МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

«Харківський Політехнічний Інститут»

Кафедра «стратегічне управління»

ЗВІТ

з лабораторної роботи №3

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-ДОДАТКУ ДЛЯ МАНІПУЛЮВАННЯ ДАНИМИ НА JAVA

Мета

1. Закріплення знань принципів реалізації ООП в Java.
2. Вдосконалення навиків роботи із класами з JCF.
3. Набуттянавиків створення JSP.
4. Вдосконалення навиків роботи із файлами xml.
5. Набуття навичок розробки userfrendlyapplications.
6. Вдосконалення навиків створення сервлетів.

Завдання роботи

Розробити веб-застосування, що реалізує виведення колекції об’єктів класу, а також додавання елементів до колекції, зміну певного елементу та вилучення елементів з колекції, а також вивантаження колекції у файл (переважно – у файл типу xml) та завантаження з нього.

Хід виконання

1 Було створено клас XMLHelper, що містить у собі усі операції CRUD. Завдяки об’єктам XMLEncoder та XMLDecoder здійснюється зчитування та запис до xml файлів з відповідними колекціями. Лістинг коду XMLHelper.java наведено у лістингу 1.

Лістинг 1. XMLHelper.java

|  |
| --- |
| package models;  import java.beans.XMLDecoder;  import java.beans.XMLEncoder;  import java.io.File;  import java.io.FileInputStream;  import java.io.FileOutputStream;  import java.util.List;  import java.util.stream.Collectors;  public class XMLHelper {  private static FileOutputStream fos;  private static FileInputStream fis;  private static XMLEncoder encoder;  private static XMLDecoder decoder;    private static void writeObject(Object object, String path) {  try {  fos = new FileOutputStream(new File(path));  encoder = new XMLEncoder(fos);  encoder.writeObject(object);  encoder.close();  fos.close();  } catch (Exception e) {  e.printStackTrace();  }  }    private static Object readObject(String path) {  try {  fis = new FileInputStream(new File(path));  decoder = new XMLDecoder(fis);    Object obj = decoder.readObject();  decoder.close();  fis.close();  return obj;  } catch (Exception e) {  e.printStackTrace();  }  return null;  }    // ElementProducts CRUD  public static void updateElementProducts(List<ElementProduct> list) {  writeObject(list, "C:/Users/Reacher/eclipse-workspace/lab3/WebContent/WEB-INF/XMLdb/elementProducts.xml");  }    public static ElementProduct addElementProduct(ElementProduct elementProduct) {  List<ElementProduct> list = getElementProducts();  list.add(elementProduct);  writeObject(list, "C:/Users/Reacher/eclipse-workspace/lab3/WebContent/WEB-INF/XMLdb/elementProducts.xml");  return elementProduct;  }    public static List<ElementProduct> getElementProducts(){  List<ElementProduct> elementProduct = (List<ElementProduct>) readObject("C:/Users/Reacher/eclipse-workspace/lab3/WebContent/WEB-INF/XMLdb/elementProducts.xml");  return elementProduct;  }    // Elements CRUD  … CRUD methods for Element  // Products CRUD  … CRUD methods for Product    // Users CRUD  … CRUD methods for User    } |

2 Було створено клас MockDB.java для заповнення бази даних (xml файлів) початковими значеннями. Код класу MockDB.java наведено у лістингу 2.

Лістинг 2. MockDB.java

|  |
| --- |
| package models;  import java.beans.ExceptionListener;  import java.beans.XMLDecoder;  import java.beans.XMLEncoder;  import java.io.File;  import java.io.FileInputStream;  import java.io.FileOutputStream;  import java.util.ArrayList;  import java.util.Arrays;  import java.util.List;  public class MockDB {  public MockDB() {  }  public static void main(String[] args) {  try {  List<Product> products = new ArrayList<>();  List<Element> elements = new ArrayList<>();  List<ElementProduct> elementProducts = new ArrayList<>();  List<User> users = new ArrayList<>();    Product p1 = new Product("1", "Хлеб пшеничный", 8.1, 1, 48, "bread.png");  Product p2 = new Product("2", "Курогрудка", 23.6, 1.9, 0.4, "chicken.jpg");  Product p3 = new Product("3", "Гречка вареная", 3.6, 0.9, 17.6, "buckwheat.png");  Product p4 = new Product("4", "Орех грецкий", 15.2, 65.2, 7, "nut.jpg");    Element e1 = new Element("1", "Калории");  Element e2 = new Element("2", "Омега-3");  Element e3 = new Element("3", "Омега-6");  Element e4 = new Element("4", "Омега-9");  ElementProduct ep1 = new ElementProduct(p4, e2);  ElementProduct ep2 = new ElementProduct(p4, e3);  ElementProduct ep3 = new ElementProduct(p4, e4);    User user1 = new User("user@gmail.com", "12345");  user1.setId("1");    products.addAll(Arrays.asList(p1, p2, p3, p4));  elements.addAll(Arrays.asList(e1, e2, e3, e4));  elementProducts.addAll(Arrays.asList(ep1, ep2, ep3));  users.addAll(Arrays.asList(user1));    FileOutputStream fos = new FileOutputStream(new File("C:/Users/Reacher/eclipse-workspace/lab3/WebContent/WEB-INF/XMLdb/products.xml"));  XMLEncoder encoder = new XMLEncoder(fos);    encoder.writeObject(products);  encoder.close();  fos.close();  fos = new FileOutputStream(new File("C:/Users/Reacher/eclipse-workspace/lab3/WebContent/WEB-INF/XMLdb/elements.xml"));  encoder = new XMLEncoder(fos);  encoder.writeObject(elements);  encoder.close();  fos.close();  fos = new FileOutputStream(new File("C:/Users/Reacher/eclipse-workspace/lab3/WebContent/WEB-INF/XMLdb/elementProducts.xml"));  encoder = new XMLEncoder(fos);  encoder.writeObject(elementProducts);  encoder.close();  fos.close();    FileOutputStream fos = new FileOutputStream(new File("C:/Users/Reacher/eclipse-workspace/lab3/WebContent/WEB-INF/XMLdb/users.xml"));  XMLEncoder encoder = new XMLEncoder(fos);    encoder.writeObject(users);  encoder.close();  fos.close();    }  catch(Exception ex) {  ex.printStackTrace();  }  }} |

3 Для демонстрації було створено нові моделі, а саме Product, Element та ElementProduct що э зв’язуючею моделю для двох попередніх. Модель у нотації IDEF1X зображено на рисунку 1.

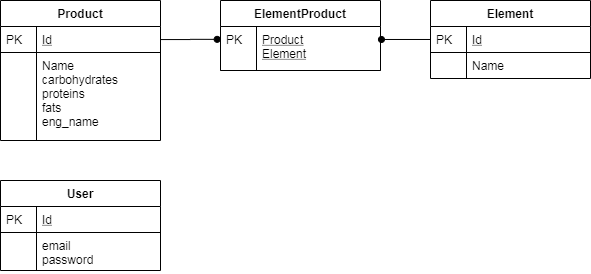


Рисунок 1 – Модель бази даних

4 Було створено відповідні класи для моделей та відповідні сервлети, що обробляють Post та Get запити та повертають JSP сторінки, що відображаються як HTML. Структуру проекту зображено на рисунку 2.

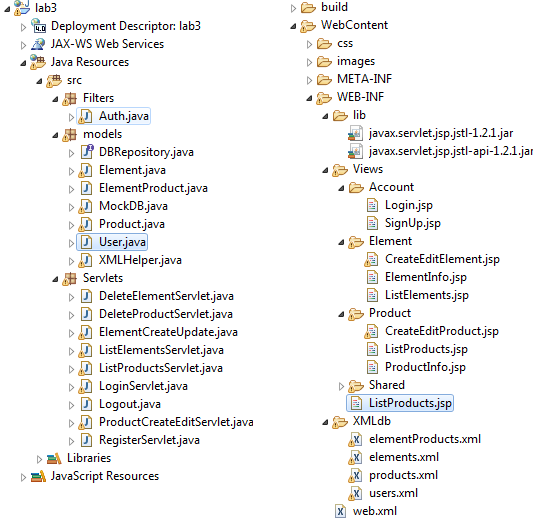


Рисунок 2 – Структура проекту

5 Для фільтрації запитів було створено Auth.java що реалізує інтерфейс Filter. Застосування цього фільтру дозволює забороняти доступ до вказаних сторінок користувачам, які не авторизувалися. Код класу Auth.java наведено у лістингу 3.

Лістинг 3. Auth.java

|  |
| --- |
| package Filters;  import java.io.IOException;  import java.io.PrintWriter;  import javax.servlet.Filter;  import javax.servlet.FilterChain;  import javax.servlet.FilterConfig;  import javax.servlet.RequestDispatcher;  import javax.servlet.ServletException;  import javax.servlet.ServletRequest;  import javax.servlet.ServletResponse;  import javax.servlet.annotation.WebFilter;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  import models.User;  @WebFilter(urlPatterns = {"/Elements", "/Products", "/CreateUpdateElement",  "/CreateUpdateProduct", "/DeleteElement", "/DeleteProduct"} )  public class Auth implements Filter {  public Auth() {  }  public void destroy() {  }  public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain) throws IOException, ServletException {  resp.setContentType("text/html; charset=UTF-8");  PrintWriter out=resp.getWriter();  HttpServletRequest httpReq = (HttpServletRequest) req;  User user = (User) httpReq.getSession().getAttribute("CURRENT\_USER");  if(user != null){  chain.doFilter(req, resp);//sends request to next resource  }  else{  HttpServletResponse httpResp = (HttpServletResponse) resp;  httpResp.sendRedirect("Login");  }  }  public void init(FilterConfig fConfig) throws ServletException {  }  } |

Якщо користувач правильно заповнює поля входу до сайту, до він додається до параметру CURRENT\_USER сесії. Ця логіку здійснюється у методі doPost сервлету LoginServlet.java. Код цього методу наведено у лістингу 4.

Лістинг 4. LoginServlet.java

|  |
| --- |
| **protected** **void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  String username = request.getParameter("username");  String pwd = request.getParameter("password");  User user = **null**;  **try**{  user = XMLHelper.*getUsers*().stream().  filter(el -> el.getEmail().equals(username) && el.getPassword().equals(pwd)).findFirst().get();  }  **catch**(NoSuchElementException ex) {  request.setAttribute("ErrorMessage", "Пользователь не найден");  request.getRequestDispatcher("WEB-INF/Views/Account/Login.jsp").forward(request, response);  }  request.getSession().setAttribute("CURRENT\_USER", user);  response.sendRedirect(request.getContextPath() + "/Products");  } |

6 Для створення JSP сторінок використовувалися як Expression Language так і Java Standard Tag Library. Зразок коду для сторінки SignUp.jsp наведено у лістингу 5.

Лістинг 5. SignUp.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"* %>  <%! **int** length = 4; %>    <div class=*"card"*>  <article class=*"card-body"*>  <h4 class=*"card-title text-center mb-4 mt-1"*>Sign up</h4>  <hr>  <form id = *"form"* class= *"needs-validation"* novalidate method = *"post"*  action = *"Signup"*>  <div class=*"form-group"*>  <div class=*"input-group"*>  <div class=*"input-group-prepend"*>  <span class=*"input-group-text"*> <i class=*"fa fa-user"*></i> </span>  </div>  <input name=*"username"* class=*"form-control"* placeholder=*"Email or login"* type=*"email"* required>  <div class=*"invalid-feedback"*>  Неверный формат email  </div>  </div> <!-- input-group.// -->  </div> <!-- form-group// -->  <div class=*"form-group"*>  <div class=*"input-group"*>  <div class=*"input-group-prepend"*>  <span class=*"input-group-text"*> <i class=*"fa fa-lock"*></i> </span>  </div>  <input  name = *"password"*  id=*"password"* class=*"form-control"* placeholder=*"\*\*\*\*\*\*"* type=*"password"*  pattern=*"(?=.\*\d)(?=.\*[a-z])(?=.\*[A-Z]).{*<%= length %>*,}"*  required>  <div class=*"invalid-feedback"*>  Должен содержать цифры, строчные и прописные буквы. Длина пароля минимум <%= length %> символа  </div>    </div> <!-- input-group.// -->  </div> <!-- form-group// -->    <div class=*"form-group"*>  <div class=*"input-group"*>  <div class=*"input-group-prepend"*>  <span class=*"input-group-text"*> <i class=*"fa fa-lock"*></i> </span>  </div>  <input  name = *"confirm\_password"*  id = *"confirm\_password"* class=*"form-control"* placeholder=*"Подтвердите пароль"* type=*"password"*  pattern=*"(?=.\*\d)(?=.\*[a-z])(?=.\*[A-Z]).{*<%= length %>*,}"*  required>  <div class=*"invalid-feedback"* id = *"message"*>  Пароли не совпадают  </div>    </div>  </div>  <div class=*"form-group"*>  <button type=*"submit"* class=*"btn btn-primary btn-block"*> Signup </button>  </div> <!-- form-group// -->  </form>  </article>  </div> <!-- card.// -->  <script>  (**function**() {  'use strict';  $('#password, #confirm\_password').on('keyup', **function** () {  **if** ($('#password').val() != $('#confirm\_password').val()){  $('#message').html('Пароли не совпадают');  $('#confirm\_password').setCustomValidity("Passwords must match");  }  **else**{  $('#message').html('');  $('#confirm\_password').setCustomValidity("");  }  });      window.addEventListener('load', **function**() {  // Fetch all the forms we want to apply custom Bootstrap validation styles to  **var** forms = document.getElementsByClassName('needs-validation');  // Loop over them and prevent submission  **var** validation = Array.prototype.filter.call(forms, **function**(form) {  form.addEventListener('submit', **function**(event) {  **if** (form.checkValidity() === **false**) {  event.preventDefault();  event.stopPropagation();    }  form.classList.add('was-validated');  }, **false**);  });  }, **false**)})()    </script>  <%@ include file = *"/WEB-INF/Views/Shared/Footer.jsp"* %> |

Вигляд сторінки реєстрації з використанням фреймворку Bootstrap наведено на рисунку 3.

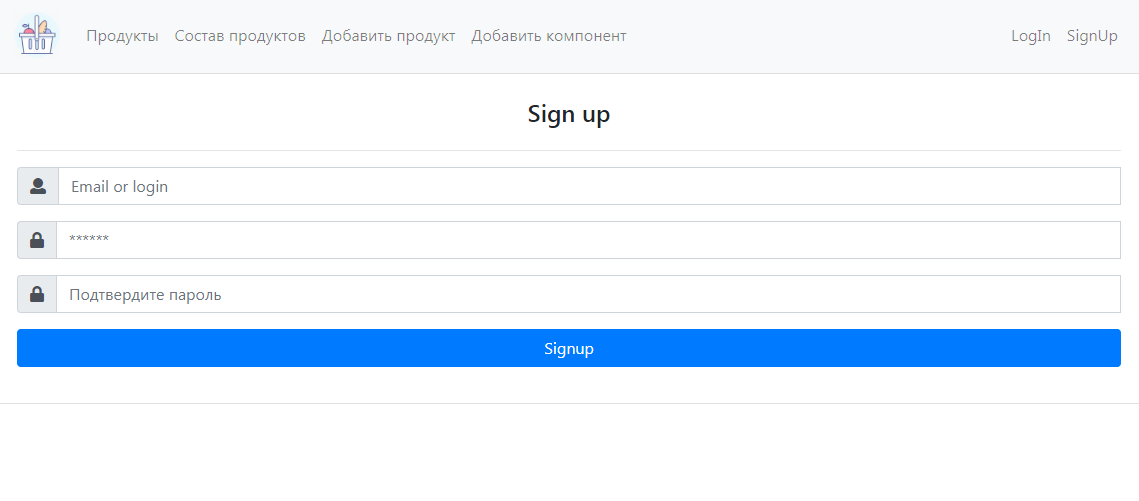


Рисунок 3 – Форма реэстрації

Вигляд інших сторінок можна побачити на рисунках 4-6.

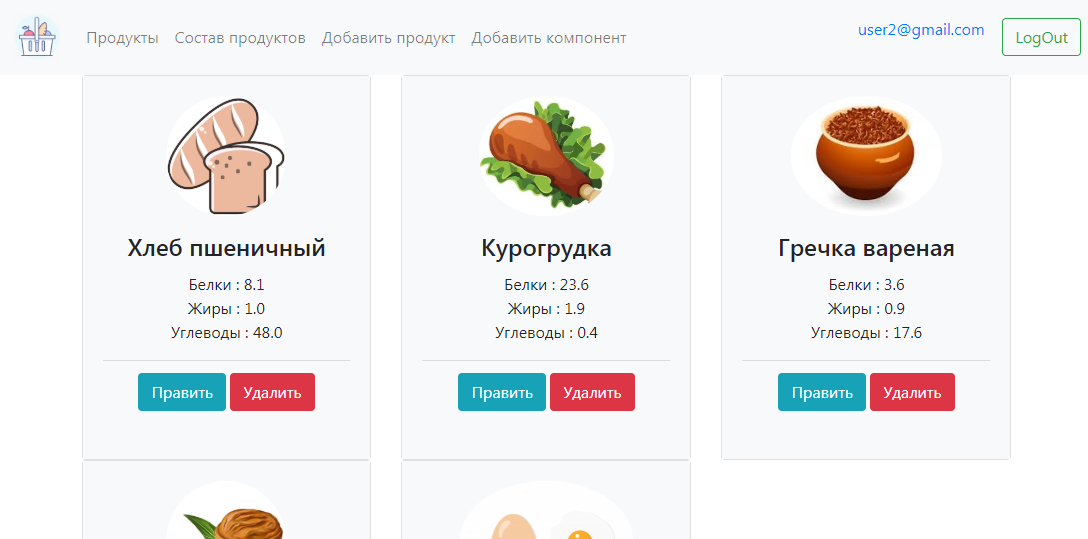


Рисунок 4 – Сторінка Продукти

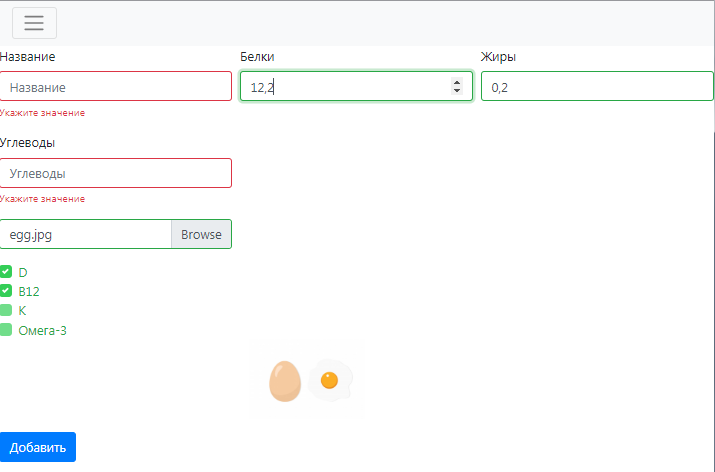


Рисунок 5 – Сторінка додавання нового продукту

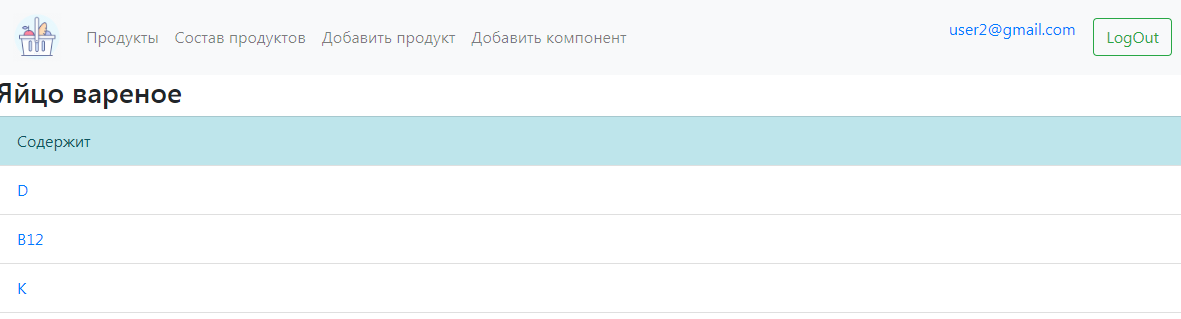


Рисунок 6 – Сторінка інформації про продукт

1. Примітки:

* для редагування продуктів використовується спільна JSP сторінка, бо поля одні й ті ж самі. Для елементів що входять до складу продукту аналогічно;
* зображення, що символізує певний продукт зберігається в теці images на сервері з ім’ям, що є трансльованим прототипом на латиницю символів кирилиці. Тобто Яблоко стає jabloko.*формат\_файлу*;
* валідація на стороні клієнта здійснюється за допомогою JavaScript та класам Bootstrap для валідації форм.

Висновки

При виконанні лабораторної роботи було створено прототип бази даних з XML файлів, клас що «спілкується» з цією базою даних. Була встановлена бібліотека JSTL, що певним чином пришвидшує розробку, але потребує ще якийсь час для освоєння. Було чітко виділено структуру проекту з відповідними теками для моделей, сервлетів, JSP сторінок. Ця структура може розростатися разом із ростом самого проекту. При створенні веб-застосування не використовувалися фреймворки для роботи з сервлетами (Spring і т.д), що, вірогідно, зробило б розроблене застосування більш безпечним, та значно б спростило розробку. Процес створення сервлетів, JSP сторінок нагадує створення Зображеннь, Контролерів у ASP Net Core, порівняно з яким у звичайних сервлетів (без використування фреймоворків) значно бракує функціоналу.