МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

«Харківський Політехнічний Інститут»

Кафедра «стратегічне управління»

ЗВІТ

з лабораторної роботи №5

з дисципліни

«ПРОГРАМУВАННЯ ВЕБ-ЗАСТОСУВАНЬ java ee»

ВИКОНАВ

Студент групи КН-317А

ПЕРЕВІРИла

ас. каф. СУ

Луценко С. Ю.

Харків 2020

ДОСЛІДЖЕННЯ ПРИНЦИПІВ СТВОРЕННЯ ВЕБ-ЗАСТОСУВАНЬ ДЛЯ РОБОТИ ІЗ БАЗАМИ ДАНИХ З ВИКОРИСТАННЯМ HIBERNATE

Мета роботи

1. Закріплення теоретичних відомостей про ORM та бібліотеку для його реалізації в Java.
2. Придбати навички реалізації доступу до даних, що містяться в БД, застосовуючи Hibernate.
3. Придбати навики застосування автоматичного збірника проектів maven, придбання навиків визначення залежностей для включення в проект необхідних бібліотек.
4. Придбати навички маппінгу класів для реалізації ORM, зокрема маппінгу полів для зв’язків Many-To-One та Many-To-Many.
5. Вдосконалення навиків створення веб-сторінок та застосування JSTL при їх розробці.
6. Вдосконалення навиків розробки user frendly applications.
7. Вдосконалення навиків створення сервлетів.

Завдання роботи

**Очікуємий результат** – веб-застосування, яке реалізує операції CRUD із таблицями клієнт-серверної бази даних, в якій присутні принаймні дві таблиці, при цьому одна з них є основною, а інша залежною від неї (ГРУПА \_- СТУДЕНТИ, СТРАНИ – МІСТА, МІСТО – ГОТЕЛІ і т.і.). При цьому робота із даними реалізована у проекті із застосуванням можливостей бібліотеки Hibernate.

**Кроки роботи.**

1 Модифікувати структуру БД, додавши в неї ще одну таблицю, яка має бути залежною від таблиці, яка вже присутня у базі даних (або навпаки – додати таблицю, яка може стати головною для існуючої. Однак у цьому випадку будуть потрібні зміни у класи через появу поля для встановлення зовнішнього зв’язку із записами у головній таблиці). Створити необхідні класи для зберігання нових даних у додатку.

Ще раз необхідно звернути увагу – ***обов’язкова наявність звязку master-detail між таблицями.***

2. Додати до web-проекту можливості автоматичної збірки maven.  
У файл pom.xml додати залежності для роботи із БД, із сервлетами, із EL та JSTL, а також Hіbernate та для використання JPA.

3. Виконати мапінг класів для зберігання даних анотаціями JPA.

4. Переробити методи доступу до даних, використовуючи принципи доступу технології Hibernate. При цьому, ***звести застосування мови HQL до мінімуму***, надавши перевагу методам класів **Criteria** та **Session**.

5. Допрацювати функціонал проекту, отриманого на ЛР №4, додавши веб-сторінки, створені як JSP, для заповнення нових таблиць.

В ЛР №4 реалізований функціонал для відображення, а також додавання, редагування та видалення даних, що містяться в одній таблиці. В ЛР №5 аналогічний функціонал необхідно реалізувати для другої таблиці бази даних. Вимогами до створення сторінок, як і в попередній роботі, є такі:

1 JSP мають бути створені із застосуванням JSTL.

2 Реалізація програмного контролю даних, що вводяться, обов’язкова.

3 На сторінці відображення переліку записів таблиці має бути обов’язково реалізований пошук даних за значеннями полів семантичного ключа (не по id і не пошук на співпадіння значення у комірках засобами браузера).

Додатково на веб-сторінці, що призначена для відображення даних залежної таблиці необхідно реалізувати наступний функціонал:

1 На веб-сторінці має бути реалізована можливість фільтрації даних. Наприклад, якщо є таблиці ГРУПИ та СТУДЕНТИ, то фільтр на сторінці студенти має дозволити вибрати групу, наприклад, зі спадаючого списку, після чого на екрані відображуються тільки дані про студентів обраної групи.

2 Реалізувати можливість додавання, редагування, вилучення даних у підлеглу таблицю для варіанту повного виводу та для варіанту, коли застосований фільтр.

Знову слід звернути увагу, що для фільтрації, пошуку, зміни та вилучення не повинні використовуватись первинні ключи при визначенні умов – тільки семантичні ключі.

Додатково продумати реалізацію вилучення записів:

1 При вилученні записів система обов’язково повинна видавати запит на підтвердження вилучення.

2 При вилученні записів із зв’язаних таблиць передбачити обробку ситуації, коли у підлеглій таблиці є записи, що мають зв’язки із записом, що вилучається з головної таблиці. В даному випадку можна:

* заборонити вилучення та запропонувати вилучити необхідні записи із підлеглої таблиці;
* реалізувати каскадне вилучення записів (необхідний режим транзакції). Цей варіант простіший в реалізації, але дуже проблемний для користувачів у плані наслідків втрати інформації.

Хід виконання

**Завдання 1.** Модифікувати структуру БД, додавши в неї ще одну таблицю.

Попередня структура бази даних була оновлена додаванням нової таблиці Groups з продуктами. Ця таблиця має відношення OneToMany з підлеглою їй таблицею Products, що існувала раніше. Ця таблиця дозволить відносити продукти до категорій (овочі, фрукти тощо). Оновлену структуру можна побачити на рисунку 1.

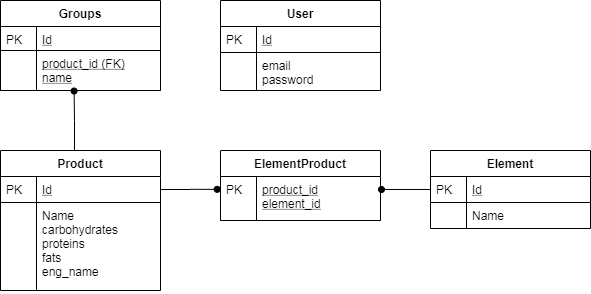


Рисунок 1 – Структура бд

Для двонаправленого доступу до даних у клас Product було додано поле group типу Group.

|  |
| --- |
| @ManyToOne  **private** Group group = **null**; |

До классу Group була додана колекція products

|  |
| --- |
| @OneToMany(cascade = CascadeType.***ALL***)  Collection<Product> products = **new** ArrayList<>(); |

**Завдання 2.** Додати до web-проекту можливості автоматичної збірки maven. У файл pom.xml додати залежності для роботи із БД, із сервлетами, із EL та JSTL, а також Hіbernate та для використання JPA.

Для виконання цієї роботи був створений новій проект Maven з архітипом (зразком проекту) maven-archetype-webapp. Потім до файлу pom.xml були додані залежності hibernate-core, mysql-connector-java, tomcat-embed-core, avax.servlet-api, jsp-api, jstl. Структуру pom.xml наведено у лістингу 1.

Лістинг 1 – структура pom.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <groupId>com.javaee.maven</groupId>  <artifactId>maven\_tomcat</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>war</packaging>  <name>maven\_tomcat Maven Webapp</name>  <!-- FIXME change it to the project's website -->  <url>http://www.example.com</url>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <maven.compiler.source>1.8</maven.compiler.source>  <maven.compiler.target>1.8</maven.compiler.target>  </properties>  <dependencies>  <dependency>  <groupId>org.hibernate</groupId>  <artifactId>hibernate-core</artifactId>  <version>5.4.12.Final</version>  </dependency>    <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>8.0.19</version>  </dependency>    <dependency>  <groupId>org.apache.tomcat.embed</groupId>  <artifactId>tomcat-embed-core</artifactId>  <version>9.0.33</version>  </dependency>    <dependency>  <groupId>javax.servlet</groupId>  <artifactId>javax.servlet-api</artifactId>  <version>3.0.1</version>  <scope>provided</scope>  </dependency>    <!-- https://mvnrepository.com/artifact/javax.servlet/jsp-api -->  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>jsp-api</artifactId>  <version>2.0</version>  <scope>provided</scope>  </dependency>  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>jstl</artifactId>  <version>1.2</version>  </dependency>  </dependencies>  <build>  <finalName>maven\_tomcat</finalName>  <pluginManagement><!-- lock down plugins versions to avoid using Maven defaults (may be moved to parent pom) -->  <plugins>  <plugin>  <artifactId>maven-clean-plugin</artifactId>  <version>3.1.0</version>  </plugin>  <!-- see http://maven.apache.org/ref/current/maven-core/default-bindings.html#Plugin\_bindings\_for\_war\_packaging -->  <plugin>  <artifactId>maven-resources-plugin</artifactId>  <version>3.0.2</version>  </plugin>  <plugin>  <artifactId>maven-compiler-plugin</artifactId>  <version>3.8.0</version>  </plugin>  <plugin>  <artifactId>maven-surefire-plugin</artifactId>  <version>2.22.1</version>  </plugin>  <plugin>  <artifactId>maven-war-plugin</artifactId>  <version>3.2.2</version>  </plugin>  <plugin>  <artifactId>maven-install-plugin</artifactId>  <version>2.5.2</version>  </plugin>  <plugin>  <artifactId>maven-deploy-plugin</artifactId>  <version>2.8.2</version>  </plugin>  </plugins>  </pluginManagement>  </build>  </project> |

Структуру проекта можна побачити на рисунку 2.

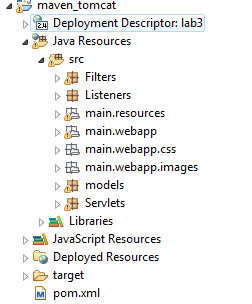


Рисунок 2 – Структура проекту

**Завдання 3.** Виконати маппінг класів для зберігання даних анотаціями JPA.

Використання анотацій для класу Product.java наведено у лістингу 2.

Лістинг 2 – Product.java

|  |
| --- |
| @Entity (name = "Products")  **public** **class** Product {  @ManyToOne  **private** Group group = **null**;  **private** String Name;  @Id  @GeneratedValue(strategy = GenerationType.***AUTO*** )  **private** **int** id;  @ManyToMany(mappedBy = "products")  **private** Collection<Element> elements = **new** ArrayList<>();  **private** String eng\_name;  **private** **double** proteins;  **private** **double** fats;  **private** **double** carbohydrates;  //getters and setters…  } |

Використання анотацій для класу Group.java наведено у лістингу 3.

Лістинг 3 – Group.java

|  |
| --- |
| @Entity (name = "FoodGroups")  **public** **class** Group {  @Id  @GeneratedValue(strategy = GenerationType.***AUTO*** )  **private** **int** group\_id;  @OneToMany(cascade = CascadeType.***ALL***)  Collection<Product> products = **new** ArrayList<>();  **private** String name;  **private** String eng\_name;  } |

**Завдання 4.** Переробити методи доступу до даних, використовуючи принципи доступу технології Hibernate.

Під час запуску сервера до сервлетів вводиться залежність типу dbOperations. Для цього було створено клас ContextListener (з анотацією @WebListener) та перевантажений метод contextInitialized(). Реалізацію вводу залежності до сервлетів – лістинг 4.

Лістинг 4 – ContextListener.java

|  |
| --- |
| @WebListener  **public** **class** ContextListener **implements** ServletContextListener{  @Override  **public** **void** contextInitialized(ServletContextEvent event) {  DbOperations hibernateHelper = **new** HibernateHelper();    System.***out***.println("Injecting hibernateHelper to servlets");  DeleteElementServlet.*setDbHelper*(hibernateHelper);  ElementCreateUpdate.*setDbHelper*(hibernateHelper);  ListElementsServlet.*setDbHelper*(hibernateHelper);  LoginServlet.*setDbHelper*(hibernateHelper);  RegisterServlet.*setDbHelper*(hibernateHelper);  DeleteProductServlet.*setDbHelper*(hibernateHelper);  ListProductsServlet.*setDbHelper*(hibernateHelper);  ProductCreateEditServlet.*setDbHelper*(hibernateHelper);  ListGroupsServlet.*setDbHelper*(hibernateHelper);  GroupCreateUpdate.*setDbHelper*(hibernateHelper);  DeleteGroupServlet.*setDbHelper*(hibernateHelper);  SearchGroupsServlet.*setDbHelper*(hibernateHelper);  SearchProductsServlet.*setDbHelper*(hibernateHelper);    event.getServletContext().setAttribute("groups", hibernateHelper.getGroups());  }  } |

Методі інтерфейсу dbOperations – лістинг 5.

Лістинг 5 – dbOperations.java

|  |
| --- |
| **public** **interface** DbOperations {  **public** Element updateElement(Element element);  **public** Element deleteElement(**int** id);  **public** List<Element> getElements();  **public** Element addElement(Element element);  **public** Product updateProduct(Product product);  **public** Product deleteProduct(**int** id);  **public** List<Product> getProducts();  **public** Product addProduct(Product product);  **public** List<User> getUsers();  **public** User addUser(User user);  **public** List<Group> getGroups();  **public** **void** addGroup(Group group);  **public** **void** updateGroup(Group group);  **public** **void** deleteGroup(**int** id);  **public** List<Product> getProductsByGroupId(String parameter);  **public** List<Product> getProductsByGroupName(String parameter);  **public** List<Group> findGroupsByText(String text);  **public** List<Product> findProductsByText(String text\_to\_find);  } |

Приклад звертання до бд за допомогою Criteria API для деяих методів – лістинг 6.

Лістинг 6 – робота з бд (клас HibernateHelper.java)

|  |
| --- |
| **public** **class** HibernateHelper **implements** DbOperations{  **private** Session session = **null**;  **private** CriteriaBuilder cb = **null**;  **private** **final** SessionFactory sessionFactory = **new** Configuration().configure().buildSessionFactory();  @Override  **public** List<Product> findProductsByText(String text) {  session = sessionFactory.openSession();  session.beginTransaction();    cb = session.getCriteriaBuilder();  CriteriaQuery<Product> cr = cb.createQuery(Product.**class**);  Root<Product> root = cr.from(Product.**class**);  cr.select(root).where(cb.like(cb.upper(root.get("Name")), "%" + text.toUpperCase() + "%"));    Query<Product> query = session.createQuery(cr);  List<Product> results = query.getResultList();  session.getTransaction().commit();  session.close();  **return** results;  }  @Override  **public** List<Product> getProductsByGroupId(String groupId) {  session = sessionFactory.openSession();  session.beginTransaction();    cb = session.getCriteriaBuilder();  CriteriaQuery<Product> cr = cb.createQuery(Product.**class**);  Root<Product> root = cr.from(Product.**class**);  cr.select(root).where(cb.equal(root.join("group").get("group\_id"), Integer.*parseInt*(groupId)));    Query<Product> query = session.createQuery(cr);  List<Product> results = query.getResultList();  session.getTransaction().commit();  session.close();  **return** results;  }  @Override  **public** Product deleteProduct(**int** id) {  session = sessionFactory.openSession();  session.beginTransaction();  Product product = (Product) session.load(Product.**class**, id);  session.delete(product);    session.getTransaction().commit();  session.close();  **return** product;  }  **public** List<Product> getProducts() {  session = sessionFactory.openSession();  session.beginTransaction();    cb = session.getCriteriaBuilder();  CriteriaQuery<Product> cr = cb.createQuery(Product.**class**);  Root<Product> root = cr.from(Product.**class**);  cr.select(root);    Query<Product> query = session.createQuery(cr);  List<Product> products = query.getResultList();    **for** (Product p : products) {  Hibernate.*initialize*(p.getElements());  Hibernate.*initialize*(p.getGroup());  }  session.getTransaction().commit();  session.close();  **return** products;  }    **public** Product addProduct(Product product) {  session = sessionFactory.openSession();  session.beginTransaction();    session.save(product);    session.getTransaction().commit();  session.close();  **return** product;  }  **public** Product updateProduct(Product product) {  session = sessionFactory.openSession();  session.beginTransaction();  session.update(product);    session.getTransaction().commit();  session.close();  **return** product;  }  // other methods…  } |

**Завдання 5.** Допрацювати функціонал проекту, отриманого на ЛР №4, додавши веб-сторінки, створені як JSP, для заповнення нових таблиць.

Були створені нові сторінки для додавання, редагування (фактично додавання, редагування це одна сторінка), перечислення груп (категорій) продуктів. Їх відображення в браузері можна побачити на рисунках 3-5.

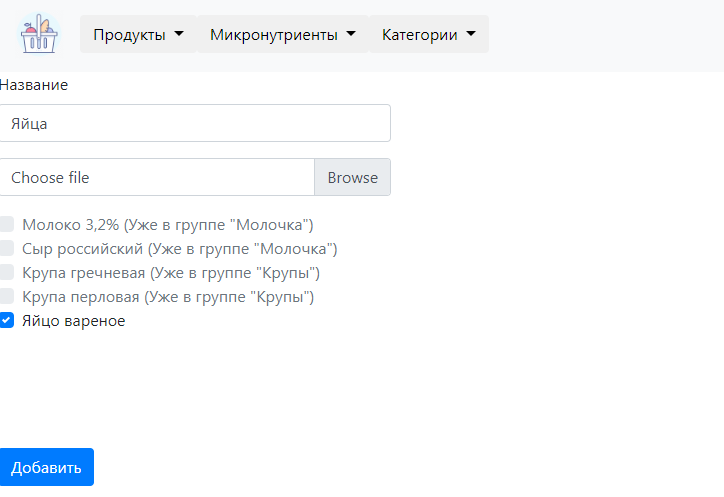


Рисунок 3 – Додавання категорії продуктів

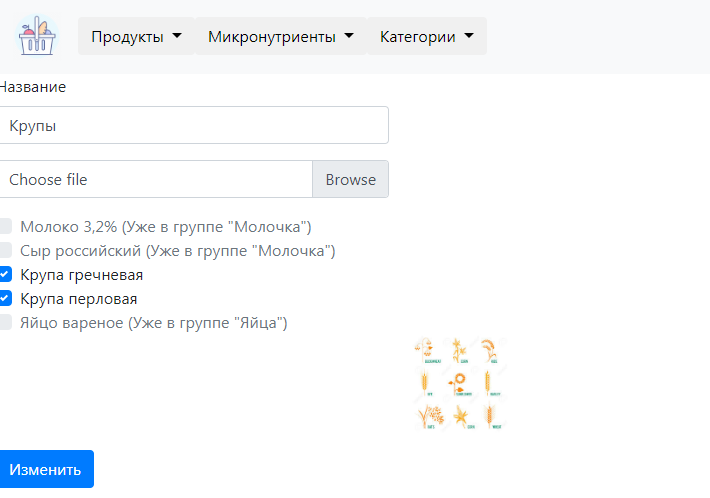


Рисунок 4 – Редагування категорії продуктів

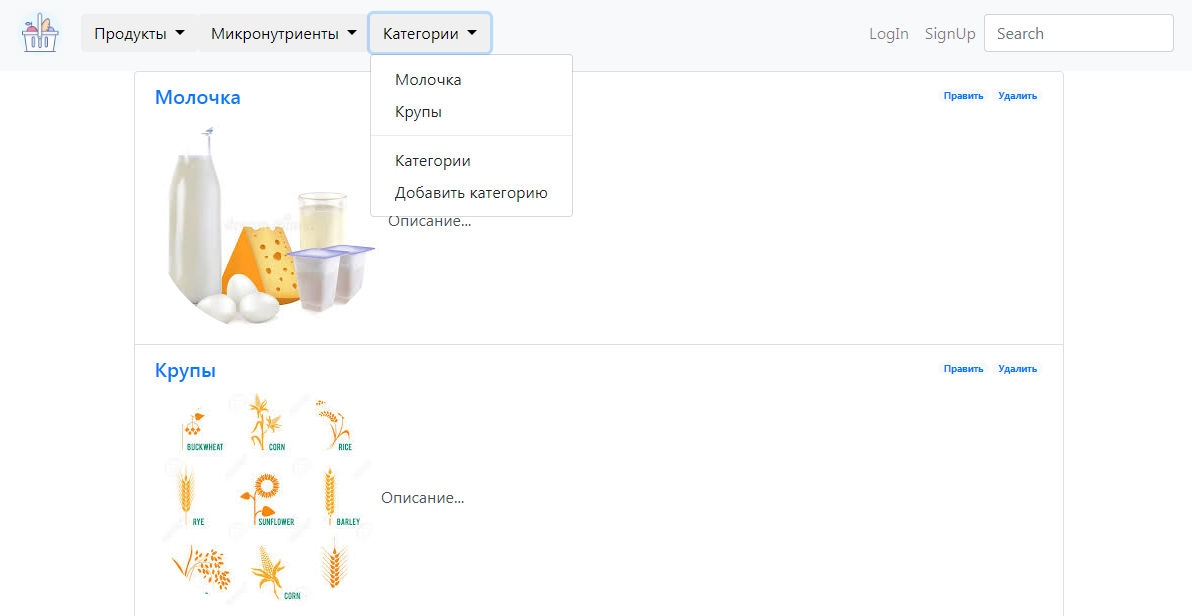


Рисунок 5 –Категорії продуктів

Приклад коду jsp сторінки ListGroups.jsp з застосуванням JSTL – лістинг 7.

Лістинг 7 – ListGroups.jsp

|  |
| --- |
| <%@taglib uri=*"http://java.sun.com/jsp/jstl/core"* prefix=*"c"* %>  <%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*  pageEncoding=*"UTF-8"*%>  <%@ include file = *"/WEB-INF/Views/Shared/Header.jsp"* %>  <div class=*"container"*>  <div class=*"list-group"*>  <!-- <div class="row"> -->  <c:forEach items = *"*${ allgroups }*"* var = *"group"*>  <div class=*"list-group-item list-group-item-action flex-column align-items-start"*>  <div class=*"d-flex w-100 justify-content-between"*>  <h5 class=*"mb-1"*><a href = *"*${pageContext.request.contextPath}*/Products?group=*${ group.getName() }*"*>${ group.getName() }</a></h5>  <small class=*"text-muted"*>  <span class=*"badge badge-light badge-pill"*><a href = *"*${pageContext.request.contextPath}*/CreateUpdateGroup?id=*${ group.getId() }*"*>Править</a></span>  <span class=*"badge badge-light badge-pill"*><a onclick = "deleteItem(this)" id = *"*${ group.getId() }*"* href = *"#"*>Удалить</a></span>  </small>  <!-- <small class="text-muted"></small> -->  </div>    <a href = *"*${pageContext.request.contextPath}*/Products?group=*${ group.getName() }*"*><img class=*"img-product"* src=*"*${pageContext.request.contextPath}*/images/*${ group.getEng\_name() }*"*></a>  Описание...  </div>  </c:forEach>  </div>  </div>  <script>  **function** deleteGroup(obj){  $(obj).parent().parent().parent().parent().remove()  $.post( "DeleteGroup?id=" + $(obj).attr("id"), **function**( data ) {  });  }  **function** deleteItem(obj) {  **if** (confirm("Удалить?")) {  deleteGroup(obj)  }  **return** **false**;  }  </script>  <%@ include file = *"/WEB-INF/Views/Shared/Footer.jsp"* %> |

На стороні користувача перевірка даних здійснена за допомогою JQuery та Bootstrap класів – рисунок 6.

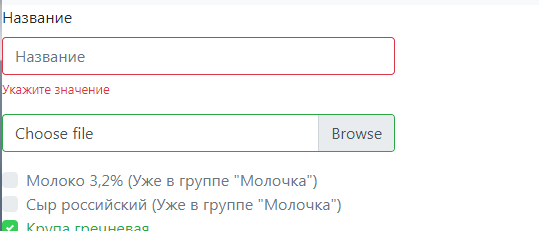


Рисунок 6 – Перевірка полів

Пошук за семантичними полями (поле ім’я таблиці Groups) здійснюється за допомогою Criteria API. Приклад використання пошуку для продуктів та категорій – рисунки 7-8.

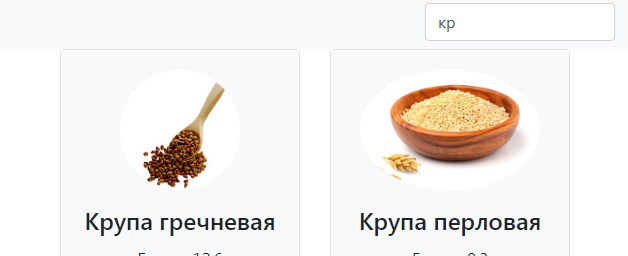


Рисунок 7 – Пошук серед продуктів

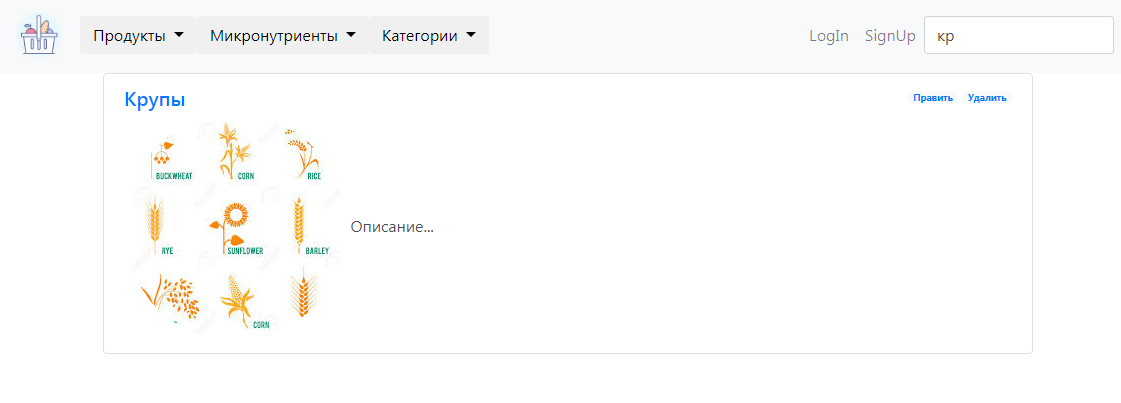


Рисунок 8 – Пошук серед категорій

Фільтрація за семантичним полем name також реалізується за допомогою Criteria API. Результат фільтрації за категоріями можна побачити на рисунку 9.

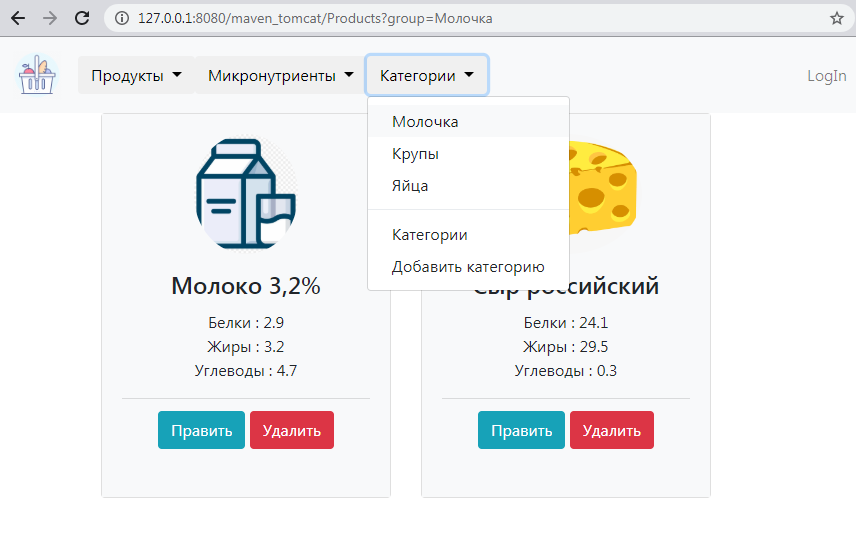


Рисунок 9 – Фільтрація за категорією «Молочка»

Видалення записів з підлеглої таблиці (таблиця Product) відбувається каскадно при видаленні запису з головної таблиці (таблиця Group). Це можливо завдяки налаштуванню cascade анотації OneToMany у класі Group.java.

|  |
| --- |
| @OneToMany(cascade = CascadeType.ALL) |

Разом з тим при видаленні будь яких даних одразу з’являється вікно з попередженням – рисунок 10.



Рисунок 10 – Спроба видалення продукту

Висновки

При виконанні цієї лабораторної роботи було розглянуте поняття ORM – Object Relational Mapping. ORM інструменти дозволяють безпечно та ефективно виконувати операції з базою даних. Найпоширенішим ORM для JAVA є Hibernate. Він реалізує інтерфейс jpa, а також додає свої анотації, дозволяє зберігати дані у кеш, пропоную свою мову запитів HQL, використовує механізми для оптимізації роботи з даними. Взагалом, це один з головних інструментів, якими повинен володіти веб розробник на java.

Використання Criteria API та Hibernate здалося складнішим ніж Linq to Sql коли вивчалася мова C#.