**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**Чорноморський національний університет імені Петра Могили**

**Факультет комп’ютерних наук**

**Кафедра інженерії програмного забезпечення**

**ЗВІТ**

**З ПРАКТИЧНОЇ РОБОТИ № 1**

з дисципліни «Безпека життєдіяльності (безпека життєдіяльності, основи охорони праці)»

121 – ПР.01 – 409.21910506

Виконав: студент 4 курсу, групи 405

напряму підготовки

123 «Комп’ютерна інженерія»

(шифр і назва напряму підготовки)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Д.В.Варанкін

(підпис, ініціали та прізвище)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (дата)

Перевірив: доцент кафедри, кандидат фізико-математичних наук.\_\_\_\_\_\_С.В. Пузирьов

(підпис, ініціали та прізвище) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата)

**м. Миколаїв – 2022**

У пакеті **entities** ми створимо Userклас, який має дві приватні рядкові змінні: **ім’я** та **пароль** . Створіть конструктори (за замовчуванням і той, який приймає обидва значення) і геттери/сеттери, а також замініть toString()метод на всякий випадок разом із методами equals()and . hashCode()Іншими словами, ми будемо робити все, що робить респектабельний Java-розробник, створюючи клас.

**public** **class** User {

**private** String name;

**private** String password;

**public** User() {

}

**public** User(String name, String password) {

**this**.name = name;

**this**.password = password;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getPassword() {

**return** password;

}

**public** **void** setPassword(String password) {

**this**.password = password;

}

@Override

**public** String toString() {

**return** "User{" +

"name='" + name + '\'' +

", password='" + password + '\'' +

'}';

}

@Override

**public** **boolean** equals(Object o) {

**if** (**this** == o) **return** true;

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

User user = (User) o;

**if** (name != **null** ? !name.equals(user.name) : user.name != **null**) **return** false;

**return** password != **null** ? password.equals(user.password) : user.password == **null**;

}

@Override

**public** **int** hashCode() {

**int** result = name != **null** ? name.hashCode() : 0;

result = 31 \* result + (password != **null** ? password.hashCode() : 0);

**return** result;

}

}

Тепер ми можемо почати створювати список користувачів. Ми додамо до нього користувачів і візьмемо з нього користувачів, щоб відобразити їх. Однак у нас є одна проблема. Ми не створюємо наші об’єкти сервлетів. ***Tomcat*** робить це за нас. Методи, які ми в них перевизначаємо, уже визначені для нас, і ми не можемо використовувати параметри. Як тоді створити спільний список, який буде видно в обох наших сервлетах? Якщо ми просто створимо об’єкт списку в кожному сервлеті, тоді ми додамо користувачів до одного списку, але відобразимо користувачів з іншого в **ListServlet**. Отже, нам потрібен об’єкт, спільний для обох сервлетів. Загалом, нам потрібен об’єкт, який є спільним для всіх класів нашої програми: один об’єкт для всієї програми. Сподіваюся, ви щось чули про шаблони проектування. Для деяких людей це може бути першою справжньою потребою в шаблоні **Singleton** у їхній програмі. Ви можете піти на розум і створити якийсь приємний **Singleton** із подвійною перевіркою та синхронізацією (так, наша програма багатопотокова, оскільки сервлети Tomcat працюють в окремих потоках). Але я збираюся використовувати техніку ранньої ініціалізації, оскільки вона цілком підходить для наших цілей.

**Створення моделі**

Створіть клас (і реалізуйте шаблон **Singleton ) у пакеті моделі** та назвіть його якось незвичайно. Наприклад, **Модель** . Ми створимо приватний список користувачів у нашому класі та реалізуємо два методи: один для додавання користувача, а інший для повернення списку рядків (імен користувачів). Оскільки наш об’єкт користувача складається з імені користувача та пароля, і ми не хочемо розкривати паролі користувачів, у нас буде лише список імен.

**public** **class** Model {

**private** **static** Model instance = **new** Model();

**private** List<User> model;

**public** **static** Model getInstance() {

**return** instance;

}

**private** Model() {

model = **new** ArrayList<>();

}

**public** **void** add(User user) {

model.add(user);

}

**public** List<String> list() {

**return** model.stream()

.map(User::getName)

.collect(Collectors.toList());

}

}

**Трохи про MVC**

Оскільки ви вже чули про **singleton** , ви, мабуть, чули про інший шаблон проектування **model-view-controller** (MVC). Його мета — відокремити бізнес-логіку від представлення. Тобто відокремити код, який визначає, що робити, від коду, який визначає, як відображати речі. Представлення відповідає за спосіб представлення даних **.** У нашому випадку представленнями є наші *JSP-сторінки* . Саме тому я помістив їх у папку під назвою **views** . **Модель** — це дані, з якими насправді працює програма. У нашому випадку це користувачі (список користувачів). А сполучною ланкою між ними є контролери. Вони беруть дані з моделі та передають їх у представлення (або отримують деякі дані з *Tomcat*, обробити його та передати моделі). Ви визначаєте свою бізнес-логіку (що має робити програма) у них, а не в моделі чи поданні. Таким чином, кожна частина займається своєю справою:

* модель зберігає дані;
* перегляди створюють гарне представлення даних;
* контролери обробляють дані.

Це дозволяє програмі бути досить простою та зручною для обслуговування, а не жахливою купою всього коду в одному класі. **MVC** підходить не тільки для веб-програмування, але і використовується в цій області особливо часто (майже завжди). У нашому випадку сервлети будуть виконувати роль контролерів. Це дуже поверховий і короткий опис шаблону, але **MVC** не є основною темою цієї статті. Якщо хтось хоче дізнатися більше, Google ваш друг! **Створіть форму для додавання користувача.** Додайте форму до **add.jsp** . Він повинен складатися з двох полів введення тексту (одне звичайне, інше — поле пароля) і кнопки для відправки даних на сервер.

<form method="post">

<label>Name:

<input type="text" name="name"><br />

</label>

<label>Password:

<input type="password" name="pass"><br />

</label>

<button type="submit">Submit</button>

</form>

Тут форма має атрибут **методу** зі значенням **post** . Це вказує на те, що дані з цієї форми надходитимуть на сервер як *запит POST* . Атрибут **дії** не вказано, що означає, що запит буде надіслано на ту ж адресу, з якої ми зайшли на цю сторінку ( **/add** ). Таким чином, отримавши *запит GET* , наш сервлет, прив’язаний до цієї адреси, повертає *JSP* із формою додавання користувача. І якщо він отримує *запит POST* , ми знаємо, що форма надіслала сюди свої дані (які ми витягуємо з об’єкта запиту вdoPost()метод, процес і перехід до моделі для збереження). Варто зазначити, що поля введення мають параметр, який називається **name** (для імен користувачів або **pass** для паролів). Це дуже важливий момент. **Таким чином, щоб отримати ці дані ( ім’я** користувача та **пароль** , які буде введено) із запиту (всередині сервлета), ми будемо використовувати ці поля імені та пароля. Але про це пізніше. Моя кнопка для надсилання даних знову була зроблена як **кнопка** , а не як поле виводу, як це прийнято. Я не знаю, наскільки поширений цей підхід, але він працює для мене (браузер Chrome).

**Обробка сервлетом запитів POST**

Повернемося до **AddServlet** . Нагадую вам, щоб дозволити нашому сервлету «ловити» *запити GET* , ми замінили doGet()метод у класі **HttpServlet** . Щоб навчити наш сервлет також перехоплювати *запити POST* , ми також повинні перевизначити doPost()метод. *Tomcat* передає йому подібні об’єкти запиту та відповіді, з якими ми будемо працювати. Для початку витягніть назву запиту та передайте параметри, надіслані формою (якщо ви вказали інші імена у формі, використовуйте ці імена). Після цього створіть об'єкт користувача, використовуючи отримані дані. Потім ми отримуємо об’єкт моделі та додаємо створеного користувача до моделі.

@Override

**protected** **void** doPost(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {

String name = req.getParameter("name");

String password = req.getParameter("pass");

User user = **new** User(name, password);

Model model = Model.getInstance();

model.add(user);

}

**Передача даних до представлення**

Давайте перейдемо до **ListServlet** . Метод doGet()вже реалізований. Він просто передає керування перегляду ( **list.jsp** ). Якщо у вас його ще немає, створіть його за аналогією з методом у **AddServlet** . Тепер було б чудово отримати список імен користувачів із моделі та передати їх у представлення, яке їх отримає та красиво відобразить. Для цього ми знову використаємо об’єкт запиту, отриманий від *Tomcat* . Ми можемо додати атрибут до цього об’єкта, давши йому якесь ім’я. Фактично, ми можемо додати об’єкт, який хочемо передати, до **представлення**. Завдяки тому факту, що під час передачі керування від сервлета до представлення ми передаємо представленню ті самі об’єкти запиту та відповіді, які отримав сервлет, ми можемо додати наш список імен до об’єкта запиту, а потім отримати наш список імен користувачів із запиту об'єкт у поданні. Ми закінчили з класом **ListServlet** , тому я покажу тут код усього класу:

**package** app.servlets;

**import** app.model.Model;

**import** javax.servlet.RequestDispatcher;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**import** java.io.IOException;

**import** java.util.List;

**public** **class** ListServlet **extends** HttpServlet {

@Override

**protected** **void** doGet(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {

Model model = Model.getInstance();

List<String> names = model.list();

req.setAttribute("userNames", names);

RequestDispatcher requestDispatcher = req.getRequestDispatcher("views/list.jsp");

requestDispatcher.forward(req, resp);

}

}

**Виконання коду Java у файлах JSP**

Настав час переглянути **list.jsp** . Він буде виконано лише тоді, коли **ListServlet** передасть йому керування. Крім того, ми вже підготували список імен користувачів із моделі в сервлеті та передали його сюди в об’єкті запиту. Оскільки у нас є список імен, ми можемо перебирати його за допомогою forциклу та відображати кожне ім’я. Як я вже говорив раніше, *файли JSP* можуть виконувати *код Java* (це те, що відрізняє їх від статичних HTML-сторінок). Щоб виконати якийсь код, все, що нам потрібно зробити, це розмістити наступну конструкцію у відповідному місці:

<!-- html code -->

<%

// Java code

%>

<!-- html code -->

**У цій конструкції ми отримуємо доступ до кількох змінних:**

* **request** — наш об’єкт запиту, який ми передали з сервлета, де він називався просто **req** ;
* **response** — об’єкт відповіді (у сервлеті називається **resp** );
* **out** — об’єкт **JspWriter** (який успадковує звичайний **Writer** ), за допомогою якого ми можемо «записати» щось безпосередньо на саму *HTML-сторінку* . Оператор **out.println("Hello, World!")** дуже схожий на **System.out.println("Hello, World!")** , але не плутайте їх!
* **out.println()** "записує" на *HTML-сторінку* , тоді як **System.out.println** записує *в системний вихідний потік* . Якщо ви викликаєте **System.out.println()** у розділі **JSP із***кодом Java* , ви побачите результати на консолі *Tomcat* , але не на сторінці.

Ви можете шукати інші об’єкти, доступні в JSP [, тут](https://www.google.com/search?q=JSP+Implicit+Objects) . Ми можемо використовувати об’єкт **запиту** , щоб отримати список імен, переданих із сервлета (ми прикріпили відповідний атрибут до цього об’єкта), і ми отримуємо використання об’єкта **out** для відображення цих імен. Давайте відобразимо їх (на даний момент у вигляді простого списку HTML):

<ul>

<%

List<String> names = (List<String>) request.getAttribute("userNames");

**if** (names != **null** && !names.isEmpty()) {

**for** (String s : names) {

out.println("<li>" + s + "</li>");

}

}

%>

</ul>

Якщо нам потрібно відобразити список лише за наявності користувачів, а в іншому випадку вивести попередження про те, що користувачів ще немає, тоді ми можемо трохи переписати цей розділ:

<%

List<String> names = (List<String>) request.getAttribute("userNames");

**if** (names != **null** && !names.isEmpty()) {

out.println("<ui>");

**for** (String s : names) {

out.println("<li>" + s + "</li>");

}

out.println("</ui>");

} **else** out.println("<p>There are no users yet!</p>");

%>

Тепер, коли ми знаємо, як передавати дані з сервлетів у представлення, ми можемо покращити наш **AddServlet** , щоб він відображав сповіщення про успішне додавання користувача. Для цього в doPost()методі після додавання нового користувача до моделі ми можемо додати це ім’я користувача до атрибутів об’єкта **req** і передати керування назад до **представлення** ( **add.jsp** ). А тепер додамо до нього розділ із кодом Java, де перевіримо, чи є у запиті такий атрибут, і якщо він є — то виведемо повідомлення про те, що користувача успішно додано. Після цих змін повний код **AddServlet** виглядатиме приблизно так:

**package** app.servlets;

**import** app.entities.User;

**import** app.model.Model;

**import** javax.servlet.RequestDispatcher;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**import** java.io.IOException;

**public** **class** AddServlet **extends** HttpServlet {

@Override

**protected** **void** doGet(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {

RequestDispatcher requestDispatcher = req.getRequestDispatcher("views/add.jsp");

requestDispatcher.forward(req, resp);

}

@Override

**protected** **void** doPost(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {

String name = req.getParameter("name");

String password = req.getParameter("pass");

User user = **new** User(name, password);

Model model = Model.getInstance();

model.add(user);

req.setAttribute("userName", name);

doGet(req, resp);

}

}

Тут в кінці doPost()методу ми створюємо атрибут з іменем користувача, який був доданий в модель, а потім викликаємо doGet()метод, якому передаємо поточний запит і відповідь. Тепер doGet()метод передає керування представленню, яке також отримує об’єкт запиту з іменем доданого користувача, доданим як атрибут. Нам залишається виправити **add.jsp** , щоб він відображав сповіщення, якщо такого атрибута немає. Ось остаточна версія **add.jsp** :

<%@ page contentType="text/html;charset=UTF-8" language="java" %>

<html>

<head>

<title>Add **new** user</title>

</head>

<body>

<div>

<h1>Super app!</h1>

</div>

<div>

<%

**if** (request.getAttribute("userName") != **null**) {

out.println("<p>User '" + request.getAttribute("userName") + "' added!</p>");

}

%>

<div>

<div>

<h2>Add user</h2>

</div>

<form method="post">

<label>Name:

<input type="text" name="name"><br />

</label>

<label>Password:

<input type="password" name="pass"><br />

</label>

<button type="submit">Submit</button>

</form>

</div>

</div>

<div>

<button onclick="location.href='/'">Back **to** main</button>

</div>

</body>

</html>

**Тіло сторінки складається з наступного:**

* div із заголовком;
* контейнер div для вмісту, який включає перевірку наявності атрибута з іменем користувача;
* div з формою додавання користувача;
* а внизу — нижній колонтитул із кнопкою повернення на головну сторінку.

Це може здатися забагато div, але ми використаємо їх пізніше, коли додамо стилі. Ось остаточна версія **list.jsp** :

<%@ page **import**="java.util.List" %>

<%@ page contentType="text/html;charset=UTF-8" language="java" %>

<html>

<head>

<title>Users</title>

</head>

<body>

<div>

<h1>Super app!</h1>

</div>

<div>

<div>

<div>

<h2>Users</h2>

</div>

<%

List<String> names = (List<String>) request.getAttribute("userNames");

**if** (names != **null** && !names.isEmpty()) {

out.println("<ui>");

**for** (String s : names) {

out.println("<li>" + s + "</li>");

}

out.println("</ui>");

} **else** out.println("<p>There are no users yet!</p>");

%>

</div>

</div>

<div>

<button onclick="location.href='/'">Back **to** main</button>

</div>

</body>

</html>

Таким чином, ми маємо повністю робочий веб-додаток, який може зберігати та додавати користувачів, а також виводити список їх імен. Тепер нам просто потрібно зробити це красиво... :)

**Додавання стилів. Ми будемо використовувати структуру W3.CSS**

На даний момент наш додаток працює, але виглядає абсолютно епатажно. Отже, давайте додамо фон, розфарбуємо текст і кнопки, додамо стилі до списків, вирівняємо елементи, додамо відступи тощо. Написання стилів вручну може зайняти багато часу та напружити наші нерви. Тому я пропоную використовувати структуру *W3.CSS* . У ньому вже є готові до використання класи зі стилями. Нам просто потрібно розташувати класи CSS, які ми хочемо використовувати, у правильних місцях. Щоб додати їх на наші сторінки, ми спочатку підключаємо файл стилів. Є два способи зробити це:

1. перегляньте наші сторінки та вставте наступне пряме посилання на файл стилів у розділі head

**<link rel="stylesheet" href="https://www.w3schools.com/w3css/4/w3.css">**

Цей варіант підходить, якщо у вас є постійне підключення до Інтернету. Коли ви відкриваєте свої сторінки на локальному сервері, стилі завантажуються з Інтернету.

1. Але якщо ви хочете мати всі стилі локально і не залежати від підключення до Інтернету, завантажте файл стилю та розмістіть його десь у **веб-** теці (наприклад **, web/styles/w3.css** ). Потім перегляньте всі наші сторінки ( **index.html, add.jsp, list.jsp** ) і додайте таке посилання до файлу стилів у розділі **head** :

**<link rel="stylesheet" href="styles/w3.css">**

Після цього просто перегляньте теги та додайте стилі, які вам подобаються. Я не буду зупинятися на цьому детально. Натомість я просто надам готові до використання версії трьох моїх файлів із класами растрового стилю.

**index.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Super app!</title>

<link rel="stylesheet" href="https://www.w3schools.com/w3css/4/w3.css">

</head>

<body **class**="w3-light-grey">

<div **class**="w3-container w3-blue-grey w3-opacity w3-right-align">

<h1>Super app!</h1>

</div>

<div **class**="w3-container w3-center">

<div **class**="w3-bar w3-padding-large w3-padding-24">

<button **class**="w3-btn w3-hover-light-blue w3-round-large" onclick="location.href='/list'">List users</button>

<button **class**="w3-btn w3-hover-green w3-round-large" onclick="location.href='/add'">Add user</button>

</div>

</div>

</body>

</html>

**add.jsp**

<%@ page contentType="text/html;charset=UTF-8" language="java" %>

<html>

<head>

<title>Add **new** user</title>

<link rel="stylesheet" href="https://www.w3schools.com/w3css/4/w3.css">

</head>

<body **class**="w3-light-grey">

<div **class**="w3-container w3-blue-grey w3-opacity w3-right-align">

<h1>Super app!</h1>

</div>

<div **class**="w3-container w3-padding">

<%

**if** (request.getAttribute("userName") != **null**) {

out.println("<div class=\"w3-panel w3-green w3-display-container w3-card-4 w3-round\">\n" +

" <span onclick=\"this.parentElement.style.display='none'\"\n" +

" class=\"w3-button w3-margin-right w3-display-right w3-round-large w3-hover-green w3-border w3-border-green w3-hover-border-grey\">×</span>\n" +

" <h5>User '" + request.getAttribute("userName") + "' added!</h5>\n" +

"</div>");

}

%>

<div **class**="w3-card-4">

<div **class**="w3-container w3-center w3-green">

<h2>Add user</h2>

</div>

<form method="post" **class**="w3-selection w3-light-grey w3-padding">

<label>Name:

<input type="text" name="name" **class**="w3-input w3-animate-input w3-border w3-round-large" style="width: 30%"><br />

</label>

<label>Password:

<input type="password" name="pass" **class**="w3-input w3-animate-input w3-border w3-round-large" style="width: 30%"><br />

</label>

<button type="submit" **class**="w3-btn w3-green w3-round-large w3-margin-bottom">Submit</button>

</form>

</div>

</div>

<div **class**="w3-container w3-grey w3-opacity w3-right-align w3-padding">

<button **class**="w3-btn w3-round-large" onclick="location.href='/'">Back **to** main</button>

</div>

</body>

</html>

**list.jsp**

<%@ page **import**="java.util.List" %>

<%@ page contentType="text/html;charset=UTF-8" language="java" %>

<html>

<head>

<title>Users list</title>

<link rel="stylesheet" href="https://www.w3schools.com/w3css/4/w3.css">

</head>

<body **class**="w3-light-grey">

<div **class**="w3-container w3-blue-grey w3-opacity w3-right-align">

<h1>Super app!</h1>

</div>

<div **class**="w3-container w3-center w3-margin-bottom w3-padding">

<div **class**="w3-card-4">

<div **class**="w3-container w3-light-blue">

<h2>Users</h2>

</div>

<%

List<String> names = (List<String>) request.getAttribute("userNames");

**if** (names != **null** && !names.isEmpty()) {

out.println("<ul class=\"w3-ul\">");

**for** (String s : names) {

out.println("<li class=\"w3-hover-sand\">" + s + "</li>");

}

out.println("</ul>");

} **else** out.println("<div class=\"w3-panel w3-red w3-display-container w3-card-4 w3-round\">\n"

+

" <span onclick=\"this.parentElement.style.display='none'\"\n" +

" class=\"w3-button w3-margin-right w3-display-right w3-round-large w3-hover-red w3-border w3-border-red w3-hover-border-grey\">×</span>\n" +

" <h5>There are no users yet!</h5>\n" +

"</div>");

%>

</div>

</div>

<div **class**="w3-container w3-grey w3-opacity w3-right-align w3-padding">

<button **class**="w3-btn w3-round-large" onclick="location.href='/'">Back **to** main</button>

</div>

</body>

</html>





