NAD Final Project Report

Team:

- **Yahia Kettani**
- **Hadi Nofal**
- **Eyad Bereh**

Brief overview

The idea behind the project is an E-commerce website that allows users to view and buy products.

The website allows the visitor to view products by categories, or search about some products that falls under a set of categories, and view all details about a specific product.

The following images are taken from several pages inside the website:

Welcome to Computers & Electronic devices

Gaming Laptops

MSI GF65 Thin 9SD-004

View details >>

Acer Predator Helios 300 Gaming Laptop

View details >>

Razer Blade 15 Gaming Laptop 2020

View details >>

Normal Laptops

Razer Blade 15 Gaming Laptop 2020

Razer Blade 15 Gaming Laptop

HP Pavilion Intel Pentium Gold 4417U

Computers & Electronice Devices Home Products Search Login

ROG Zephyrus M Thin and Portable Gaming Laptop

View details >>

MSI GL65 Leopard 10SFK-062

View details >>

Search form

Enter part of product name (or leave empty to get all products):

iPad

Choose categories:

Laptops | Desktops | Gaming | Monitors | Tablets | Computer Accessories | Networking | Computer Components | Storage & Hard Drives

Search

Apple iPad Pro (12.9-inch, Wi-Fi + Cellular, 256GB) - Space Gray (4th Generation)

View Details>>

Computers & Electronice Devices Home Products Search Login

Skytech Archangel Gaming Computer PC Desktop

Categories: Desktops | Gaming

Description:

- AMD Ryzen 5 3600 6-Core 12-Thread 3.6GHz (4.2 GHz Max Boost) CPU | 500GB SSD – Up to 30x faster than traditional HDD | B450 Motherboard
- GeForce RTX 2070 8GB GDDR6 Graphics Card (Brand May Varies) | 16GB DDR4 3000MHz Gaming Memory with Heat Spreaders | Windows 10 Home 64-bit | AMD High Performance Wraith Cooler
- 802.11AC Wi-Fi | No Bloatware | 3 x DisplayPort 1.4, 1 x HDMI | HD Audio and Mic | Free Gaming Keyboard and Mouse | 2 x USB 3.0, 2 x USB 2.0, 4 x USB 3.2 Gen1
- 3 x RGB RING Fans for Maximum Air Flow | Powered by 80 Plus Certified 500 Watt Power Supply | Skytech Archangel Gaming Case with Tempered Glass - White
- 1 Year Warranty on Parts and Labor | Lifetime Free Technical Support | Assembled in the USA | This powerful gaming PC is capable of running all your favorite games such as World of Warcraft, League of

Computers & Electronice Devices Home Products Search Login

Enter your E-mail:

eyadbere@gmail.com

Enter your password:

••••••••

Login

You're successfully loggedin.
Please wait while you're being redirected.

Computers & Electronice Devices Home Products Search Purchase Eyad Mohammed Osama Logout

HP 56 | Ink Cartridge | Black | C6656AN

Categories: Computer Accessories

Description:
HP 56 | Ink Cartridge | Black | C6656AN

- HP 56 ink cartridges work with: HP Deskjet 450, 5550, 5650, 5850, 9650, 9680. HP Officejet 4215, 5610, 6110.
- HP Photosmart 7260, 7350, 7450, 7550, 7755, 7760, 7762, 7960. HP PSC 1210, 1315, 1350, 2110, 2175, 2210, 2410.
- Up to 2x more prints with Original HP ink vs refill cartridges.
- Cartridge yield (approx.): 520 pages
- Original HP ink cartridges: genuine ink for your HP printer.
- What's in the box: 1 New Original HP 56 ink cartridge (C6656AN)
- Color: Black

Price: 39$ Add to cart

Profile of user (Eyad Mohammed Osama)

General info

Username:	Eyad Mohammed Osama
E-mail	eyadbere@gmail.com
Balance:	75718$

Purchases history

Product name	Quantity	Unit price	Total price	Datetime	Session ID
Samsung Business CH890 Series	2	1945 $	3890 $	1970/01/19 14:03:40	a8ddcc89-f5b8-4308-92c8-c8d6444d5793
Samsung 32 inch CF391 Curved Monitor (LC32F391FWNXZA)	10	39 $	390 $	1970/01/19 14:03:40	a8ddcc89-f5b8-4308-92c8-c8d6444d5793
Acer Predator XB271HU	1	1199 $	1199 $	1970/01/19 14:03:40	a8ddcc89-f5b8-4308-92c8-c8d6444d5793
Apple iPad Pro (12.9-inch, Wi-Fi + Cellular, 256GB) - Space Gray (4th Generation)	1	649 $	649 $	1970/01/19 14:03:40	a8ddcc89-f5b8-4308-92c8-c8d6444d5793
All-new Fire HD 8 tablet, 8" HD display, 32 GB, designed for portable entertainment, Black	1	1945 $	1945 $	1970/01/19 14:03:45	c1ec184b-44ea-4bec-8dc3-7c33a1fd828f

BENGO headset

Product info

Product name:	**BENGOO G9000 Stereo Gaming Headset**
Unit price:	**38$**
Quantity:	2
Total price:	**76**$

Confirm

The product has been added successfully

Close

Categories: Gaming Computer Accessories

Description:
BENGOO G9000 Stereo Gaming Headset for PS4, PC, Xbox One Controller, Noise Cancelling Over Ear Headphones with Mic, LED Light, Bass Surround, Soft Memory Earmuffs for Laptop Mac Nintendo PS3 Games

- 【MULTI-PLATFORM COMPATIBLE】Support PlayStation 4, New Xbox One, PC, Nintendo 3DS, Laptop, PSP, Tablet, iPad, Computer, Mobile Phone. Please note you need an extra Microsoft Adapter (Not Included) when connect with an old version Xbox One controller.
- 【SURROUNDING STEREO SUBWOOFER】Clear sound operating strong brass, splendid ambient noise isolation and high precision 40mm magnetic neodymium driver, acoustic positioning precision enhance the sensitivity of the speaker unit, bringing you vivid sound field, sound clarity, shock feeling sound. Perfect for various games like Halo 5 Guardians, Metal Gear Solid, Call of Duty, Star Wars Battlefront, Overwatch, World of Warcraft Legion, etc.
- 【NOISE ISOLATING MICROPHONE】Headset integrated onmi-directional microphone can transmits high quality communication with its premium noise-concellng feature, can pick up sounds with great sensitivity and remove the noise, which enables you clearly deliver or receive messages while you are in a game. Long flexible mic design very convenient to adjust angle of the microphone.
- 【GREAT HUMANIZED DESIGN】Superior comfortable and good air permeability protein over-ear pads, muti-points headbeam, acord with human body engineering specification can reduce hearing impairment and heat sweat.Skin friendly leather material for a longer period of wearing. Glaring LED lights desigend on the earcups to highlight game atmosphere.
- 【EFFORTLESSLY VOLUME CONTROL】High tensile strength, anti-winding braided USB cable with rotary volume controller and key microphone mute effectively prevents the 49-inches long cable from twining and allows you to control the volume easily and mute the mic as effortless volume control one key mute.

Price: 38$ Add to cart

Check the items you want to purchase

Product name	Quantity	Unit price	Total price	
ROG Zephyrus M Thin and Portable Gaming Laptop	1	1945	1945	Remove ✕
BENGOO G9000 Stereo Gaming Headset	2	38	76	Remove ✕
Acer Predator XB271HU	1	649	649	Remove ✕
Apple iPad Pro (12.9-inch, Wi-Fi + Cellular, 256GB) - Space Gray (4th Generation)	3	1199	3597	Remove ✕

Purchase ✔

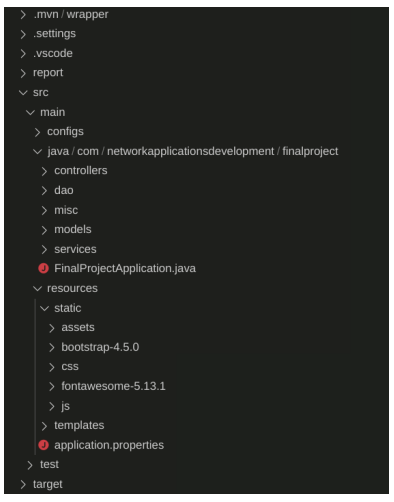
Computers & Electronice Devices Home Products Search Purchase Eyad Mohammed Osama Logout

Showing products for category (Desktops)

CybertronPC Patriot Gaming Desktop - AMD A4-6300

View details >>

The project has the following structure:

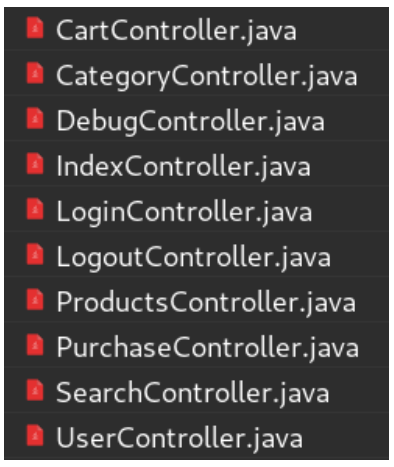


- **Controllers folder:** used to hold the buisness logic layer components
- **DAO folder:** used to provide the functionality of data access layer & CRUD operations
- **Misc folder:** used to hold some helpful classes
- **Models folder:** contains modeling for database tables as classes
- **Service folder:** provides the functionality of service layer components
- **Resources folder:** contains two important folders:
  - **Static folder:** contains sub-folders for external images and CSS and JS files
  - **Templates folder:** storage of templates that gets rendered by the controllers

---

Controllers

We have the following controllers in our project:



Let's talk briefly about the functionalities:

- **CartController.java:** addition & removal of items in shopping cart
- **CategoryController.java:** view products that falls under a specific category
- **DebugController.java:** used during development to print user info and sessions
- **IndexController.java:** the controller that renders the home page
- **LoginController.java:** provides login logic and the required session parameters
- **LogoutController.java:** invalidates and destroys sessions on demand
- **ProductsController.java:** view all products or view a product by its name
- **PurchaseController.java:** view purchasing page and contains purchasing logic
- **SearchController.java:** view search page and performs search
- **UserController.java:** view user profile

The **CartController** accepts requests for addition or removal of items through POST requests and directly removes it from the session:

```java
package com.networkapplicationsdevelopment.finalproject.controllers;

import java.util.HashMap;

import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.networkapplicationsdevelopment.finalproject.misc.CartItem;
import com.networkapplicationsdevelopment.finalproject.models.Product;
import com.networkapplicationsdevelopment.finalproject.services.ProductService;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
@RequestMapping("/cart")
public class CartController {
    @Autowired
    private ProductService productService;

    @PostMapping(value = "/add", produces = "application/json")
    @ResponseBody
    @SuppressWarnings("unchecked")
    private String addProduct(@RequestParam("product_id") int product_id,
                              @RequestParam("quantity") int quantity,
                               HttpSession session, HttpServletResponse response) throws JsonProcessingException {
```

```
        // Check whether the user is logged in
        if (session.getAttribute("uid") == null) {
            response.setHeader("location", "/");
            response.setStatus(302);
        }
        HashMap<String, Object> finalResponse = new HashMap<String, Object>();

        if (!productService.productExists(product_id)) {
            finalResponse.put("error", "The product you're trying to add doesn't exist in database");
            finalResponse.put("isOkay", false);
        }
        else {
            Product product = productService.getProductById(product_id);
            if (session.getAttribute("items") == null) {
                session.setAttribute("items", new HashMap<Integer, CartItem>());
            }

            CartItem item = new CartItem(product, quantity);
            HashMap<Integer, CartItem> items = (HashMap<Integer, CartItem>)session.getAttribute("items");
            items.put(product_id, item);
            session.setAttribute("items", items);

            finalResponse.put("isOkay", true);
        }

        return new ObjectMapper().writeValueAsString(finalResponse).toString();
    }

    @PostMapping(value = "/remove", produces = "application/json")
    @ResponseBody
    @SuppressWarnings("unchecked")
    private String removeProduct(@RequestParam("product_id") int product_id,
                                 HttpSession session,
                                 HttpServletResponse response) throws JsonProcessingException {
        if (session.getAttribute("uid") == null) {
            response.setHeader("location", "/");
            response.setStatus(302);
        }

        HashMap<String, Object> finalResponse = new HashMap<String, Object>();

        if (!productService.productExists(product_id)) {
            finalResponse.put("error", "The product you're trying to remove doesn't exist in database");
            finalResponse.put("isOkay", false);
        }
        else {
            if (session.getAttribute("items") != null) {
                HashMap<Integer, CartItem> items = (HashMap<Integer, CartItem>) session.getAttribute("items");
                if (items.get(product_id) != null) {
                    items.remove(product_id);
                    if (items.size() == 0) {
                        items = null;
                    }
                    session.setAttribute("items", items);
                    finalResponse.put("isOkay", true);
                }
                else {
                    finalResponse.put("error", "The product you're trying to remove doesn't exist in the cart");
                    finalResponse.put("isOkay", false);
                }
            }
        }

        return new ObjectMapper().writeValueAsString(finalResponse).toString();
    }

    @PostMapping(value = "/retrieve", produces = "application/json")
    @ResponseBody
    @SuppressWarnings("unchecked")
    private String getProducts(HttpServletResponse response, HttpSession session) throws JsonProcessingException {
        HashMap<Integer, CartItem> items = (HashMap<Integer, CartItem>)session.getAttribute("items");
        return new ObjectMapper().writeValueAsString(items);
    }
}
```

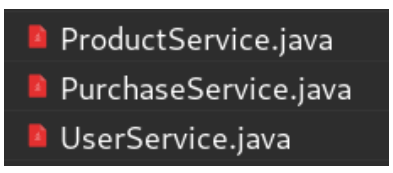
The **LoginController** contacts indirectly with the database to verify whether the user credentials is correct or not:

```
package com.networkapplicationsdevelopment.finalproject.controllers;

import java.util.HashMap;

import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.networkapplicationsdevelopment.finalproject.models.User;
import com.networkapplicationsdevelopment.finalproject.services.UserService;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
@RequestMapping("/login")
public class LoginController {
    @Autowired
    private UserService userService;

    @GetMapping(value = { "", "/" })
    private String viewLogin(Model model, HttpSession session, HttpServletResponse response) {
        if (session.getAttribute("uid") != null) {
            response.setHeader("location", "/");
            response.setStatus(302);
        }
```

```
        model.addAttribute("title", "Login to your account");
        model.addAttribute("loggedIn", (session.getAttribute("uid") != null));
        model.addAttribute("content", "Login");
        return "Layout";
    }

    @PostMapping(value = { "", "/" }, produces = "application/json")
    @ResponseBody
    private String verifyLogin(Model model, HttpSession session, HttpServletResponse response,
        @RequestParam("email") String email, @RequestParam("password") String password)
        throws JsonProcessingException {
        if (session.getAttribute("uid") != null) {
            response.setHeader("location", "/");
            response.setStatus(302);
        }

        HashMap responseProperties = new HashMap();

        boolean userExists = userService.userExists(email, password);
        responseProperties.put("successful", userExists);
        if (userExists) {
            User user = userService.getUserByEmail(email);
            responseProperties.put("email", email);
            session.setAttribute("uid", user.getId());
            session.setAttribute("username", user.getUsername());
        }

        return new ObjectMapper().writeValueAsString(responseProperties);
    }
}
```

Services

We have the following services:



Let's talk briefly about the functionalities:

- **ProductService:** assist SearchController and ProductsController and IndexController in functionalities related to products
- **PurchaseService:** transactions related to purchasing operations
- **UserService:** several methods to query for users

That's how **ProductService** works:

```
package com.networkapplicationsdevelopment.finalproject.services;

import java.util.ArrayList;
import java.util.List;

import javax.transaction.Transactional;

import com.networkapplicationsdevelopment.finalproject.dao.CategoryDAO;
import com.networkapplicationsdevelopment.finalproject.dao.ProductCategoryDAO;
import com.networkapplicationsdevelopment.finalproject.dao.ProductDAO;
import com.networkapplicationsdevelopment.finalproject.models.Category;
import com.networkapplicationsdevelopment.finalproject.models.Product;

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

@Service
@Transactional
public class ProductService {
    @Autowired
    private ProductDAO productDAO;

    @Autowired
    private CategoryDAO categoryDAO;

    @Autowired
    private ProductCategoryDAO productCategoryDAO;

    public Product getProductById(int product_id) {
        return productDAO.retrieve(product_id);
    }

    public List<Product> getProducts() {
        return productDAO.retrieveAll();
    }

    public List<Product> getProducts(int limit) {
        return productDAO.retrieveLimit(limit);
    }

    public List<Product> getProductsByCategory(String category_name) {
        int category_id = categoryDAO.retrieve(category_name).getId();
        List<Integer> pids = productCategoryDAO.retrieveProducts(category_id);
        List<Product> products = new ArrayList<Product>();
        for (int i = 0; i < pids.size(); i++) {
            Product product = productDAO.retrieve(pids.get(i));
            products.add(product);
        }
        return products;
    }

    public List<Product> getProductsByCategory(int category_id, int limit) {
```

```
        return productDAO.retrieveLimit(category_id, limit);
    }

    public List<Product> getProductsByCategories(List<Integer> categories, int limit) {
        List<Product> products = new ArrayList<Product>();

        List<List<Integer>> pidsByCategories = new ArrayList<>();
        for (int i = 0; i < categories.size(); i++) {
            List<Integer> matchingProducts = productCategoryDAO.retrieveProducts(categories.get(i));
            pidsByCategories.add(matchingProducts);
        }

        List<Integer> allPids = pidsByCategories.get(0);
        for (int i = 1; i < pidsByCategories.size(); i++) {
            allPids.retainAll(pidsByCategories.get(i));
        }

        for (int i = 0; i < allPids.size(); i++) {
            int pid = allPids.get(i);
            Product product = productDAO.retrieve(pid);
            products.add(product);
        }

        int size = products.size();
        if (size > limit) {
            products = products.subList(size - 1 - limit, size - 1);
        }
        return products;
    }

    public List<Product> getProductsByIds(List<Integer> pids) {
        return productDAO.retrieveAll(pids);
    }

    public List<Product> getProductsByName(String name) {
        return productDAO.retrieve(name);
    }

    public List<Product> getProductsByNameAndCategories(String name, List<Integer> categories) {
        List<Product> products = new ArrayList<Product>();

        List<Product> productsByName = productDAO.retrieve(name);
        List<Integer> pidsByName = new ArrayList<Integer>();
        for (int i = 0; i < productsByName.size(); i++) {
            pidsByName.add(productsByName.get(i).getId());
        }
        List<List<Integer>> pidsByCategories = new ArrayList<>();
        for (int i = 0; i < categories.size(); i++) {
            List<Integer> matchingProducts = productCategoryDAO.retrieveProducts(categories.get(i));
            pidsByCategories.add(matchingProducts);
        }

        List<Integer> allPids = pidsByName;
        for (int i = 0; i < pidsByCategories.size(); i++) {
            allPids.retainAll(pidsByCategories.get(i));
        }

        for (int i = 0; i < allPids.size(); i++) {
            int pid = allPids.get(i);
            Product product = productDAO.retrieve(pid);
            products.add(product);
        }

        return products;
    }

    public List<Category> getCategories() {
        return categoryDAO.retrieveAll();
    }

    public List<Category> getCategoriesByProductId(int product_id) {
        return productCategoryDAO.retrieveCategories(product_id);
    }

    public List<Category> getCategoriesByProductName(String product_name) {
        Product product = productDAO.retrieve(product_name).get(0);
        return productCategoryDAO.retrieveCategories(product.getId());
    }

    public boolean productExists(String product_name) {
        return productDAO.exists(product_name);
    }

    public boolean productExists(int product_id) {
        return (productDAO.retrieve(product_id) != null);
    }
}
```

It would be interesting to see how **PurchaseService** works:

```
package com.networkapplicationsdevelopment.finalproject.services;

import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.HashMap;
import java.util.List;

import javax.transaction.Transactional;

import com.networkapplicationsdevelopment.finalproject.dao.ProductDAO;
import com.networkapplicationsdevelopment.finalproject.dao.PurchaseDAO;
import com.networkapplicationsdevelopment.finalproject.dao.UserDAO;
import com.networkapplicationsdevelopment.finalproject.misc.CartItem;
import com.networkapplicationsdevelopment.finalproject.models.Product;
import com.networkapplicationsdevelopment.finalproject.models.Purchase;
```

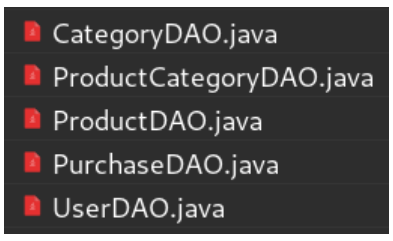
```java
import com.networkapplicationsdevelopment.finalproject.models.User;

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

@Service
@Transactional
public class PurchaseService {
    @Autowired
    private PurchaseDAO purchaseDAO;

    @Autowired
    private UserDAO userDAO;

    @Autowired
    private ProductDAO productDAO;

    @Transactional
    public boolean makePurchase(int uid, String session_id, ArrayList<CartItem> items) {
        User user = userDAO.retrieve(uid);
        int currentBalance = userDAO.retrieve(uid).getBalance();
        int totalPrice = 0;
        for (int i = 0; i < items.size(); i++) {
            totalPrice += items.get(i).getTotalPrice();
        }
        if (totalPrice > currentBalance) {
            return false;
        }
        int newBalance = currentBalance - totalPrice;
        purchaseDAO.create(uid, newBalance, session_id, items);
        user.setBalance(newBalance);
        userDAO.update(user);
        return true;
    }

    public List<HashMap<String, Object>> getPurchases(int uid) {
        List<HashMap<String, Object>> result = new ArrayList<HashMap<String, Object>>();
        List<Purchase> purchases = purchaseDAO.retrieve(uid);

        for (int i = 0; i < purchases.size(); i++) {
            Purchase purchase = purchases.get(i);
            int product_id = purchase.getId();
            Product product = productDAO.retrieve(product_id);
            Date productPurchaseDatetime = new Date(purchase.getTimestamp());
            DateFormat formattedDate = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");

            HashMap<String, Object> temp = new HashMap<String, Object>();
            temp.put("productName", product.getName());
            temp.put("productQuantity", purchase.getQuantity());
            temp.put("productUnitPrice", purchase.getPrice() / purchase.getQuantity());
            temp.put("productTotalPrice", purchase.getPrice());
            temp.put("productPurchaseDatetime", formattedDate.format(productPurchaseDatetime));
            temp.put("sessionId", purchase.getSessionId());
            result.add(temp);
        }

        return result;
    }
}
```

The service layer works in the middle between data access layer and buisness logic layer

---

Data Access Objects

We have the following data access objects classes:



Let's take a look at the contents of **ProductDAO** class:

```java
package com.networkapplicationsdevelopment.finalproject.dao;

import java.util.ArrayList;
import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.transaction.Transactional;

import org.springframework.stereotype.Repository;

import com.networkapplicationsdevelopment.finalproject.models.Product;
import com.networkapplicationsdevelopment.finalproject.models.ProductCategory;

@Repository
@Transactional
public class ProductDAO {
    @PersistenceContext
    private EntityManager entityManager;

    public void create(Product product) {
        entityManager.persist(product);
    }

```

```
    public Product retrieve(int id) {
        return (Product)entityManager.find(Product.class, id);
    }

    @SuppressWarnings("unchecked")
    public List<Product> retrieve(String name) {
        return entityManager.createQuery("FROM Product WHERE name LIKE CONCAT('%', :name, '%')").setParameter("name", name).ge
    }

    @SuppressWarnings("unchecked")
    public List<Product> retrieveAll() {
        return entityManager.createQuery("FROM Product").getResultList();
    }

    @SuppressWarnings("unchecked")
    public List<Product> retrieveAll(List<Integer> pids) {
        return entityManager.createQuery("FROM Product WHERE id IN :pids")
                .setParameter("pids", pids)
                .getResultList();
    }

    @SuppressWarnings("unchecked")
    public List<Product> retrieveLimit(int limit) {
        return entityManager.createQuery("FROM Product ORDER BY id DESC").setMaxResults(limit).getResultList();
    }

    @SuppressWarnings("unchecked")
    public List<Product> retrieveLimit(int category_id, int limit) {
        List<ProductCategory> productsCategories = entityManager.createQuery("FROM ProductCategory WHERE category_id = :catego
            ).setMaxResults(limit).getResultList();
        List<Integer> products_ids = new ArrayList<>();
        for (int i = 0; i < productsCategories.size(); i++) {
            products_ids.add(productsCategories.get(i).getProductId());
        }
        List<Product> products = entityManager.createQuery("FROM Product WHERE id IN :pids").setParameter("pids", products_ids
        return products;
    }

    public void update(Product product) {
        entityManager.merge(product);
    }

    public boolean delete(Product product) {
        if (entityManager.contains(product)) {
            entityManager.remove(product);
            return true;
        }
        return false;
    }

    @SuppressWarnings("unchecked")
    public boolean exists(String product_name) {
        List<Product> products = entityManager.createQuery("FROM Product WHERE name = :name").setParameter("name", product_name
        return (products.size() != 0);
    }
}
```

Also, here's the content of **PurchaseDAO** as well:

```
package com.networkapplicationsdevelopment.finalproject.dao;

import java.time.Instant;
import java.util.ArrayList;
import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.transaction.Transactional;

import com.networkapplicationsdevelopment.finalproject.misc.CartItem;
import com.networkapplicationsdevelopment.finalproject.models.Purchase;

import org.springframework.stereotype.Repository;

@Repository
@Transactional
public class PurchaseDAO {
    @PersistenceContext
    private EntityManager entityManager;

    @Transactional
    public void create(int uid, int newBalance, String session_id, ArrayList items) {
        Purchase purchase;
        CartItem item;
        int timestamp;

        for (int i = 0; i < items.size(); i++) {
            purchase = new Purchase();
            item = items.get(i);
            timestamp = (int) Instant.now().getEpochSecond();

            purchase.setUserId(uid);
            purchase.setProductId(item.getProduct().getId());
            purchase.setQuantity(item.getQuantity());
            purchase.setPrice(item.getTotalPrice());
            purchase.setTimestamp(timestamp);
            purchase.setSessionId(session_id);

            entityManager.persist(purchase);
        }
    }

    /*
    @SuppressWarnings("unchecked")
    public List<Purchase> retrieve(int product_id) {
```

```
    public List<Purchase> retrieve(int product_id) {
        return  entityManager
                .createQuery("FROM Purchase WHERE product_id = :product_id")
                .setParameter("product_id", product_id)
                .getResultList();
    }
    */

    @SuppressWarnings("unchecked")
    public List<Purchase> retrieve(int uid) {
        return entityManager
                .createQuery("FROM Purchase WHERE user_id = :user_id")
                .setParameter("user_id", uid)
                .getResultList();
    }

    @SuppressWarnings("unchecked")
    public List<Purchase> retrieve(String useremail, int product_id) {
        return entityManager
                .createQuery("FROM Purchase WHERE useremail = :useremail AND product_id = :product_id")
                .setParameter("useremail", useremail)
                .setParameter("product_id", product_id)
                .getResultList();
    }
}
```

Models

The following models exist:

Content of **Product**:

```
package com.networkapplicationsdevelopment.finalproject.models;

import java.io.Serializable;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Table;
import javax.validation.constraints.NotNull;

import com.google.gson.Gson;

import javax.persistence.Id;

@Entity
@Table(name = "products")
public class Product implements Serializable {
    /**
     *
     */
    private static final long serialVersionUID = 6652581902938778501L;

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int id;

    @NotNull
    private String name;

    @NotNull
    private int price;

    @NotNull
    private String description;

    @NotNull
    private String image;

    public Product() {
    }

    public Product(int id, String name, int price, String description, String image) {
        this.id = id;
        this.name = name;
        this.price = price;
        this.description = description;
        this.image = image;
    }

    public int getId() {
        return this.id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return this.name;
    }

    public void setName(String name) {
```

```
    public void setName(String name) {
        this.name = name;
    }

    public int getPrice() {
        return this.price;
    }

    public void setPrice(int price) {
        this.price = price;
    }

    public String getDescription() {
        return this.description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public String getImage() {
        return this.image;
    }

    public void setImage(String image) {
        this.image = image;
    }

    @Override
    public String toString() {
        return new Gson().toJson(this).toString();
    }
}
```