МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Факультет информационных систем и технологий

Кафедра «Информационные системы»

Дисциплина «Интернет программирование»

**Отчёт по лабораторной работе №3**

**Персистентность**

Выполнил студент

гр. ПИбд-21:

Сергеев Н.И.

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

Научный руководитель:

Филиппов А.А.

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

Ульяновск

2022

**Задание**

1. Реализовать не менее 3 классов-сущностей (Entity) по выбранной предметной области. Рекомендуется использовать предметную область из лабораторных работ предыдущего семестра.
2. Между сущностями должно быть не менее одного двунаправленного отношения вида «один ко многим» или «многие ко многим».
3. Реализовать для каждой сущности собственный класс с бизнес-логикой (Service) для ее создания, редактирования, чтения и удаления (CRUD).
4. Написать модульные тесты для тестирования классов с бизнес-логикой.

**Сервер**

1. **Классы-сущности**

Компонент (связан с продуктом многие-ко-многим)

package ip.labwork.shop.model;

import com.fasterxml.jackson.annotation.JsonIgnore;

import jakarta.persistence.\*;

import java.util.ArrayList;

import java.util.List;

import java.util.Objects;

@Entity

@Table(name = "components")

public class Component {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private Long id;

@Column(name = "name")

private String componentName;

@Column(name = "price")

private Integer price;

@OneToMany(mappedBy = "component", cascade = CascadeType.ALL, fetch = FetchType.EAGER)

@JsonIgnore

private List<ProductComponents> products;

public Component() {

}

public Component(String componentName, Integer price) {

this.componentName = componentName;

this.price = price;

}

public Long getId() {

return id;

}

public String getComponentName() {

return componentName;

}

public void setComponentName(String componentName) {

this.componentName = componentName;

}

public Integer getPrice() {

return price;

}

public void setPrice(Integer price) {

this.price = price;

}

public List<ProductComponents> getProducts() {

return products;

}

public void setProducts(List<ProductComponents> products) {

this.products = products;

}

public void addProduct(ProductComponents productComponents) {

if (products == null) {

products = new ArrayList<>();

}

if (!products.contains(productComponents))

this.products.add(productComponents);

}

public void removeProduct(ProductComponents productComponents) {

if (products.contains(productComponents))

this.products.remove(productComponents);

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

Component component = (Component) o;

return Objects.equals(id, component.id);

}

@Override

public int hashCode() {

return Objects.hash(id);

}

@Override

public String toString() {

return "Component{" +

"id=" + id +

", componentName='" + componentName + '\'' +

", price='" + price + '\'' +

'}';

}

}

Продукт (связан многие-ко-многим с компонентом и заказом)

package ip.labwork.shop.model;

import com.fasterxml.jackson.annotation.JsonIgnore;

import jakarta.persistence.\*;

import java.util.ArrayList;

import java.util.List;

import java.util.Objects;

@Entity

@Table(name = "products")

public class Product {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private Long id;

@Column(name = "name")

private String productName;

@Column(name = "price")

private Integer price;

@OneToMany(mappedBy = "product", cascade = CascadeType.ALL, fetch = FetchType.EAGER)

private List<ProductComponents> components;

@OneToMany(mappedBy = "product", cascade = CascadeType.ALL, fetch = FetchType.EAGER)

@JsonIgnore

private List<OrderProducts> orders;

public Product() {

}

public Product(String productName, Integer price) {

this.productName = productName;

this.price = price;

}

public Long getId() {

return id;

}

public String getProductName() {

return productName;

}

public void setProductName(String productName) {

this.productName = productName;

}

public Integer getPrice() {

return price;

}

public void setPrice(Integer price) {

this.price = price;

}

public List<ProductComponents> getComponents() {

return components;

}

public void setComponents(List<ProductComponents> components) {

this.components = components;

}

public void update(Product product){

this.productName = product.productName;

this.price = product.price;

this.components = product.getComponents();

}

public void addComponent(ProductComponents productComponents){

if (components == null){

this.components = new ArrayList<>();

}

if (!components.contains(productComponents))

this.components.add(productComponents);

}

public void removeComponent(ProductComponents productComponents){

if (components.contains(productComponents))

this.components.remove(productComponents);

}

public List<OrderProducts> getOrders() {

return orders;

}

public void setOrders(List<OrderProducts> orders) {

this.orders = orders;

}

public void addOrder(OrderProducts orderProducts){

if (orders == null){

orders = new ArrayList<>();

}

if (!orders.contains(orderProducts))

this.orders.add(orderProducts);

}

public void removeOrder(OrderProducts orderProducts){

if (orders.contains(orderProducts))

this.orders.remove(orderProducts);

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

Product product = (Product) o;

return Objects.equals(id, product.id);

}

@Override

public int hashCode() {

return Objects.hash(id);

}

@Override

public String toString() {

return "Product{" +

"id=" + id +

", productName='" + productName + '\'' +

", price='" + price + '\'' +

'}';

}

}

Сущность связь компонента и продукта

package ip.labwork.shop.model;

import com.fasterxml.jackson.annotation.JsonIgnore;

import jakarta.persistence.\*;

@Entity

@Table(name = "product\_component")

public class ProductComponents {

@EmbeddedId

private ProductComponentsKey id;

@ManyToOne

@MapsId("componentId")

@JoinColumn(name = "component\_id")

private Component component;

@ManyToOne

@MapsId("productId")

@JoinColumn(name = "product\_id")

@JsonIgnore

private Product product;

@Column(name = "count")

private Integer count;

public ProductComponents() {

}

public ProductComponents(Component component, Product product, Integer count) {

this.component = component;

this.id = new ProductComponentsKey(product.getId(), component.getId());

this.id.setComponentId(component.getId());

this.id.setProductId(product.getId());

this.product = product;

this.count = count;

}

public ProductComponentsKey getId() {

return id;

}

public void setId(ProductComponentsKey id) {

this.id = id;

}

public Component getComponent() {

return component;

}

public void setComponent(Component component) {

this.component = component;

}

public Product getProduct() {

return product;

}

public void setProduct(Product product) {

this.product = product;

}

public Integer getCount() {

return count;

}

public void setCount(Integer count) {

this.count = count;

}

}

Композитный ключ для Продукт-Компонент

package ip.labwork.shop.model;

import jakarta.persistence.Embeddable;

import java.io.Serializable;

import java.util.Objects;

@Embeddable

public class ProductComponentsKey implements Serializable {

private Long productId;

private Long componentId;

public ProductComponentsKey() {

}

public ProductComponentsKey(Long productId, Long componentId) {

this.productId = productId;

this.componentId = componentId;

}

public Long getProductId() {

return productId;

}

public void setProductId(Long productId) {

this.productId = productId;

}

public Long getComponentId() {

return componentId;

}

public void setComponentId(Long componentId) {

this.componentId = componentId;

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (!(o instanceof ProductComponentsKey that)) return false;

return Objects.equals(getProductId(), that.getProductId()) && Objects.equals(getComponentId(), that.getComponentId());

}

@Override

public int hashCode() {

return Objects.hash(getProductId(), getComponentId());

}

}

Сущность связь заказа и продукта

package ip.labwork.shop.model;

import com.fasterxml.jackson.annotation.JsonIgnore;

import jakarta.persistence.\*;

@Entity

@Table(name = "order\_product")

public class OrderProducts {

@EmbeddedId

private OrderProductsKey id;

@ManyToOne

@MapsId("productId")

@JoinColumn(name = "product\_id")

private Product product;

@ManyToOne

@MapsId("orderId")

@JoinColumn(name = "order\_id")

@JsonIgnore

private Order order;

@Column(name = "count")

private Integer count;

public OrderProducts() {

}

public OrderProducts(Order order, Product product, Integer count) {

this.order = order;

this.id = new OrderProductsKey(product.getId(), order.getId());

this.id.setOrderId(order.getId());

this.id.setProductId(product.getId());

this.product = product;

this.count = count;

}

public OrderProductsKey getId() {

return id;

}

public void setId(OrderProductsKey id) {

this.id = id;

}

public Order getOrder() {

return order;

}

public void setOrder(Order order) {

this.order = order;

}

public Product getProduct() {

return product;

}

public void setProduct(Product product) {

this.product = product;

}

public Integer getCount() {

return count;

}

public void setCount(Integer count) {

this.count = count;

}

}

Композитный ключ для Заказ-Продукт

package ip.labwork.shop.model;

import jakarta.persistence.Embeddable;

import java.io.Serializable;

import java.util.Objects;

@Embeddable

public class OrderProductsKey implements Serializable {

private Long productId;

private Long orderId;

public OrderProductsKey() {

}

public OrderProductsKey(Long productId, Long orderId) {

this.productId = productId;

this.orderId = orderId;

}

public Long getProductId() {

return productId;

}

public void setProductId(Long productId) {

this.productId = productId;

}

public Long getOrderId() {

return orderId;

}

public void setOrderId(Long orderId) {

this.orderId = orderId;

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (!(o instanceof OrderProductsKey that)) return false;

return Objects.equals(getProductId(), that.getProductId()) && Objects.equals(getOrderId(), that.getOrderId());

}

@Override

public int hashCode() {

return Objects.hash(getProductId(), getOrderId());

}

}

1. **Бизнес-логика**

Компонент

package ip.labwork.shop.service;

import ip.labwork.shop.model.Component;

import ip.labwork.shop.model.OrderProducts;

import ip.labwork.shop.model.Product;

import ip.labwork.shop.model.ProductComponents;

import jakarta.persistence.EntityManager;

import jakarta.persistence.EntityNotFoundException;

import jakarta.persistence.PersistenceContext;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import org.springframework.util.StringUtils;

import java.util.ArrayList;

import java.util.List;

@Service

public class ComponentService {

@PersistenceContext

private EntityManager em;

@Transactional

public Component addComponent(String componentName, Integer price) {

if (!StringUtils.hasText(componentName) || price == 0) {

throw new IllegalArgumentException("Component is null or empty");

}

final Component component = new Component(componentName, price);

em.persist(component);

return component;

}

@Transactional(readOnly = true)

public Component findComponent(Long id) {

final Component component = em.find(Component.class, id);

if (component == null) {

throw new EntityNotFoundException(String.format("Component with id [%s] is not found", id));

}

return component;

}

@Transactional(readOnly = true)

public List<Component> findAllComponent() {

return em.createQuery("select c from Component c", Component.class)

.getResultList();

}

@Transactional(readOnly = true)

public List<Component> findFiltredComponents(Long[] arr) {

if (arr.length == 0) {

throw new IllegalArgumentException("Array id is empty");

}

List<Component> componentList = new ArrayList<>();

for (int i = 0; i < arr.length; i++) {

componentList.add(em.find(Component.class, arr[i]));

}

return componentList;

}

@Transactional

public Component updateComponent(Long id, String componentName, Integer price) {

if (!StringUtils.hasText(componentName) || price == 0) {

throw new IllegalArgumentException("Component is null or empty");

}

final Component currentComponent = findComponent(id);

currentComponent.setComponentName(componentName);

currentComponent.setPrice(price);

return em.merge(currentComponent);

}

@Transactional

public Component deleteComponent(Long id) {

final Component currentComponent = findComponent(id);

int size = currentComponent.getProducts().size();

for (int i = 0; i < size; i++) {

ProductComponents temp = currentComponent.getProducts().get(0);

temp.getComponent().removeProduct(temp);

temp.getProduct().removeComponent(temp);

em.remove(temp);

}

em.remove(currentComponent);

return currentComponent;

}

@Transactional

public void deleteAllComponent() {

em.createQuery("delete from ProductComponents").executeUpdate();

em.createQuery("delete from Component").executeUpdate();

}

}

Продукт

package ip.labwork.shop.service;

import ip.labwork.shop.model.Component;

import ip.labwork.shop.model.OrderProducts;

import ip.labwork.shop.model.Product;

import ip.labwork.shop.model.ProductComponents;

import jakarta.persistence.EntityManager;

import jakarta.persistence.EntityNotFoundException;

import jakarta.persistence.PersistenceContext;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import org.springframework.util.StringUtils;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

import java.util.Objects;

@Service

public class ProductService {

@PersistenceContext

private EntityManager em;

@Transactional

public Product addProduct(String productName, Integer price, Integer[] count, List<Component> components) {

if (!StringUtils.hasText(productName) || price == 0 || count.length == 0 || Arrays.stream(count).filter(c -> c == 0).toList().size() != 0 || components.size() == 0 || components.stream().filter(Objects::isNull).toList().size() != 0 || count.length != components.size()) {

throw new IllegalArgumentException("Product name is null or empty");

}

final Product product = new Product(productName, price);

em.persist(product);

for (int i = 0; i < components.size(); i++) {

final ProductComponents productComponents = new ProductComponents(components.get(i), product, count[i]);

product.addComponent(productComponents);

components.get(i).addProduct(productComponents);

em.persist(productComponents);

}

return product;

}

@Transactional(readOnly = true)

public Product findProduct(Long id) {

final Product product = em.find(Product.class, id);

if (product == null) {

throw new EntityNotFoundException(String.format("Product with id [%s] is not found", id));

}

return product;

}

@Transactional(readOnly = true)

public List<Product> findAllProduct() {

return em.createQuery("select p from Product p", Product.class)

.getResultList();

}

@Transactional

public Product updateProduct(Long id, String productName, Integer price, Integer[] count, List<Component> components) {

if (!StringUtils.hasText(productName) || price == 0 || count.length == 0 || Arrays.stream(count).filter(c -> c == 0).toList().size() != 0 || components.size() == 0 || components.stream().filter(Objects::isNull).toList().size() != 0 || count.length != components.size()) {

throw new IllegalArgumentException("Product name is null or empty");

}

final Product currentProduct = findProduct(id);

currentProduct.setProductName(productName);

currentProduct.setPrice(price);

em.merge(currentProduct);

List<ProductComponents> productComponentsList = em.createQuery("select p from ProductComponents p where p.id.productId = " + id, ProductComponents.class)

.getResultList();

List<Long> component\_id = new ArrayList<>(productComponentsList.stream().map(p -> p.getId().getComponentId()).toList());

for (int i = 0; i < components.size(); i++) {

final Long currentId = components.get(i).getId();

if (component\_id.contains(currentId)) {

final ProductComponents productComponents = productComponentsList.stream().filter(x -> Objects.equals(x.getId().getComponentId(), currentId)).toList().get(0);

productComponentsList.remove(productComponents);

component\_id.remove(components.get(i).getId());

productComponents.setCount(count[i]);

em.merge(productComponents);

} else {

final ProductComponents productComponents = new ProductComponents(components.get(i), currentProduct, count[i]);

currentProduct.addComponent(productComponents);

components.get(i).addProduct(productComponents);

em.persist(productComponents);

}

}

for (int i = 0; i < productComponentsList.size(); i++) {

productComponentsList.get(i).getComponent().removeProduct(productComponentsList.get(i));

productComponentsList.get(i).getProduct().removeComponent(productComponentsList.get(i));

em.remove(productComponentsList.get(i));

}

return currentProduct;

}

@Transactional

public Product deleteProduct(Long id) {

final Product currentProduct = findProduct(id);

int size = currentProduct.getComponents().size();

for (int i = 0; i < size; i++) {

ProductComponents temp = currentProduct.getComponents().get(0);

temp.getComponent().removeProduct(temp);

temp.getProduct().removeComponent(temp);

em.remove(temp);

}

int ordSize = currentProduct.getOrders().size();

for (int i = 0; i < ordSize; i++){

OrderProducts temp = currentProduct.getOrders().get(0);

temp.getProduct().removeOrder(temp);

temp.getOrder().removeProducts(temp);

em.remove(temp);

}

em.remove(currentProduct);

return currentProduct;

}

@Transactional

public void deleteAllProduct() {

em.createQuery("delete from ProductComponents").executeUpdate();

em.createQuery("delete from OrderProducts ").executeUpdate();

em.createQuery("delete from Product").executeUpdate();

}

@Transactional

public List<Product> findFiltredProducts(Long[] arr) {

if (arr.length == 0) {

throw new IllegalArgumentException("Array id is empty");

}

List<Product> productList = new ArrayList<>();

for (int i = 0; i < arr.length; i++) {

productList.add(em.find(Product.class, arr[i]));

}

return productList;

}

}

Заказ

package ip.labwork.shop.service;

import ip.labwork.shop.model.Order;

import ip.labwork.shop.model.OrderProducts;

import ip.labwork.shop.model.Product;

import jakarta.persistence.EntityManager;

import jakarta.persistence.EntityNotFoundException;

import jakarta.persistence.PersistenceContext;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import org.springframework.util.StringUtils;

import java.text.SimpleDateFormat;

import java.util.\*;

@Service

public class OrderService {

@PersistenceContext

private EntityManager em;

@Transactional

public Order addOrder(String date, Integer price, Integer[] count, List<Product> products) {

if (!StringUtils.hasText(date) || price == 0 || count.length == 0 || Arrays.stream(count).filter(c -> c == 0).toList().size() != 0 || products.size() == 0 || products.stream().filter(Objects::isNull).toList().size() != 0 || count.length != products.size()) {

throw new IllegalArgumentException("Order is null or empty");

}

Date correctDate = getDate(date);

final Order order = new Order(correctDate, price);

em.persist(order);

for (int i = 0; i < products.size(); i++) {

final OrderProducts orderProducts = new OrderProducts(order, products.get(i), count[i]);

order.addProduct(orderProducts);

products.get(i).addOrder(orderProducts);

em.persist(orderProducts);

}

return order;

}

public Date getDate(String date) {

SimpleDateFormat format = new SimpleDateFormat();

format.applyPattern("dd.MM.yyyy");

Date newDate;

try {

newDate = format.parse(date);

} catch (Exception exception) {

newDate = new Date();

}

return newDate;

}

@Transactional(readOnly = true)

public Order findOrder(Long id) {

final Order order = em.find(Order.class, id);

if (order == null) {

throw new EntityNotFoundException(String.format("Order with id [%s] is not found", id));

}

return order;

}

@Transactional(readOnly = true)

public List<Order> findAllOrder() {

return em.createQuery("select o from Order o", Order.class)

.getResultList();

}

@Transactional

public Order updateOrder(Long id, String date, Integer price, Integer[] count, List<Product> products) {

if (!StringUtils.hasText(date) || price == 0 || count.length == 0 || Arrays.stream(count).filter(c -> c == 0).toList().size() != 0 || products.size() == 0 || products.stream().filter(Objects::isNull).toList().size() != 0 || count.length != products.size()) {

throw new IllegalArgumentException("Order is null or empty");

}

final Order currentOrder = findOrder(id);

currentOrder.setDate(getDate(date));

currentOrder.setPrice(price);

em.merge(currentOrder);

List<OrderProducts> orderProductsList = em.createQuery("select o from OrderProducts o where o.id.orderId = " + id, OrderProducts.class)

.getResultList();

List<Long> product\_id = new ArrayList<>(orderProductsList.stream().map(p -> p.getId().getProductId()).toList());

for (int i = 0; i < products.size(); i++) {

final Long currentId = products.get(i).getId();

if (product\_id.contains(currentId)) {

final OrderProducts orderProducts = orderProductsList.stream().filter(x -> Objects.equals(x.getId().getProductId(), currentId)).toList().get(0);

orderProductsList.remove(orderProducts);

product\_id.remove(products.get(i).getId());

orderProducts.setCount(count[i]);

em.merge(orderProducts);

} else {

final OrderProducts orderProducts = new OrderProducts(currentOrder, products.get(i), count[i]);

currentOrder.addProduct(orderProducts);

products.get(i).addOrder(orderProducts);

em.persist(orderProducts);

}

}

for (int i = 0; i < orderProductsList.size(); i++) {

orderProductsList.get(i).getProduct().removeOrder(orderProductsList.get(i));

orderProductsList.get(i).getOrder().removeProducts(orderProductsList.get(i));

em.remove(orderProductsList.get(i));

}

return currentOrder;

}

@Transactional

public Order deleteOrder(Long id) {

final Order currentOrder = findOrder(id);

int size = currentOrder.getProducts().size();

for (int i = 0; i < size; i++) {

OrderProducts temp = currentOrder.getProducts().get(0);

temp.getProduct().removeOrder(temp);

temp.getOrder().removeProducts(temp);

em.remove(temp);

}

em.remove(currentOrder);

return currentOrder;

}

@Transactional

public void deleteAllOrder() {

em.createQuery("delete from OrderProducts").executeUpdate();

em.createQuery("delete from Order").executeUpdate();

}

}

1. **Контроллеры**

Компонент

package ip.labwork.shop.controller;

import ip.labwork.shop.model.Component;

import ip.labwork.shop.service.ComponentService;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.RestController;

import java.util.List;

@RestController

@RequestMapping("/component")

public class ComponentController {

private final ComponentService componentService;

public ComponentController(ComponentService componentService) {

this.componentService = componentService;

}

@GetMapping("/add")

public Component create(@RequestParam("name") String name,

@RequestParam("price") Integer price) {

return componentService.addComponent(name, price);

}

@GetMapping("/update")

public Component update(@RequestParam("id") Long id,

@RequestParam("name") String name,

@RequestParam("price") Integer price) {

return componentService.updateComponent(id, name, price);

}

@GetMapping("/remove")

public Component remove(@RequestParam("id") Long id) {

return componentService.deleteComponent(id);

}

@GetMapping("/removeAll")

public void remove() {

componentService.deleteAllComponent();

}

@GetMapping("/find")

public Component find(@RequestParam("id") Long id) {

return componentService.findComponent(id);

}

@GetMapping("/findAll")

public List<Component> findAll() {

return componentService.findAllComponent();

}

Продукт

package ip.labwork.shop.controller;

import ip.labwork.shop.service.ProductService;

import ip.labwork.shop.model.Order;

import ip.labwork.shop.service.OrderService;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.RestController;

import java.util.List;

@RestController

@RequestMapping("/order")

public class OrderController {

private final OrderService orderService;

private final ProductService productService;

public OrderController(OrderService orderService, ProductService productService) {

this.orderService = orderService;

this.productService = productService;

}

@GetMapping("/add")

public Order create(@RequestParam("date") String date,

@RequestParam("price") Integer price,

@RequestParam("count") Integer[] count,

@RequestParam("prod") Long[] prod){

return orderService.addOrder(date, price, count, productService.findFiltredProducts(prod));

}

@GetMapping("/update")

public Order update(@RequestParam("id") Long id,

@RequestParam("date") String date,

@RequestParam("price") Integer price,

@RequestParam("count") Integer[] count,

@RequestParam("prod") Long[] prod){

return orderService.updateOrder(id, date, price, count, productService.findFiltredProducts(prod));

}

@GetMapping("/remove")

public Order remove(@RequestParam("id") Long id){

return orderService.deleteOrder(id);

}

@GetMapping("/removeAll")

public void remove(){

orderService.deleteAllOrder();

}

@GetMapping("/find")

public Order find(@RequestParam("id") Long id){

return orderService.findOrder(id);

}

@GetMapping("/findAll")

public List<Order> findAll(){

return orderService.findAllOrder();

}

}

Заказ

package ip.labwork.shop.controller;

import ip.labwork.shop.service.ProductService;

import ip.labwork.shop.model.Product;

import ip.labwork.shop.service.ComponentService;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.RestController;

import java.util.List;

@RestController

@RequestMapping("/product")

public class ProductController {

private final ProductService productService;

private final ComponentService componentService;

public ProductController(ProductService productService, ComponentService componentService) {

this.productService = productService;

this.componentService = componentService;

}

@GetMapping("/add")

public Product create(@RequestParam("name") String name,

@RequestParam("price") Integer price,

@RequestParam("count") Integer[] count,

@RequestParam("comp") Long[] comp){

return productService.addProduct(name, price, count, componentService.findFiltredComponents(comp));

}

@GetMapping("/update")

public Product update(@RequestParam("id") Long id,

@RequestParam("name") String name,

@RequestParam("price") Integer price,

@RequestParam("count") Integer[] count,

@RequestParam("comp") Long[] comp){

return productService.updateProduct(id, name, price, count, componentService.findFiltredComponents(comp));

}

@GetMapping("/remove")

public Product remove(@RequestParam("id") Long id){

return productService.deleteProduct(id);

}

@GetMapping("/removeAll")

public void remove(){

productService.deleteAllProduct();

}

@GetMapping("/find")

public Product find(@RequestParam("id") Long id){

return productService.findProduct(id);

}

@GetMapping("/findAll")

public List<Product> findAll(){

return productService.findAllProduct();

}

}

1. **Тесты**

package ip.labwork;

import ip.labwork.shop.model.Component;

import ip.labwork.shop.model.Order;

import ip.labwork.shop.model.Product;

import ip.labwork.shop.service.ComponentService;

import ip.labwork.shop.service.OrderService;

import ip.labwork.shop.service.ProductService;

import jakarta.persistence.EntityNotFoundException;

import org.junit.jupiter.api.Assertions;

import org.junit.jupiter.api.Test;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.boot.test.context.SpringBootTest;

import java.util.ArrayList;

import java.util.Date;

import java.util.List;

@SpringBootTest

public class JpaStudentTests {

private static final Logger log = LoggerFactory.getLogger(JpaStudentTests.class);

@Autowired

ComponentService componentService;

@Autowired

ProductService productService;

@Autowired

OrderService orderService;

@Test

void test() {

componentService.deleteAllComponent();

productService.deleteAllProduct();

orderService.deleteAllOrder();

//TestCreate

final Component component = componentService.addComponent("Огурец", 4);

log.info(component.toString());

Assertions.assertNotNull(component.getId());

List<Component> componentList = new ArrayList<>();

componentList.add(componentService.findComponent(component.getId()));

final Product product = productService.addProduct("Бургер", 100, new Integer[]{ 2 }, componentList);

log.info(product.toString());

Assertions.assertNotNull(product.getId());

List<Product> productList = new ArrayList<>();

productList.add(productService.findProduct(product.getId()));

final Order order = orderService.addOrder(new Date().toString(), 200, new Integer[]{ 3 }, productList);

log.info(order.toString());

Assertions.assertNotNull(order.getId());

//TestRead

final Component findComponent = componentService.findComponent(component.getId());

log.info(findComponent.toString());

Assertions.assertEquals(component, findComponent);

final Product findProduct = productService.findProduct(product.getId());

log.info(findProduct.toString());

Assertions.assertEquals(product, findProduct);

final Order findOrder = orderService.findOrder(order.getId());

log.info(findOrder.toString());

Assertions.assertEquals(order, findOrder);

//TestReadAll

final List<Component> components = componentService.findAllComponent();

log.info(components.toString());

Assertions.assertEquals(components.size(), 1);

final List<Product> products = productService.findAllProduct();

log.info(products.toString());

Assertions.assertEquals(products.size(), 1);

final List<Order> orders = orderService.findAllOrder();

log.info(orders.toString());

Assertions.assertEquals(orders.size(), 1);

//TestReadNotFound

componentService.deleteAllComponent();

productService.deleteAllProduct();

orderService.deleteAllOrder();

Assertions.assertThrows(EntityNotFoundException.class, () -> componentService.findComponent(-1L));

Assertions.assertThrows(EntityNotFoundException.class, () -> productService.findProduct(-1L));

Assertions.assertThrows(EntityNotFoundException.class, () -> orderService.findOrder(-1L));

//TestReadAllEmpty

final List<Component> newComponents = componentService.findAllComponent();

log.info(newComponents.toString());

Assertions.assertEquals(newComponents.size(), 0);

final List<Product> newProducts = productService.findAllProduct();

log.info(newProducts.toString());

Assertions.assertEquals(newProducts.size(), 0);

final List<Order> newOrders = orderService.findAllOrder();

log.info(newOrders.toString());

Assertions.assertEquals(newOrders.size(), 0);

}

}