Domain не должны зависеть от компонентов в слое Data.

Один из способов достичь этого - использование инверсии зависимостей (Dependency Inversion Principle, DIP). Вместо того, чтобы компоненты в верхнем слое зависели от компонентов в нижнем слое, управление зависимостями и обмен данными происходит через общий интерфейс. Это позволяет упростить добавление, удаление или изменение компонентов с минимальными изменениями внутри каждого слоя.

Граничные интерфейсы (Boundary Interfaces) – это интерфейсы, которые разделяют используемые элементы на две области: внутри приложения и вне его. Они служат для определения, какие элементы способны перейти за границу приложения и какие нет.

Граничные интерфейсы играют важную роль в Clean Architecture, так как они определяют, как пользовательский интерфейс должен общаться с приложением. Пользовательский интерфейс является внешней частью приложения, которая взаимодействует с пользователем и передает запросы в приложение. Граничные интерфейсы определяют, какие запросы пользовательский интерфейс может отправлять в приложение, и какие данные он может получать в ответ.

Примером граничного интерфейса является REST API веб-сервера, который позволяет клиентской стороне общаться с серверной стороной приложения. Если слой приложения использует граничные интерфейсы, то он был бы независим от клиентской стороны и мог бы быть более гибким при изменениях в клиентской стороне.

1. **Описать все слои RestFull приложения и описать зачем каждый из них**

RESTful приложение, в соответствии с архитектурным стилем REST (Representational State Transfer), обычно состоит из различных слоев, каждый из которых выполняет определенные функции. Ниже описаны основные слои RESTful приложения и их назначение:

Слой ресурсов (Resource Layer)

Назначение: Этот слой определяет ресурсы (или эндпойнты) вашего приложения, которые клиенты могут получить, изменить, создать или удалить. Ресурсы представляют собой конкретные объекты или данные, например, пользователей, продукты, заказы.

Роль: Слой ресурсов определяет URL-адреса (URI) и методы HTTP, связанные с каждым ресурсом, и обрабатывает запросы от клиентов.

Слой бизнес-логики (Business Logic Layer)

Назначение: Этот слой содержит бизнес-логику приложения, которая определяет, как взаимодействовать с данными и выполнять операции над ресурсами. В этом слое обрабатываются запросы из слоя ресурсов, выполняются бизнес-правила и происходит взаимодействие с хранилищем данных.

Роль: Этот слой обеспечивает высокоуровневую логику приложения, не зависящую от специфики HTTP-запросов и ответов.

Слой доступа к данным (Data Access Layer)

Назначение: Этот слой отвечает за взаимодействие с хранилищем данных, таким как база данных. Здесь выполняются операции чтения и записи данных, а также преобразование данных из и в формат, понятный бизнес-логике и слою ресурсов.

Роль: Этот слой скрывает детали взаимодействия с конкретным источником данных, обеспечивая более высокоуровневой слоям удобный интерфейс для работы с данными.

Слой представления (Presentation Layer)

Назначение: Этот слой отвечает за представление данных в формате, который может быть понятен клиентам. Он генерирует HTTP-ответы, обычно в формате JSON или XML, и управляет форматированием данных и обработкой ошибок.

Роль: Слой представления преобразует данные из бизнес-логики в формат, который клиенты могут понять, и отправляет их клиентам в ответ на запросы.

Слой безопасности (Security Layer):

Назначение: Этот слой обеспечивает безопасность RESTful приложения, включая аутентификацию и авторизацию пользователей, контроль доступа, обработку токенов и защиту от угроз безопасности.

Роль: гарантирует, что только уполномоченные пользователи имеют доступ к определенным ресурсам, и обеспечивает конфиденциальность и целостность данных.

**Ссылка на удаленный репозиторий проекта**

<https://github.com/descenty/server-side-5-sem>

**Список использованной литературы**

* 1. Статья про REST приложения – <https://habr.com/ru/articles/471140>
  2. Статья про Чистую архитектуру – <https://habr.com/ru/articles/269589>
  3. Паттерны MVC, MVP, MVVM – <https://habr.com/ru/articles/215605>
  4. CRUD приложение на Java – <https://ru.hexlet.io/courses/java-web/lessons/crud-in-db/theory_unit>
  5. Статья про Spring Data JPA – <https://habr.com/ru/articles/435114>