Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

**Курсовой проект**

**Проектирование системы магазина консолей, видеоигр и игровых аксессуаров**

**Дисциплина**: Программное обеспечение распределенных вычислительных систем

Выполнил студент гр. 3540901/81502 Бурсиков Л. Д.

(подпись)

Руководитель Стручков И.В.

(подпись)

“\_\_\_\_” \_\_\_\_\_\_\_\_\_\_\_ 2019 г.

Санкт – Петербург

2019

Содержание

[1.Анализ задания 3](#_Toc29839049)

[1.1Описание бизнес процессов: 3](#_Toc29839050)

[1.2. Варианты использования 5](#_Toc29839051)

[1.3. Моделирование 9](#_Toc29839052)

[2.Разработка 16](#_Toc29839053)

[2.2 Интерфейс веб-приложения 19](#_Toc29839054)

[2.3 Тестирование 20](#_Toc29839055)

[2.4 Инструкция системного администратора 20](#_Toc29839056)

[2.5 Инструкция пользователя 21](#_Toc29839057)

[3. Выводы 21](#_Toc29839058)

[Приложение. Исходный код 22](#_Toc29839059)

# 1.Анализ задания

Название проектируемой системы: «Магазин консолей, видеоигр и игровых аксессуаров»

Назначение:

Розничная продажа консолей, видеоигр (с возможностью предварительного заказа), игровых аксессуаров (контроллеры, наушники и др.) и сувенирной продукции по тематике. Так же существует программа Trade-In – покупатель может продать б/у игры, и получить за них деньги, другую игру по такой же цене, скидку на приобретение другого товара.

Участники:

1. Клиент – совершает покупки, участие в программе Trade-In.
2. Менеджер – совершает продажу товара, осуществляет непосредственное взаимодействие с клиентом.
3. Администратор – работает с внутренними данными внутри системы, косвенная работа с клиентами.

## 1.1Описание бизнес процессов:

1)Покупка товара

Участники: Клиент, менеджер, администратор.

Этапы:

* Клиент выбирает товар(товары) на сайте.
* Клиент заполняет форму на сайте, указывая контактные данные, и выбирает способ оплаты – онлайн, или на месте при получении товара.
* В случае оплаты онлайн клиент вводит платежную информацию, и получает чек об оплаченной покупке.
* Администратор просматривает информацию о заказе, проверяет наличие товара, передает менеджеру информацию для связи с клиентом.
* Менеджер связывается с клиентом по указанным контактным данным, уточняет дату и время, когда клиент приедет за товаром.
* Клиент приезжает в магазин за товаром, оплачивает покупку (если оплата проводилась онлайн – предоставляет полученный чек), забирает товар.

Менеджер закрывает заказ.

Сущности:

1. Товар
2. Заказ
3. Чек
4. Контактная информация

2) Обмен

Участники: Клиент, менеджер, администратор.

Этапы:

* Клиент приходит с игрой, которую хочет обменять
* Менеджер проверяет, что игра находится в списке программы Trade-In, передает её администратору.
* Администратор определяет категорию (от категории зависит цена игры), проверяет работоспособность.
* Клиент выбирает другую игру из той же категории.
* Менеджер выдает игру клиенту.
* Администратор вносит изменения в базу данных, касательно наличия игр по программе.

Сущности:

1. Игра на обмен
2. Категории игр
3. Заказ

## 1.2. Варианты использования

Ниже приведены диаграммы вариантов использования для каждой роли, выделенной в системе.

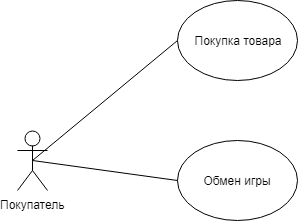


Рисунок 1. Диаграмма вариантов использования для роли "Клиент"

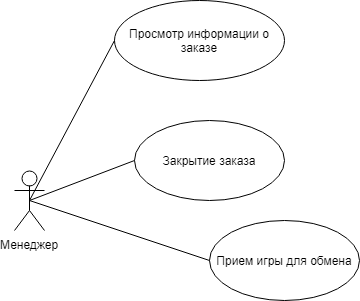


Рисунок 2. Диаграмма вариантов использования для роли "Менеджер"

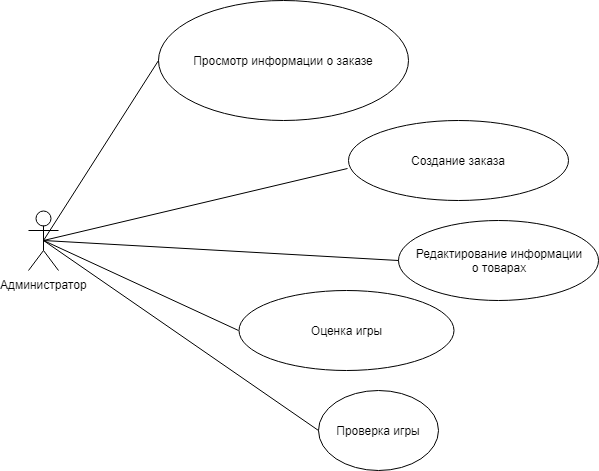


Рисунок 3. Диаграмма вариантов использования для роли "Администратор"

Текстовое описание вариантов использования

1)Клиент

1.1)Покупка товара

1. Клиент просматривает каталог и выбирает товар
2. Клиент вводит контактную информацию
3. Клиент выбирает способ оплаты (онлайн или при получении)
4. Клиент получает чек об оплате
5. Клиент забирает товар из магазина, называя номер заказа забирает товар

Альтернатива: Если клиент выбирает оплату при получении, при получении товара клиентоплачивает товар, перед тем, как забрать.

1.2) Обмен игры

1. Клиент приносит игру на обмен в магазин и отдает менеджеру
2. Клиент выбирает и забирает другую игру из предложенной категории

Альтернатива: Если клиент не устраивает оценка принесенной игры, он забирает её обратно

2) Менеджер

2.1) Просмотр информации о заказе

1. Просмотр контактной информации клиента для связи с ним, касательно заказа
2. Просмотр информации об оплате и составе заказа, при выдаче товара клиенту

2.2) Закрытие заказа

1. Менеджер закрывает заказ в системе, после выдачи товара клиенту

2.3) Прием игры для обмена

1. Менеджер принимает игру у клиента
2. Менеджер передает игру администратору для проверки и оценки
3. Менеджер выдает клиенту другую игру, из той же категории

Альтернатива: если игра не запустилась у администратора – менеджер возвращает её клиенту

3) Администратор

3.1)Просмотр информации о заказе

1. Администратор просматривает информацию о заказе, чтобы проверить наличие товара и передать контактную информацию менеджеру

3.2) Создание заказа

1. Администратор создает в системе запись о заказе

3.3) Редактирование информации о товарах

1. Администратор добавляет товары в каталог, изменяет информацию об их количестве и цене, удаляет товары из каталога

3.4) Проверка игры

1. Администратор проверяет работоспособность принесенной клиентом игры В случае, если она не работает, возвращает клиенту

3.5) Оценка игры

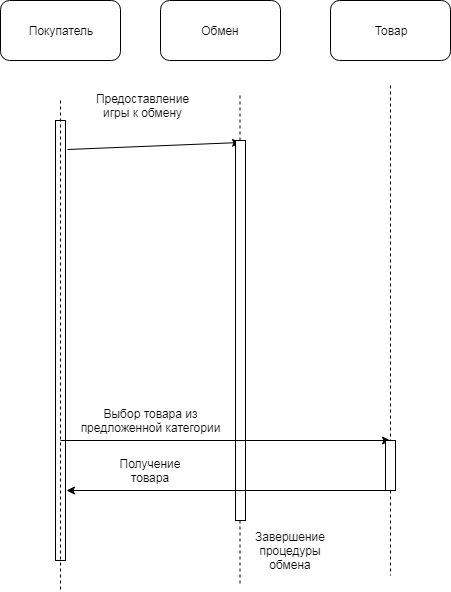
1. Администратор оценивает игру, присваивая её к одной из категорий
2. Администраторзаносит информацию о принятой игре в систему

## 1.3. Моделирование

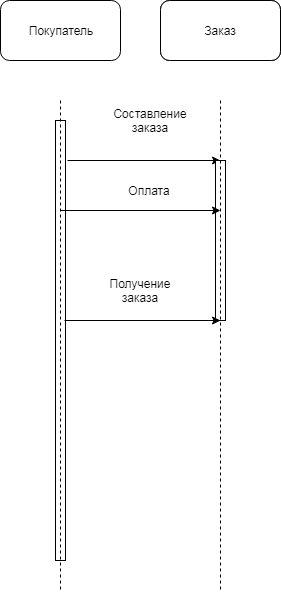
В этом разделе представлены диаграммы последовательностей для каждой из ролей и диаграмма классов, описывающая сущности существующие в системе.

Диаграммы для роли "Клиент"

Обмен

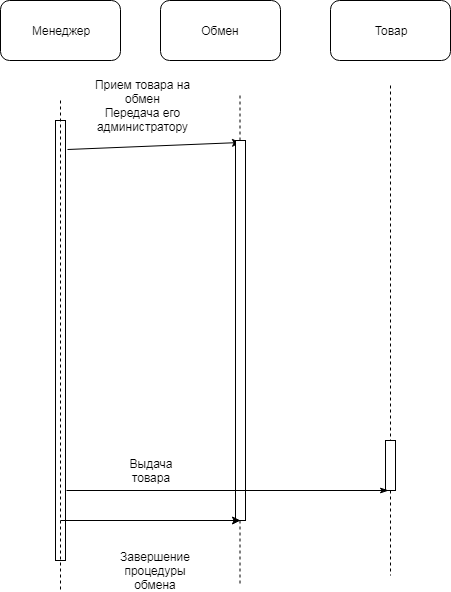


Покупка

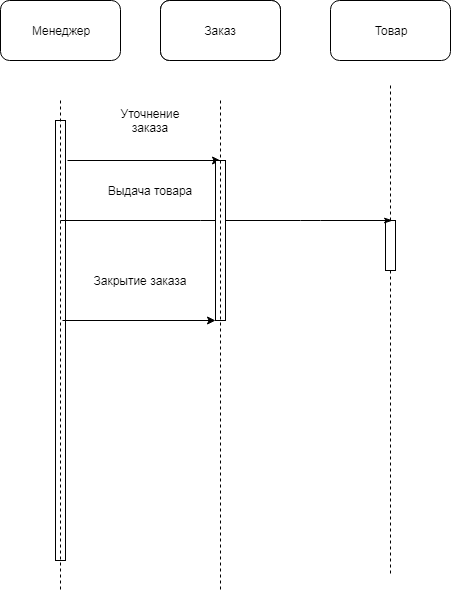


Диаграммы для роли "Менеджер"

Обмен

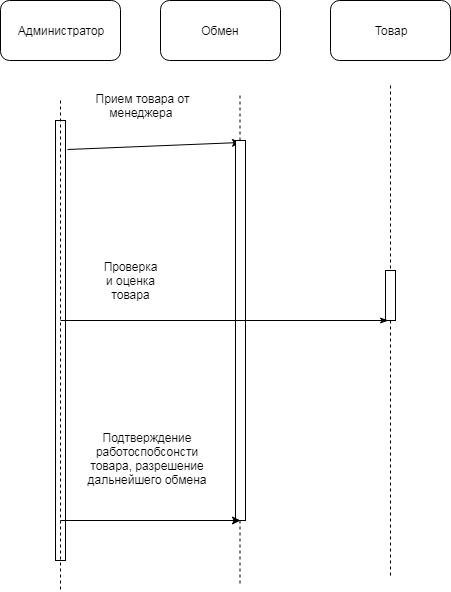


Покупка

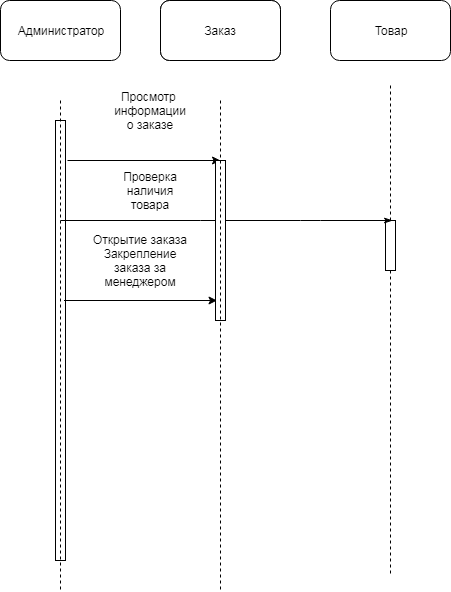


Диаграммы для роли "Администратор"

Обмен



Покупка



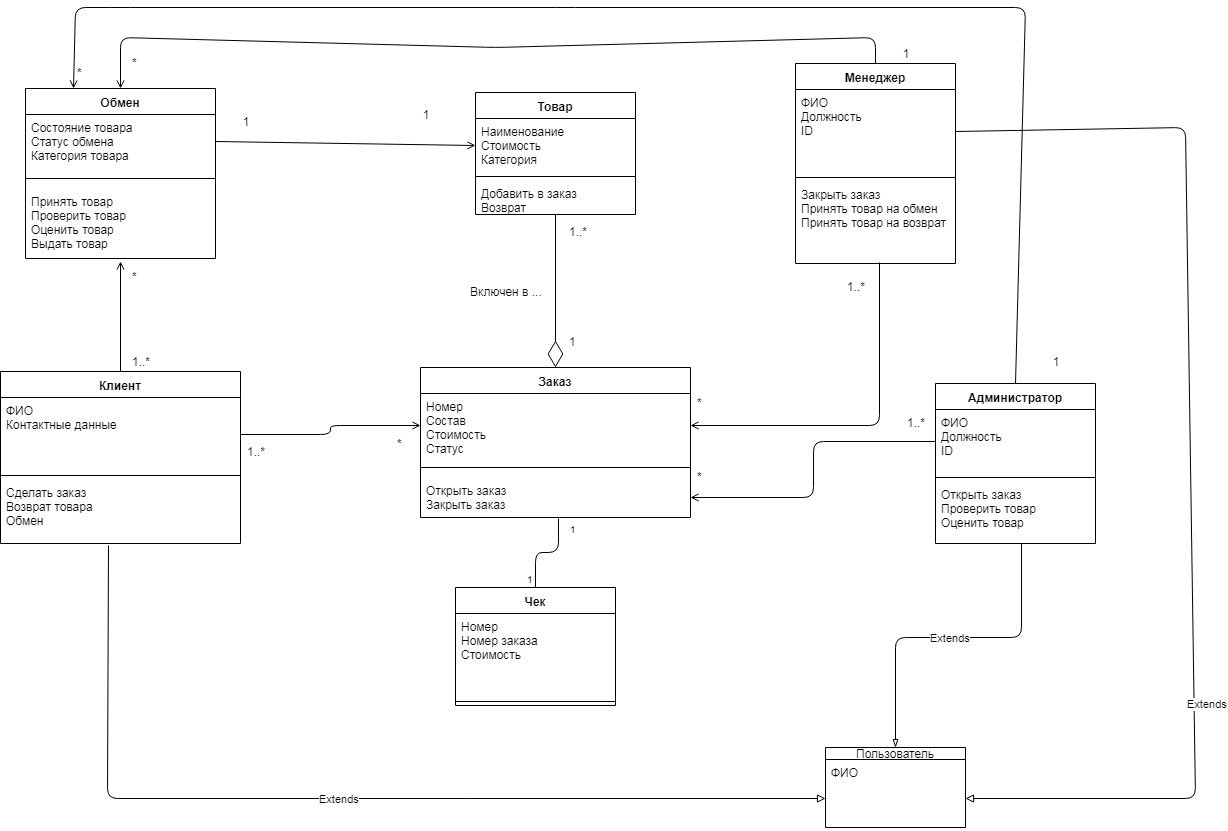


Рисунок .Cтатическая модель предметной области

# 2.Разработка

## 2.1 Проектирование

Для реализации системы используется Spring Framework (в частности компонент Spring Boot) и для реализации пользовательского интерфейса в виде Single Page Application используется JavaScript и с фреймворком Vue.js. В качестве сервера базы данных используется MySQL.

Сущностные классы повторяют представленные в UML-диаграмме выше классы, за исключением двух классов «order\_product» и «OrderProductPK» которые необходимы для реализации отношения «многие ко многим» (т.к. в одном заказе могут быть несколько товаров, и в разных заказах могут быть одинаковые товары). Ниже приведен код этих двух классов:  
 Класс «order\_product»

@Entity  
@Table(name = "order\_products")  
public class order\_product implements Serializable {  
 @EmbeddedId  
 @JsonIgnore  
 private OrderProductPK pk;  
  
 public order\_product(){  
 super();  
 }  
 public order\_product(Order order\_id, Product product\_id, int id)  
 {  
  
 this.pk = new OrderProductPK();  
 pk.setOrder(order\_id);  
 pk.setProduct(product\_id);  
 }  
 public OrderProductPK getPk() {  
 return pk;  
 }  
  
 public void setPk(OrderProductPK pk) {  
 this.pk = pk;  
 }  
 @Transient  
 public Product getProduct() {  
 return this.pk.getProduct();  
 }  
  
 @Transient  
 public float PriceOfOrder() {  
 return getProduct().getProductPrice();  
 }  
}

Класс «OrderProductPK»

@Embeddable  
public class OrderProductPK implements Serializable {  
  
 @JsonBackReference  
 @ManyToOne(fetch = FetchType.*LAZY*)  
 @NotFound(action = NotFoundAction.*IGNORE*)  
 @JoinColumn(name = "order\_id")  
 private Order order;  
  
 @ManyToOne(fetch = FetchType.*LAZY*)  
 @NotFound(action = NotFoundAction.*IGNORE*)  
 @JoinColumn(name = "product\_id")  
 private Product product;  
  
 OrderProductPK(){}  
 public void setProduct(Product product) {  
 this.product = product;  
 }  
  
 public Product getProduct() {  
 return product;  
 }  
  
 public void setOrder(Order order)  
 {  
 this.order= order;  
 }  
  
 public Order getOrder() {  
 return order;  
 }  
}

Аннотация «@EmbeddedId» в классе «order\_product» около поля pk, говорит о том, что объект этого класса по сути выступает ключом, необходимым при реализации отношения «многие ко многим». Так же класс «OrderProductPK» определяет какие поля в классах «product» и «order» являются внешними ключами.

Из функциональности описанной выше в рамках данной работы реализован только вариант использования «Заказ товара» для роли «Клиент».

В соответствии с этим реализованы классы-репозитории и сервисы необходимые для работоспособности (репозитории и сервисы заказов, товаров, клиентов, и сборной сущности «заказ-товар»).

Также реализовано 4 класса-контроллера MainController – являющийся точкой входа, и обрабатывающий запросы на вход; ClientController – обрабатывает запросы, связанные с получением данных о пользователе;

ProductController – обрабатывает запросы связанные с получением данных о товарах; OrderController – обрабатывает запросы связанные с получением данных о заказах и созданием новых заказов.

Контоллеры используют аннотации «GetMapping», «PostMapping» и «и RequestMapping» для разрешения запросов от фронтэнда.

Пример аннотации «RequestMapping» для функции logInUser

@RequestMapping(value = "/login", method = RequestMethod.*GET*,  
 produces = MediaType.*APPLICATION\_JSON\_VALUE*)  
@ResponseBody  
public ResponseEntity<Integer> logInUser(@RequestParam String login, @RequestParam String password) {  
 if (null == login) {  
 return new ResponseEntity<>(HttpStatus.*BAD\_REQUEST*);  
 }  
 try {  
 Client client = clientService.findClient1(login, password);  
 if (client != null)  
 return new ResponseEntity<>(3, HttpStatus.*OK*);  
 else return new ResponseEntity<>(HttpStatus.*NOT\_FOUND*);  
 } catch (NoSuchElementException e) {  
 return new ResponseEntity<>(HttpStatus.*NOT\_FOUND*);  
 }

Если при обработке запросах на обновление/добавление данных от фронтенда поступают данные, которые не подходят по своей структуре на существующие сущностные классы и нет четкого разделения по параметрам (использование аннотации «RequestParam»), могут понадобиться «оберточные» классы как например внутренний класс «OrderWrapper» внутри класса «OrderController»:

public static class OrderWrapper{  
  
 @JsonProperty public int client\_id;  
 @JsonProperty public Float cost;  
 @JsonProperty List<Integer> list;  
 public OrderWrapper(){}  
 public OrderWrapper (int client\_id, Float cost, List<Integer> productsoforder)  
 {  
 this.client\_id=client\_id;  
 this.cost=cost;  
 setproductsoforder(productsoforder);  
 }  
 public List<Integer> getproductsoforder() {  
 return list;  
 }  
  
 public void setproductsoforder(List<Integer> productsoforder) {  
 this.list = productsoforder;  
 }  
  
  
}

Т.к. от фронтенда при создании заказа поступают только данные о id клиента, цене и составе заказа (в виде id товаров), было решено, что проще использовать оберточный класс для принятия данных от фронтэнда, и последующего их использования в создании объектов сущностей для сохранения в базе данных.

## 2.2 Интерфейс веб-приложения

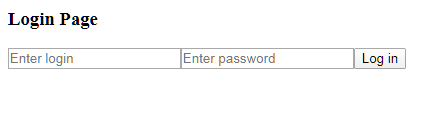


Рисунок . Окно входа

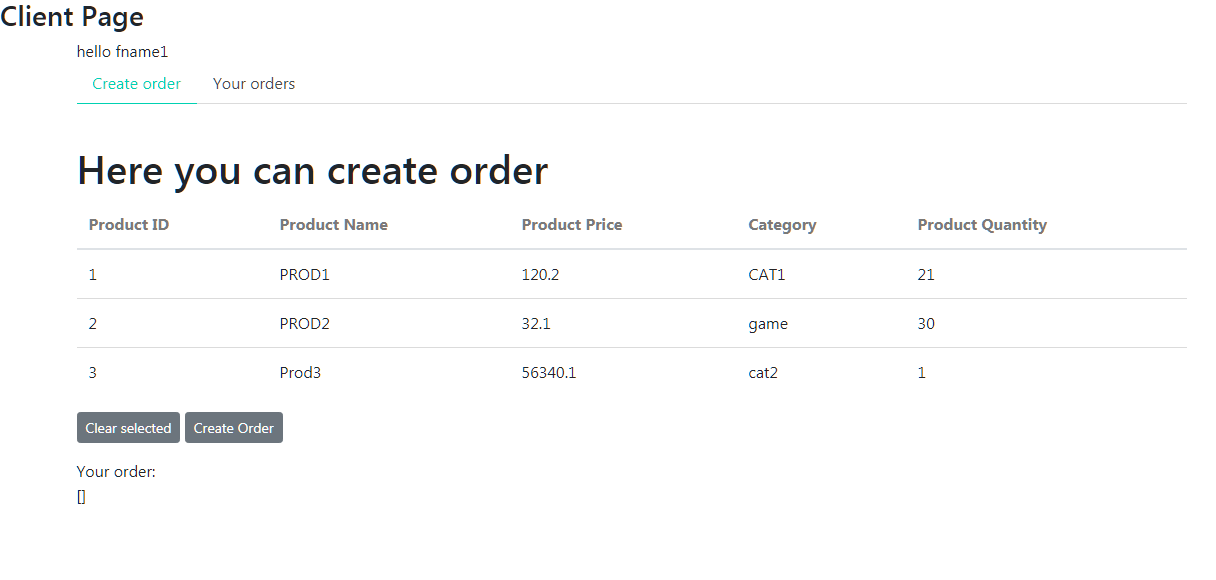


Рисунок . Страница клиента, панель создания заказа.

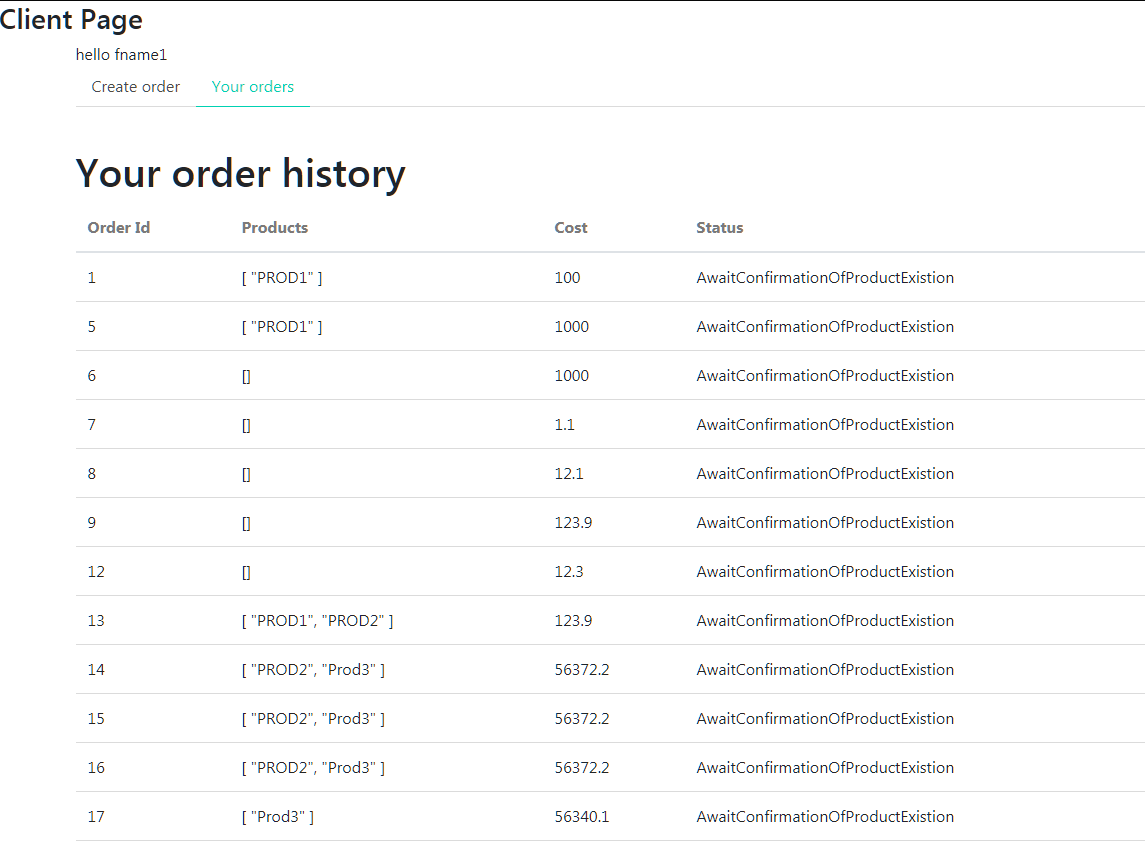


Рисунок . Страница клиента, панель просмотра заказов клиента

## 2.3 Тестирование

Для проверки работы приложения было использовано ручное тестирование (в виду малого количество реализованных вариантов использования). В ходе тестирование ошибок не обнаружено.

## 2.4 Инструкция системного администратора

Для обеспечения работы приложения необходимо :

1)MySQL Server версии 8 (или выше). Так же необходимо убедиться что доступ к базе данных имеется по адресу «jdbc:mysql://${MYSQL\_HOST:localhost}:3306/shop»

2)WebServer Tomcat версии 9 (или выше)

3)Убедиться что есть доступ к порту 3333 извне (на этом порте работает приложение)

## 2.5 Инструкция пользователя

Работа начинается с открытия страницы логина, где необходимо ввести email и пароль, которые существуют в базе данных. После чего открывается страница на которой предоставляется возможность выбрать один или несколько товаров из списка, после чего необходимо нажать кнопку «Create order» для создания заказа. На вкладке «Your orders» отображены все заказы зашедшего пользователя. Для просмотра только что созданного заказа необходимо обновить страницу.

# 3. Выводы

Задание выполнено в минимально возможном объеме. Необходимо реализовать остальные варианты использования, но в идеале – создать приложение с нуля, используя опыт полученный при текущей разработке, т.к. некоторые аспекты можно было бы реализовать лучше, чем они есть в текущем виде, а так же возможно стоит пересмотреть унаследованную модель предметной области.

Итоговое приложение получилось прозрачным и открытым (в основном за счет использования Java-платформы) но слабо масштабируемым.

# Приложение. Исходный код

Класс «Client»

@Entity  
@Table(name = "clients", schema = "shop")  
public class Client extends User   
{  
 @Column(name = "telenumber")  
 protected String TeleNumber;  
 public Client() {}  
 public Client (String FirstName, String SurName,int id, String TeleNumber, String Email, String password)  
 {   
 super(FirstName,SurName,id,password, Email);  
 this.TeleNumber = TeleNumber;  
 }  
 public String getTNumber()  
 {  
 return TeleNumber;  
 }  
 public void setTNumber(String TNumber)  
 {  
 this.TeleNumber = TNumber;  
 }  
  
}

Класс «Order»

package courseWork.logic;  
import java.util.ArrayList;  
import java.util.List;  
  
import courseWork.utils.Order\_status;  
import org.hibernate.annotations.Fetch;  
import org.hibernate.annotations.FetchMode;  
  
import javax.persistence.\*;  
  
  
@Entity  
@Table(name = "orders", schema = "shop")  
public class Order   
{  
 @Id  
 @Column(name = "order\_id")  
 @GeneratedValue  
 private int id;  
 @Column(name = "client\_id")  
 private int client\_id;  
 @OneToMany(mappedBy = "pk.order",cascade = CascadeType.*ALL*)  
 @Fetch(FetchMode.*JOIN*)  
 private List<order\_product> Product = new ArrayList<order\_product>();  
 @Column(name = "order\_status")  
 @Enumerated(EnumType.*STRING*)  
 private Order\_status Status;  
 private double cost;  
 public Order (){}  
 public Order (int id,int client\_id, String status, List<order\_product> list){  
 this.client\_id=client\_id;  
 this.id=id;  
 this.Product = list;  
 this.Status = Order\_status.*valueOf*(status);  
 }  
 public Order (int id, int client\_id, String status, List<order\_product> list, float Cost){  
 this.client\_id=client\_id;  
 this.id=id;  
 this.Product = list;  
 this.Status = Order\_status.*valueOf*(status);  
 this.cost =Cost;  
 }  
 public Order (int id, List<order\_product> list, int client\_id)  
 {  
 this.client\_id=client\_id;  
 this.id=id;  
 this.Product = list;  
 this.cost = TotalOrderPrice();  
 this.Status = Order\_status.*AwaitConfirmationOfProductExistion*;  
 }  
  
 public int getOrderId()  
 {  
 return id;  
 }  
 public int getClientId()  
 {  
 return client\_id;  
 }  
  
 public void setStatus(Order\_status status)  
{  
 Status = status;  
}  
 public Order\_status getStatus()  
 {  
 return this.Status;  
 }  
 public void setcost(double cost)  
 {  
 this.cost = cost;  
 }  
 public double getcost()  
 {  
 return this.cost;  
 }  
 @OneToMany(mappedBy = "pk.order\_id", fetch = FetchType.*LAZY*)  
 public List<order\_product> getOrdersProducts() {  
 return Product;  
 }  
 @Transient  
 public float TotalOrderPrice() {  
 float sum = 0;  
 List<order\_product> orderProducts = getOrdersProducts();  
 for (order\_product op : orderProducts) {  
 sum += op.PriceOfOrder();  
 }  
 return sum;  
 }  
  
}

Класс «order\_product»

package courseWork.logic;  
  
import com.fasterxml.jackson.annotation.JsonIgnore;  
  
  
import javax.persistence.\*;  
import java.io.Serializable;  
  
  
@Entity  
@Table(name = "order\_products")  
public class order\_product implements Serializable {  
 @EmbeddedId  
 @JsonIgnore  
 private OrderProductPK pk;  
  
 public order\_product(){  
 super();  
 }  
 public order\_product(Order order\_id, Product product\_id, int id)  
 {  
  
 this.pk = new OrderProductPK();  
 pk.setOrder(order\_id);  
 pk.setProduct(product\_id);  
 }  
 public OrderProductPK getPk() {  
 return pk;  
 }  
  
 public void setPk(OrderProductPK pk) {  
 this.pk = pk;  
 }  
 @Transient  
 public Product getProduct() {  
 return this.pk.getProduct();  
 }  
  
 @Transient  
 public float PriceOfOrder() {  
 return getProduct().getProductPrice();  
 }  
}

Класс «OrderProductPK»

package courseWork.logic;  
  
import com.fasterxml.jackson.annotation.JsonBackReference;  
import org.hibernate.annotations.NotFound;  
import org.hibernate.annotations.NotFoundAction;  
  
import javax.persistence.Embeddable;  
import javax.persistence.FetchType;  
import javax.persistence.JoinColumn;  
import javax.persistence.ManyToOne;  
import java.io.Serializable;  
  
@Embeddable  
public class OrderProductPK implements Serializable {  
  
 @JsonBackReference  
 @ManyToOne(fetch = FetchType.*LAZY*)  
 @NotFound(action = NotFoundAction.*IGNORE*)  
 @JoinColumn(name = "order\_id")  
 private Order order;  
  
 @ManyToOne(fetch = FetchType.*LAZY*)  
 @NotFound(action = NotFoundAction.*IGNORE*)  
 @JoinColumn(name = "product\_id")  
 private Product product;  
  
 OrderProductPK(){}  
 public void setProduct(Product product) {  
 this.product = product;  
 }  
  
 public Product getProduct() {  
 return product;  
 }  
  
 public void setOrder(Order order)  
 {  
 this.order= order;  
 }  
  
 public Order getOrder() {  
 return order;  
 }  
}

Класс «Product»

package courseWork.logic;  
  
import com.fasterxml.jackson.annotation.JsonIgnore;  
  
import javax.persistence.\*;  
  
@Entity  
@Table(name = "product", schema = "shop")  
public class Product {  
  
 @Column(name = "product\_name")  
 private String Product\_Name;  
 @Id  
 @Column(name = "id")  
 @GeneratedValue  
 private int id;  
 @Column(name = "price")  
 private float price;  
 @Column(name = "category")  
 private String category;  
 @Column(name = "quantity")  
 private int quantity;  
 public Product (){}  
 public Product (String Name,String category, int id, float price,int quantity)  
 {  
 this.Product\_Name = Name;  
 this.id = id;  
 this.price = price;  
 this.category = category;  
 this.quantity = quantity;  
 }  
   
 public String getProductName()  
 {  
 return Product\_Name;  
 }  
 public void setProductName(String Product\_Name)  
 {  
 this.Product\_Name = Product\_Name;  
 }  
 public int getProductID()  
 {  
 return id;  
 }  
 public void setProductID(int id)  
 {  
 this.id = id;  
 }  
 public float getProductPrice()  
 {  
 return price;  
 }  
 public void setProductPrice(float price)  
 {  
 this.price = price;  
 }  
 public int getProductQuantity()  
 {  
 return quantity;  
 }  
 public void setProductQuantity(int quantity)  
 {  
 this.quantity = quantity;  
 }  
 public String getCategory()  
 {  
 return category;  
 }  
 public void setCategory(String category)  
 {  
 this.category = category;  
 }  
   
}

Класс «User»

package courseWork.logic;  
import javax.persistence.Column;  
import javax.persistence.GeneratedValue;  
import javax.persistence.Id;  
import javax.persistence.MappedSuperclass;  
  
@MappedSuperclass  
public class User  
{ @Column(name = "first\_name")  
 protected String FirstName;  
 @Column(name = "Surname")  
 protected String SurName;  
 @Id  
 @Column(name = "id")  
 @GeneratedValue  
 protected int id;  
 @Column(name = "password")  
 protected String password;  
 @Column(name = "Email")  
 protected String Email;  
 public User() {  
  
 }  
 public User (String FirstName, String SurName, int id, String password, String Email)  
 {  
 this.FirstName = FirstName;  
 this.SurName = SurName;  
 this.id=id;  
 this.password=password;  
 this.Email = Email;  
 }  
  
 public String getFirstName()  
 {  
 return FirstName;  
 }  
 public String getSurName()  
 {  
 return SurName;  
 }  
 public void setFirstName(String FirstName)  
 {  
 this.FirstName = FirstName;  
 }  
 public void setSurName(String SurName)  
 {  
 this.SurName = SurName;  
 }  
 public String getEmail()  
 {  
 return Email;  
 }  
 public void setEmail(String Email)  
 {  
 this.Email = Email;  
 }  
 public int getID ()  
 {  
 return id;  
 }  
 public void setID (int id)  
 {  
 this.id=id;  
 }  
 public String getPass ()  
 {  
 return password;  
 }  
 public void setPass (String pass)  
 {  
 this.password=pass;  
 }  
}

Класс «ClientRepo»

package courseWork.storage;  
  
  
import org.springframework.data.jpa.repository.JpaRepository;  
import courseWork.logic.Client;  
import org.springframework.data.jpa.repository.Query;  
import org.springframework.data.repository.query.Param;  
import org.springframework.stereotype.Repository;  
  
import java.util.Optional;  
  
@Repository  
public interface ClientRepo extends JpaRepository<Client, Integer> {  
 @Query(value = "SELECT \* FROM clients u WHERE u.Email = :Email AND u.password = :password", nativeQuery = true)  
 Optional<Client> getClientForLogin(@Param("Email") String Email, @Param("password") String password);  
  
 @Query(value = "SELECT \* FROM clients u WHERE u.Email = :Email", nativeQuery = true)  
 Optional<Client> getClientForLogin1(@Param("Email") String Email);  
  
}

Класс «OrderRepo»

package courseWork.storage;  
  
  
import courseWork.utils.Order\_status;  
import org.springframework.data.jpa.repository.JpaRepository;  
import courseWork.logic.Order;  
import org.springframework.data.jpa.repository.Modifying;  
import org.springframework.data.jpa.repository.Query;  
import org.springframework.data.repository.query.Param;  
import org.springframework.transaction.annotation.Transactional;  
  
import java.util.List;  
import java.util.Optional;  
  
public interface OrderRepo extends JpaRepository<Order, Integer> {  
 @Query(value = "SELECT \* FROM orders u where u.client\_id = :client\_id", nativeQuery = true)  
 List<Order> findOrdersByClient(@Param("client\_id") Integer client\_id);  
 @Query(value = "SELECT \* FROM orders ORDER BY order\_id DESC LIMIT 1", nativeQuery = true)  
 Order findLastOrder();  
 @Modifying  
 @Transactional  
 @Query(value = "INSERT INTO orders (order\_id, client\_id, order\_status, cost) VALUES (:order\_id, :client\_id, :order\_status, :cost)", nativeQuery = true)  
 void putNewOrder(@Param("order\_id") Integer order\_id, @Param("client\_id") int client\_id, @Param("order\_status") String order\_status, @Param("cost") Float cost);  
 @Query(value = "SELECT \* FROM orders u where u.order\_id = :order\_id", nativeQuery = true)  
 Optional<Order> findByOrderId(@Param("order\_id") Integer order\_id);  
}

Класс «ProductRepo»

package courseWork.storage;  
  
  
import org.springframework.data.jpa.repository.JpaRepository;  
import courseWork.logic.Product;  
import org.springframework.stereotype.Repository;  
  
@Repository  
public interface ProductRepo extends JpaRepository<Product, Integer> {  
  
}

Класс «OrderProductRepo»

package courseWork.storage;  
  
import courseWork.logic.OrderProductPK;  
import courseWork.logic.order\_product;  
import courseWork.utils.Order\_status;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.data.jpa.repository.Modifying;  
import org.springframework.data.jpa.repository.Query;  
import org.springframework.data.repository.query.Param;  
import org.springframework.transaction.annotation.Transactional;  
  
  
public interface OrderProductRepo extends JpaRepository<order\_product, Integer> {  
 @Modifying  
 @Transactional  
 @Query(value = "INSERT INTO order\_products (order\_id,product\_id) VALUES (:order\_id,:product\_id)", nativeQuery = true)  
 void putNewOrder(@Param("order\_id") Integer order\_id, @Param("product\_id") Integer product\_id);  
}

Класс «ClientService»

package courseWork.service;  
  
import courseWork.logic.Client;  
import courseWork.storage.ClientRepo;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
  
import javax.validation.constraints.NotNull;  
import java.util.NoSuchElementException;  
  
@Service  
@Transactional  
public class ClientService {  
  
 private Client currentClient;  
 private ClientRepo clientRepository;  
 @Autowired  
 public ClientService(ClientRepo clientRepository) {  
 this.clientRepository = clientRepository;  
 }  
  
 public Iterable<Client> getAllClients() {  
 return clientRepository.findAll();  
 }  
  
 public Client findClient (@NotNull final String userId, @NotNull final String password) throws NoSuchElementException {  
 if (clientRepository.getClientForLogin(userId, password).isPresent()) {  
 Client client = clientRepository.getClientForLogin(userId,password).get();  
 return client;  
 }  
 else {  
 throw new NoSuchElementException("User not found!");  
 }  
 }  
  
 public Client findClient1 (@NotNull final String userId, @NotNull final String password) throws NoSuchElementException {  
 if (clientRepository.getClientForLogin1(userId).isPresent()) {  
 Client client = clientRepository.getClientForLogin1(userId).get();  
 if (password.equals(client.getPass())) {  
 //if (client.getPass() == password)  
 this.currentClient = client;  
 return client;  
 }  
 else throw new NoSuchElementException("User not found/ wrong pass!");  
 }  
 else {  
 throw new NoSuchElementException("User not found!");  
 }  
 }  
  
 public String getCurrentClientName()  
 {  
 return this.currentClient.getFirstName();  
 }  
 public Client getCurrentClient()  
 {  
 return this.currentClient;  
 }  
 public String getClientPassword (@NotNull final String userId)  
 {  
 if (clientRepository.getClientForLogin1(userId).isPresent()) {  
 Client client = clientRepository.getClientForLogin1(userId).get();  
 return client.getPass();  
 }  
 else {  
 throw new NoSuchElementException("cant get password client");  
 }  
 }  
}

Класс «OrderService»

package courseWork.service;  
  
import courseWork.logic.Order;  
import courseWork.logic.order\_product;  
import courseWork.storage.OrderProductRepo;  
import courseWork.storage.OrderRepo;  
import courseWork.utils.Order\_status;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
  
import java.util.Iterator;  
import java.util.List;  
import java.util.Optional;  
  
@Service  
@Transactional  
public class OrderService {  
  
 private OrderRepo orderRepository;  
 private OrderProductRepo orderProductRepo;  
 @Autowired  
 public OrderService(OrderRepo orderRepository,OrderProductRepo orderProductRepo) {  
 this.orderRepository = orderRepository;  
 this.orderProductRepo = orderProductRepo;  
 }  
  
 public Iterable<Order> getAllOrders() {  
 return this.orderRepository.findAll();  
 }  
 public Order getLastOrder() {  
 return this.orderRepository.findLastOrder();  
 }  
 public Iterable<Order> getOrdersByClientId(int id) {  
 return this.orderRepository.findOrdersByClient(id);  
 }  
 public Optional<Order> getOrderbyId(int id) {  
 return this.orderRepository.findByOrderId(id);  
 }  
 public void createOrder(Integer order\_id, Integer client\_id, List<Integer> products,float cost) {  
 this.orderRepository.putNewOrder(order\_id,client\_id, Order\_status.*AwaitConfirmationOfProductExistion*.toString(),cost);  
 for (int i = 0; i < products.size(); i++) {  
 this.orderProductRepo.putNewOrder(order\_id,products.get(i));  
 }  
  
 }  
}

Класс «ProductService»

package courseWork.service;  
  
import courseWork.logic.Product;  
import courseWork.storage.ProductRepo;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
  
@Service  
@Transactional  
public class productService{  
  
 private ProductRepo productRepository;  
 @Autowired  
 public productService(ProductRepo productRepository) {  
 this.productRepository = productRepository;  
 }  
  
 public Iterable<Product> getAllProducts() {  
 return this.productRepository.findAll();  
 }  
}

Класс «OrderProductService»

package courseWork.service;  
  
import courseWork.storage.OrderProductRepo;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
  
@Service  
@Transactional  
public class OrderProductService{  
  
 private OrderProductRepo orderProductRepository;  
  
 public OrderProductService(OrderProductRepo orderProductRepository) {  
 this.orderProductRepository = orderProductRepository;  
 }  
}

Класс «MainController»

package courseWork.controllers;  
  
import courseWork.logic.Client;  
import courseWork.service.ClientService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.MediaType;  
import org.springframework.http.ResponseEntity;  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
import org.springframework.web.bind.annotation.\*;  
  
import javax.validation.constraints.NotNull;  
import java.util.Objects;  
import java.util.NoSuchElementException;  
  
@Controller  
public class MainController {  
  
 private ClientService clientService;  
 @Autowired  
 public void setService(@NotNull final ClientService service) {  
 this.clientService = service;  
 }  
  
 @GetMapping("/enter")  
 public String index1( ) {  
 return "index";  
 }  
  
 @RequestMapping(value = "/login", method = RequestMethod.*GET*,  
 produces = MediaType.*APPLICATION\_JSON\_VALUE*)  
 @ResponseBody  
 public ResponseEntity<Integer> logInUser(@RequestParam String login, @RequestParam String password) {  
 if (null == login) {  
 return new ResponseEntity<>(HttpStatus.*BAD\_REQUEST*);  
 }  
 try {  
 Client client = clientService.findClient1(login, password);  
 if (client != null)  
 return new ResponseEntity<>(3, HttpStatus.*OK*);  
 else return new ResponseEntity<>(HttpStatus.*NOT\_FOUND*);  
 } catch (NoSuchElementException e) {  
 return new ResponseEntity<>(HttpStatus.*NOT\_FOUND*);  
 }  
 }  
}

Класс «ClientController»

package courseWork.controllers;  
  
import courseWork.logic.Product;  
import courseWork.service.ClientService;  
import courseWork.service.productService;  
import courseWork.storage.OrderRepo;  
  
import org.apache.commons.lang3.StringUtils;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.MediaType;  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.\*;  
import courseWork.storage.ClientRepo;  
import courseWork.logic.Client;  
  
import javax.validation.constraints.NotNull;  
import java.util.Optional;  
  
@Controller  
@RequestMapping(path="/shop")  
public class ClientController {  
 private ClientService clientService;  
 courseWork.service.productService productService;  
 @Autowired  
 public void setService(@NotNull final ClientService service, @NotNull final productService productService) {  
 this.clientService = service;  
 this.productService = productService;  
 }  
  
  
 @GetMapping(path="/allClients")  
 public @ResponseBody Iterable<Client> getAllUsers() {  
 return clientService.getAllClients();  
 }  
  
 @RequestMapping(value = "/client", params = { "login", "password" }, method = RequestMethod.*GET*,  
 produces = MediaType.*APPLICATION\_JSON\_VALUE*)  
 public @ResponseBody Client getClient(@RequestParam("login") String login, @RequestParam("password") String password) {  
 Client client = clientService.findClient1(login, password);;  
// if (client != null)  
// this.currentClient = client;  
 return client;  
 }  
  
 @GetMapping(path="/currentclient")  
 public @ResponseBody Client getCurrentClient()  
 {  
 return this.clientService.getCurrentClient();  
 }  
}

Класс «ProductController»

package courseWork.controllers;  
  
import courseWork.logic.Product;  
import courseWork.service.productService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.ResponseBody;  
import org.springframework.web.bind.annotation.RestController;  
  
import javax.validation.constraints.NotNull;  
  
@RestController  
public class ProductController {  
  
 private productService productService;  
 @Autowired  
 public void setService(@NotNull final productService service) {  
 this.productService = service;  
 }  
  
 @GetMapping(path="/getAllProducts/")  
 public @ResponseBody  
 @NotNull Iterable<Product> getOrderProducts() {  
 // This returns a JSON or XML with the users  
 return this.productService.getAllProducts();  
 }  
}

Класс «OrderController»

package courseWork.controllers;  
  
  
import com.fasterxml.jackson.annotation.JsonProperty;  
import courseWork.logic.Order;  
import courseWork.service.OrderProductService;  
import courseWork.service.OrderService;  
import courseWork.service.productService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import javax.validation.constraints.NotNull;  
import java.util.ArrayList;  
import java.util.NoSuchElementException;  
import java.util.Optional;  
import java.util.List;  
  
@RestController  
@RequestMapping(path="/shop")  
public class OrderController {  
 @Autowired  
 productService productService;  
 @Autowired  
 OrderService orderService;  
 @Autowired  
 OrderProductService orderProductService;  
 public OrderController(productService productService, OrderService orderService, OrderProductService orderProductService) {  
 this.productService = productService;  
 this.orderService = orderService;  
 this.orderProductService = orderProductService;  
 }  
 @GetMapping(path="/getOrder/{order\_id}")  
 public @ResponseBody  
 Optional<Order> getOrderProductsByID(@PathVariable("order\_id") int order\_id) {  
 // This returns a JSON or XML with the users  
 return orderService.getOrderbyId(order\_id);  
 }  
  
 @GetMapping(path="/getAllOrder/")  
 public @ResponseBody @NotNull Iterable<Order> getOrderProducts() {  
 // This returns a JSON or XML with the users  
 return this.orderService.getAllOrders();  
 }  
  
 @GetMapping(path="/getLastOrder/")  
 public @ResponseBody @NotNull Order getLastOrder() {  
 // This returns a JSON or XML with the users  
 return this.orderService.getLastOrder();  
 }  
  
 @GetMapping(value = "/index")  
 public String list() {  
 return "start.html";  
 }  
  
 @GetMapping(path="/getOrdersByClient/{client\_id}")  
 public @ResponseBody  
 Iterable<Order> getOrdersByClientId(@PathVariable("client\_id") int client\_id) {  
 // This returns a JSON or XML with the users  
 return orderService.getOrdersByClientId(client\_id);  
 }  
 @PostMapping(path = "/neworder/",  
 consumes = "application/json")  
 public @ResponseBody  
 //ResponseEntity<String> clientOrder(@PathVariable("client\_id") int client\_id, @PathVariable("cost") Float cost,@PathVariable("list")List<Integer> productsoforder) {  
 //ResponseEntity<String> clientOrder( Integer client\_id, Float cost, List<Integer> productsoforder) {  
 ResponseEntity<String> clientOrder(@RequestBody OrderWrapper order) {  
 // This returns a JSON or XML with the users  
 Order lastorder = this.orderService.getLastOrder();  
 Integer lastOrderId = lastorder.getOrderId();  
 List<Integer> productsoforder = new ArrayList<Integer>();  
 productsoforder = order.getproductsoforder();  
 int client\_id = order.client\_id;  
 Float cost = order.cost;  
 try {  
 orderService.createOrder(lastOrderId+1, client\_id, productsoforder, cost);  
 return new ResponseEntity<>( HttpStatus.*CREATED*);  
 } catch (NoSuchElementException e) {  
 return new ResponseEntity<>(HttpStatus.*BAD\_REQUEST*);  
 }  
 }  
  
 public static class OrderWrapper{  
  
 @JsonProperty public int client\_id;  
 @JsonProperty public Float cost;  
 @JsonProperty List<Integer> list;  
 public OrderWrapper(){}  
 public OrderWrapper (int client\_id, Float cost, List<Integer> productsoforder)  
 {  
 this.client\_id=client\_id;  
 this.cost=cost;  
 setproductsoforder(productsoforder);  
 }  
 public List<Integer> getproductsoforder() {  
 return list;  
 }  
  
 public void setproductsoforder(List<Integer> productsoforder) {  
 this.list = productsoforder;  
 }  
  
  
 }  
  
  
}

Страница входа «index.html»

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>Login</title>  
 <!-- версия для разработки, отображает полезные предупреждения в консоли -->  
 <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>  
 <script src="https://cdn.jsdelivr.net/npm/vue-resource@1.5.1"></script>  
</head>  
<body>  
<h3>Login Page </h3>  
<div id="app">{{message}}</div>  
<script src="main.js"></script>  
</body>  
</html>

Скрипт «main.js»

Vue.component('message-form', {  
 props: ['messages'],  
 data: function() {  
 return {  
 login: '',  
 password: ''  
 }  
 },  
 template:  
 '<div>' +  
 '<input type="text" placeholder="Enter login" v-model="login" />' +  
 '<input type="text" placeholder="Enter password" v-model="password" />' +  
 '<input type="button" value="Log in" v-on:click="logIn" />' +  
 '</div>',  
 methods: {  
 logIn: function() {  
 Vue.http.get('http://localhost:3333/login?login=' + this.login + '&password=' +  
 this.password).then(result => {  
 if(result.ok) {  
 result.json().then(data => {  
 console.log(data);  
 this.login = '';  
 this.password = '';  
 switch(data.toString()) {  
 case '3':  
 location.replace("client.html");  
 break;  
 default:  
 console.log('No such user!');  
 }  
 })  
 } else {  
 console.log(result);  
 }  
 },  
 result => {  
 console.log(result);  
 }  
 );  
 }  
 }  
  
});  
  
Vue.component('messages-list', {  
 props: ['messages'],  
 template:  
 `<div id="app">  
 <message-form :messages="messages" />  
 </div>`  
});  
  
var app = new Vue({  
 el: '#app',  
 template: '<messages-list :messages="messages" />',  
 data: {  
 messages: [ ]  
 }  
 });

Страница клиента «client.html»

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>Shop</title>  
 <link href="https://cdnjs.cloudflare.com/ajax/libs/bulma/0.4.2/css/bulma.min.css" rel="stylesheet" />  
 <link type="text/css" rel="stylesheet" href="//unpkg.com/bootstrap/dist/css/bootstrap.min.css"/>  
 <link type="text/css" rel="stylesheet" href="//unpkg.com/bootstrap-vue@latest/dist/bootstrap-vue.css"/>  
 <!-- версия для разработки, отображает полезные предупреждения в консоли -->  
 <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>  
 <script src="https://cdn.jsdelivr.net/npm/vue-resource@1.5.1"></script>  
 <script src="//unpkg.com/bootstrap-vue@latest/dist/bootstrap-vue.js"></script>  
 <script src="https://cdnjs.cloudflare.com/ajax/libs/axios/0.19.1/axios.js"></script>  
</head>  
<body>  
<h3>Client Page</h3>  
<div id="app">{{message}}</div>  
<script src="client.js"></script>  
</body>  
</html>

Скрипт «client.js»

Vue.component('tabs', {  
 template: `  
 <div>  
 <div class="tabs">  
 <ul>  
 <li v-for="tab in tabs" :class="{ 'is-active': tab.isActive }">  
 <a :href="tab.href" @click="selectTab(tab)">{{ tab.name }}</a>  
 </li>  
 </ul>  
 </div>  
  
 <div class="tabs-details">  
 <slot></slot>  
 </div>  
 </div>  
 `,  
  
 data() {  
 return {tabs: [] };  
 },  
  
 created() {  
  
 this.tabs = this.$children;  
  
 },  
 methods: {  
 selectTab(selectedTab) {  
 this.tabs.forEach(tab => {  
 tab.isActive = (tab.name == selectedTab.name);  
 });  
 }  
 }  
});  
  
Vue.component('tab', {  
  
 template: `  
  
 <div v-show="isActive"><slot></slot></div>  
  
 `,  
  
 props: {  
 name: { required: true },  
 selected: { default: false}  
 },  
  
 data() {  
  
 return {  
 isActive: false  
 };  
  
 },  
  
 computed: {  
  
 href() {  
 return '#' + this.name.toLowerCase().replace(/ /g, '-');  
 }  
 },  
  
 mounted() {  
  
 this.isActive = this.selected;  
  
 }  
});  
  
Vue.component('ClientName', {  
 props:[ ],  
 data: function() {return {  
 results: ''  
 }  
 },  
 created: function () {  
 axios.get("http://localhost:3333/shop/currentclient").then((response) => {  
 console.log(response.data.firstName);  
 this.results = response.data.firstName  
 })  
 },  
 template:'<div> hello {{results}} </div>'  
});  
  
Vue.component('ProductsTable', {  
 props: [],  
  
 template: `<div>  
 <b-table  
 ref="selectableTable"  
 selectable  
 :select-mode='multi'  
 :items="items"  
 :fields="fields"  
 @row-selected="onRowSelected"  
  
 responsive="sm"  
 >  
 <!-- Example scoped slot for select state illustrative purposes -->  
 <template v-slot:cell(selected)="{ rowSelected }">  
 <template v-if="rowSelected">  
 <span aria-hidden="true">&check;</span>  
 <span class="sr-only">Selected</span>  
 </template>  
 <template v-else>  
 <span aria-hidden="true">&nbsp;</span>  
 <span class="sr-only">Not selected</span>  
 </template>  
 </template>  
 </b-table>  
 <p>  
 <b-button size="sm" @click="clearSelected">Clear selected</b-button>  
 <b-button size="sm" @click="createOrder">Create Order</b-button>  
 </p>  
 <p>  
 Your order:<br>  
 {{ selected }}  
 </p>  
 </div>  
 `,  
 data() {  
 return {  
 fields: ['productID','productName', 'productPrice', 'category', 'productQuantity'],  
 items: [  
 ],  
 selected: [],  
 clientId: 0,  
 orderCost: 0,  
 orderId: ''  
 }  
 },  
 computed: {  
 orderCost: function(){  
 let sum = 0;  
 for(let i = 0; i < this.selected.length; i++){  
 console.log(this.selected[i].productPrice)  
 sum += (parseFloat(this.selected[i].productPrice));  
 }  
 return sum;  
 }  
 },  
 methods: {  
 onRowSelected(items) {  
 this.selected = items  
 },  
 createOrder(items) {  
 var id = 0  
 axios.get("http://localhost:3333/shop/currentclient").then((response) => {  
  
 this.clientId = response.data.id  
 })  
  
 let sum = 0;  
 for(let i = 0; i < this.selected.length; i++){  
 sum += (parseFloat(this.selected[i].productPrice))  
 }  
 this.orderCost = sum  
  
  
 var productlist = []  
 var productsID = []  
 let product = {productID : 0, productName : 0, productPrice : 0, category : 0, productQuantity : 0}  
 for(let i = 0; i < this.selected.length; i++){  
 Object.assign(product, {productID : this.selected[i].productID,productName : this.selected[i].productName,  
 productPrice : this.selected[i].productPrice, category : this.selected[i].category,  
 productQuantity : this.selected[i].productQuantity})  
 productlist.push(product)  
 productsID.push(this.selected[i].productID)  
 product = {productID : 0, productName : 0, productPrice : 0, category : 0, productQuantity : 0}  
 }  
  
 axios({  
 method: 'post',  
 url: 'http://localhost:3333/shop/neworder/',  
 data: {  
 client\_id : this.clientId,  
 cost : this.orderCost,  
 list: productsID  
 }  
 });  
 productlist = []  
 productsID = []  
 },  
 clearSelected() {  
 this.$refs.selectableTable.clearSelected()  
 }  
 },  
 created: function () {  
 axios.get("http://localhost:3333/getAllProducts/").then((response) => {  
  
 this.items = response.data  
 })  
 axios.get("http://localhost:3333/shop/currentclient").then((response) => {  
  
 this.clientId = response.data.id  
 })  
 }  
});  
  
  
Vue.component('OrdersTable', {  
 props: [],  
  
 template: `<div>  
 <b-table  
 ref="selectableTable"  
 selectable  
 :select-mode='none'  
 :items="items"  
 :fields="fields"  
 @row-selected="onRowSelected"  
  
 responsive="sm"  
 >  
 <!-- Example scoped slot for select state illustrative purposes -->  
 <template v-slot:cell(selected)="{ rowSelected }">  
 <template v-if="rowSelected">  
 <span aria-hidden="true">&check;</span>  
 <span class="sr-only">Selected</span>  
 </template>  
 <template v-else>  
 <span aria-hidden="true">&nbsp;</span>  
 <span class="sr-only">Not selected</span>  
 </template>  
 </template>  
 </b-table>  
 <p>  
  
 </p>  
 </div>  
 `,  
 data() {  
 return {  
 fields: ['orderId','products', 'cost', 'status'],  
 items: [  
 ],  
 selected: [],  
 clientId: ''  
 }  
 },  
 methods: {  
 onRowSelected(items) {  
 this.selected = items  
 },  
 async getUser (){  
 var self = this  
 await axios.get("http://localhost:3333/shop/currentclient").then((response) => {  
 console.log(response.data.id)  
 self.clientId = response.data.id  
 })  
 console.log(self.clientId)  
  
  
  
  
 },  
 async getItems (){  
 var self = this  
 var Orders= []  
 console.log("http://localhost:3333/getOrdersByClient/"+self.clientId)  
 await axios.get("http://localhost:3333/shop/getOrdersByClient/"+self.clientId).then((response) => {  
 console.log(response.data)  
 Orders = response.data})  
 console.log(Orders)  
 let Order = {orderId : 0, products : [], cost : 0, status : ''}  
 for(let i = 0; i < Orders.length; i++){  
 var productInOrder = []  
 for (let j = 0; j < Orders[i].ordersProducts.length; j++)  
 {  
 console.log(Orders[i].ordersProducts[j])  
 console.log(Orders[i].ordersProducts[j].product.productName)  
 productInOrder.push(Orders[i].ordersProducts[j].product.productName)  
 }  
 Object.assign(Order,{orderId : Orders[i].orderId, products : productInOrder, cost : Orders[i].cost, status : Orders[i].status})  
 self.items.push(Order)  
 Order = {orderId : 0, products : [], cost : 0, status : ''}  
 productInOrder = []  
 }  
 console.log(self.items)  
 },  
 async create()  
 {  
 await this.getUser ()  
 console.log("http://localhost:3333/getOrdersByClient/"+this.clientId)  
 await this.getItems()  
 console.log(this.Items)  
 }  
 },  
 created : function () {  
 this.create()  
 }  
  
});  
  
Vue.component('messages-list', {  
 props: [],  
 template: `  
 <div id="app" class="container">  
 <ClientName> </ClientName>  
 <tabs>  
 <tab name="Create order" :selected="true">  
 <h1>Here you can create order</h1>  
 <ProductsTable> </ProductsTable>  
 </tab>  
 <tab name="Your orders">  
 <h1> Your order history</h1>  
 <OrdersTable> </OrdersTable>  
 </tab>  
 </tabs>  
 </div>`,  
});  
  
var app = new Vue({  
 el: '#app',  
 template: '<messages-list />',  
 data: {  
 messages: [ ]  
 }  
 });