Київський національний університет імені Тараса Шевченка

Курсова робота на тему

**«Web-проект резервації готельних номерів»**

Виконав

студент 1 курсу магістратури, 2 групи

Цимбал Антон Миколайович

Науковий керівник:

Бородін Віктор Анатолійович

Київ 2017

**ЗМІСТ**

**Розділ 1.** Опис проекту

**Тема 1.1.** Запуск програми………………………………………………...3

**Тема 1.2.** Взаємодія з програмою…………………………………………4

**Тема 1.3.** Стек технологій…………………………………………............6

**Розділ 2.** Реалізація проекту

**Тема 2.1.** Apache Maven…………………………………………………...7

**Тема 2.2.** Apache Tomcat…………......……………………………………8

**Тема 2.3.** Java Servlets……………………………………………………...8

**Тема 2.4.** JSP (JavaServer Pages)…………..……………………………..11

**Тема 2.5.** Взаємодія Сервлета й JSP.…………………………………….17

**Тема 2.6.** Sessions……………………..…………………………………..17

**Тема 2.7.** Filters………………………..…………………………………..18

**Тема 2.8.** JDBC (Java Database Connectivity)..…………………………..21

**Тема 2.9.** Log4j………………………..…………………………………..25

**Тема 2.10.** Структура проекту………….………………………………..26

**Тема 2.11.** Структура бази даних...…….………………………………..28

**Список використаних джерел**………………………………………………...29

**Розділ 1. Опис проекту**

**Тема 1.1. Запуск програми**

§1.1.1. Підняття проекту

Оскільки проект використовує *Maven*, його запуск не викликає особливих труднощів. Потрібно зробити лише наступні дії:

1. Завантажити та налаштувати [Maven](https://maven.apache.org/download.cgi) та [Git](https://git-scm.com/downloads).
2. Завантажити проект із репозиторія:

**git clone https://github.com/fantotsy/HotelApp.git**

1. Перейти в папку з проектом і, знаходячись в ній, виконати наступну команду: **mvn tomcat7:run**

Дана команда запускає проект на порту 8086, тому цей порт має бути вільним. Якщо ж Ви бажаєте запустити проект на іншому порту, то необхідно спочатку внести зміни в конфігурацію файлу **HotelApp\pom.xml**, а саме:

<**plugin**>  
 <**groupId**>org.apache.tomcat.maven</**groupId**>  
 <**artifactId**>tomcat7-maven-plugin</**artifactId**>  
 <**version**>${tomcat.version}</**version**>  
 <**configuration**>  
 <**port**>8086</**port**>  
 </**configuration**>  
</**plugin**>

1. Результатом виконання попереднього пункту буде, зокрема, поява директорії **target** зі скомпільованим кодом проекту. Потрібно файл **HotelApp\src\main\webapp\META-INF\context.xml** скопіювати в директорію **HotelApp\target\tomcat\conf**.
2. Тепер необхідно заново запустити сервер (знову виконати пункт 3).

Тепер проект зконфігурований та піднятий на порту, який обрано в пункті 2. Отже, потрібно перейти за посиланням **http://localhost:\*\*\*\*/HotelApp**, де зірочками позначено вищезгаданий порт.

§1.1.2. Налаштування бази даних

Залишається налаштувати базу даних із початковими тестовими значеннями, щоб використовувати всі можливості даної програми.

Для початку потрібно завантажити та встановити [MySQL Server](https://dev.mysql.com/downloads/mysql/) та за бажанням для зручності маніпуляції з БД можна встановити [MySQL Workbench](https://dev.mysql.com/downloads/workbench/).

Потім необхідно створити базу даних (**CREATE DATABASE hoteldb;**). Слід зауважити, що можна створити БД і з іншим іменем, але потім доведеться внести свої конфігурації у спеціальний файл, який розглянемо трохи пізніше.

За шляхом **HotelApp\DB** містяться скрипти для створення таблиць бази даних (**DDL.sql**), а також для заповнення таблиць тестовими даними (**DML.sql**). Виконавши SQL-скрипти цих файлів, матимемо повністю зконфігуровану та готову до використання БД. Однак, потрібно ще перевірити дані доступу до створеної бази у файлі **HotelApp\target\tomcat\conf\context.xml**.

Тепер проект можна вважати повністю налаштованим та готовим до використання.

**Тема 1.2. Взаємодія з програмою**

Даний проект являє собою Web-програму для резервації готельних номерів. Програма дає можливість користуватися як користувачеві, так і адміністратору.

§1.2.1. Використання програми користувачем

Робота з програмою починається зі стартової сторінки авторизації, яка знаходиться за посиланням **/HotelApp**. Оскільки дана програма підтримує інтернаціоналізацію, то можна вибрати одну з двох доступних мов: українську чи англійську.

Перегляд доступних номерів та їх замовлення можливе лише за умови реєстрації користувача. Отже, перш за все необхідно зареєструватися, використовуючи сторінку реєстрації, яка знаходиться за посиланням **/HotelApp/registration**, або ж натиснувши кнопку **Sign up** на стартовій сторінці. Слід зазначити, що у формі реєстрації всі поля є обов’язковими для заповнення.

Після успішної реєстрації користувач потрапляє на сторінку привітання з реєстрацією. Тут він має зробити вибір зайти на сайт, або ж перейти на стартову сторінку та залишитися в незалогіненому стані. Якщо користувач обирає перейти до стартової сторінки, то він повертається до початкового положення. Інакше він потрапляє на сторінку вибору готельних номерів та перегляду своїх існуючих резервацій.

Номери готелю відрізняються двома характеристиками: *тип номеру* (**lux**, **simple**) та *кількість місць у номері* (від **1** до **4**-ох). Також потрібно обрати *початкову та кінцеву дату*. Після цього користувач може почати пошук вільних номерів за своїм запитом. Слід зазначити, що можна обрати декілька варіантів кожної з двох характеристик. Наприклад, можна переглянути **lux** номери, які мають **2** або **3** місця.

Нарешті, користувач потрапляє на сторінку з результатом свого пошуку, де вказана інформація за номерами, які його цікавлять. Тут користувач може забронювати конкретні номери, вибравши їх у вказаній категорії та натиснувши відповідну кнопку. Резервація проходить одразу, тому можна перейти на попередню сторінку та побачити, що поле з існуючими резерваціями поповнилося новими записами.

§1.2.2. Використання програми адміністратором

Щоб зайти на сайт від імені адміністратора, необхідно ввести *логін*/*пароль* **admin**/**admin**, вибравши відмітку **administrator**. Після цього адміністратор потрапляє на сторінку, яка містить 3 розділи: **дії з номерами**, **дії з існуючими резерваціями** та **дії з зареєстрованими користувачами**. Взагалі кажучи, кожен з даних пунктів інтуїтивно зрозумілий, слід лише зазначити основні права адміністратора. Адміністратор може виконувати наступні дії:

* Переглядати інформацію про номери.
* Додавати номери в конкретні категорії.
* Видаляти номери з конкретних категорій.
* Переглядати інформацію про існуючі резервації.
* Змінювати період резервації конкретного номера.
* Переглядати інформацію про зареєстрованих користувачів.

Загалом, на цьому опис дій адміністратора можна завершити, як і опис можливостей програми.

**Тема 1.3. Стек технологій**

Для реалізації даного проекту використані наступні технології та фреймворки:

* **Apache Maven** – фреймворк для автоматизації зборки.
* **Apache Tomcat 7** – контейнер Сервлетів.
* **Java Servlets** – *Java* інтерфейс, реалізація якого розширює функціональні можливості сервера.
* **JSP** – технологія для створення створювати як статичні, так і динамічні компоненти на *Web-сторінках*.
* **MySQL** – реляційна система управління базами даних (*СУБД*).
* **JDBC** – стандарт взаємодії *Java-програм* з різними *СУБД*.
* **Apache Log4j** – бібліотека логування *Java-програм*.

Кожен із зазначених компонентів буде розглянуто більш детально в наступному розділі.

**Розділ 2. Реалізація проекту**

**Тема 2.1. Apache Maven**

Apache Maven – це фреймворк для автоматизації зборки проектів на основі опису їхньої структури в файлах на мові POM (Project Object Model), який є підмножиною XML.

Maven забезпечує декларативну зборку проекту, що означає, що в файлах опису проекту міститься його специфікація, а не окремі команди виконання. Інформація зборки проекту міститься в XML-файлі з назвою **pom.xml**.

Мінімальна конфігурація включає версію конфігураційного файлу, ім’я проекту та його версію. За допомогою **pom.xml** конфігуруються залежності від інших проектів (Maven самостійно завантажує необхідні бібліотеки), індивідуальні фази процесу побудови проекту, список плагінів, які реалізують порядок зборки.

Наприклад, у даному проекті контейнер *Сервлетів* Apache Tomcat 7 завантажується та використовується якраз за допомогою Maven плагіна, що набагато спрощує запуск проекту. Налаштування цього плагіна наведено нижче:

<**plugin**>  
 <**groupId**>org.apache.tomcat.maven</**groupId**>  
 <**artifactId**>tomcat7-maven-plugin</**artifactId**>  
 <**version**>${tomcat.version}</**version**>  
 <**configuration**>  
 <**port**>8086</**port**>  
 </**configuration**>  
</**plugin**>

Слід зазначити, що версія береться з **properties** поточного **pom.xml** файлу:

<**properties**>  
 …  
 <**tomcat.version**>2.2</**tomcat.version**>  
</**properties**>

**Тема 2.2. Apache Tomcat**

*Apache Tomcat* – це контейнер *Сервлетів* з відкритим кодом, написаний на мові *Java*. Зокрема, він реалізує специфікації *Сервлетів* та *JSP*. Коротко кажучи, *Tomcat* дозволяє запускати Web-програми.

Налаштування *Apache Tomcat* частково описано в попередній темі. В подальших темах також буде розглянуто деякі його застосування.

**Тема 2.3. Java Servlets**

*Сервлети* – це компоненти *Java Enterprise Edition* (*Java EE*/*J2EE*), які виконуються на стороні сервера. Вони обробляють клієнтські запити та динамічно генерують відповіді на них.

Усі *Сервлети* реалізують загальний інтерфейс **javax.servlet.Servlet**. Для обробки *HTTP-запитів*, в якості базового класу можна скористатися абстрактним класом **javax.servlet.http.HttpServlet**.

§2.3.1. Життєвий цикл Сервлета

Життєвий цикл *Сервлета* починається з його завантаження в пам’ять *Контейнером Сервлетів* при старті *Контейнера*, або ж при першому зверненні до *Сервлета* (тобто у відповідь на перший запит). Як було зазначено раніше, у даному проекті використовується *Контейнер Сервлетів* *Apache Tomcat 7*.

При завантаженні *Сервлета* першим викликається метод **init()**. Цей метод дає можливість *Сервлета* ініціалізувати дані та підготуватися до обробки запитів.

Після цього *Сервлет* можна вважати запущеним, він знаходиться в очікуванні запитів від клієнтів. Запит обслуговується методом **service(HttpServletRequest request, HttpServletResponse response)** *Сервлета*, а всі параметри запита містяться в об’єкті **request** класу **javax.servlet.http.HttpServletRequest**, який передається в *Сервлет*. Ще одним параметром цього метода є об’єкт **response** класу **javax.servlet.http.HttpServletResponse**, в який завантажується інформація для передачі клієнту. Для кожного нового клієнта при зверненні до *Cервлета* *Контейнер Сервлетів* створює окремий незалежний потік (Thread), в якому і викликається метод **service**. Слід також зазначити, що метод **service** призначений для одночасної обробки декількох запитів, тому він має бути написаний із урахуванням питань багатопотоковості.

В задачу метода **service** входить аналіз отриманого через запит метода доступу до ресурсів (у даному проекті *HTTP-метода*) та виклик метода, ім’я якого збігається з назвою метода доступу до ресурсів з додаванням префікса **do**. (наприклад, якщо *HTTP-метод* – GET, то викличеться метод **doGet()**). Цей метод має бути перевизначеним та містити кастомну функціональну логіку.

Після завершення виконання *Сервлета* *Контейнер Сервлетів* викликає метод **destroy()**, в тілі котрого зазвичай розміщена логіка по звільненню зайнятих *Сервлетом* ресурсів.

У якості прикладу *Сервлета* можна навести *Сервлет* із даного проекту **ua.fantotsy.controllers.ServletController**. Слід також зазначити, що цей проект є односервлетним (моносервлетним), тобто один *Сервлет* приймає всі запити, від клієнта, а всередині вирішує яку логіку застосувати виходячи з запиту. Такий підхід є реалізацією поведінкового паттерна *Command*.

§2.3.2. Конфігурація Сервлета

Для того, щоб *Сервлет* був доступний для *Контейнера Сервлетів*, необхідно прописати ім’я та шлях до нього в так званому *Дескрипторі Розгортання*, що являє собою *XML-файл* **src\main\webapp\WEB-INF\web.xml**. Крім того, в *Дескрипторному Файлі* можна визначати параметри ініціалізації, так званий мапінг *Сервлетів* та *JSP*, стартові сторінки та сторінки з повідомленнями про помилки, а також багато іншого.

Витяг з *Дескриптора* **web.xml** даного проекту, де проводиться описане вище налаштування:

Налаштування стартового *URL* (стартової сторінки):

<**welcome-file-list**>  
 <**welcome-file**>index</**welcome-file**>  
</**welcome-file-list**>

Мапінг *Сервлета* **ServletController**:

<**servlet**>  
 <**servlet-name**>Controller</**servlet-name**>  
 <**servlet-class**>ua.fantotsy.controllers.ServletController</**servlet-class**>  
</**servlet**>

<**servlet-mapping**>  
 <**servlet-name**>Controller</**servlet-name**>  
 <**url-pattern**>/index</**url-pattern**>  
 <**url-pattern**>/registration</**url-pattern**>  
 <**url-pattern**>/main</**url-pattern**>  
 <**url-pattern**>/guest</**url-pattern**>  
 <**url-pattern**>/admin</**url-pattern**>  
 <**url-pattern**>/apartments</**url-pattern**>  
 <**url-pattern**>/reservations</**url-pattern**>  
 <**url-pattern**>/guests</**url-pattern**>  
 <**url-pattern**>/booking</**url-pattern**>  
 <**url-pattern**>/order\_valid</**url-pattern**>  
 <**url-pattern**>/error</**url-pattern**>  
 <**url-pattern**>/set\_locale</**url-pattern**>  
</**servlet-mapping**>

Налаштування сторінки з повідомленням про помилку:

<**error-page**>  
 <**error-code**>404</**error-code**>  
 <**location**>/error</**location**>  
</**error-page**>  
  
<**error-page**>  
 <**error-code**>500</**error-code**>  
 <**location**>/error</**location**>  
</**error-page**>  
  
<**error-page**>  
 <**exception-type**>java.lang.Throwable</**exception-type**>  
 <**location**>/error</**location**>  
</**error-page**>

**Тема 2.4. JSP (JavaServer Pages)**

*JSP* забезпечує розподілення динамічної та статичної частин сторінки, результатом чого являється можливість зміни дизайну сторінки, не змінюючи динамічну складову. Ця властивість використовується при розробці та підтримці сторінок, оскільки дизайнерам немає необхідності знати, як працювати з динамічними даними. Під термінами динамічного та статичного змісту зазвичай розуміється не частина *JSP*, а складова Web-програми:

* Динамічні ресурси – змінні в процесі роботи: *Сервлети*, *JSP*, а також *Java-код*.
* Статичні ресурси – незмінні в процесі роботи: *HTML*, *JavaScript*, зображення тощо.

Можна також зазначити, що такий підхід розподілення логіки та представлення є невід’ємною частиною партерна проектування *MVC* (*Model-View-Controller*).

§2.4.1. Життєвий цикл JSP

Процеси, які виконуються з файлом *JSP* при першому виклику:

1. Браузер робить запит до *JSP-сторінки*.
2. *JSP-engine* аналізує зміст файлу *JSP*.
3. *JSP-engine* створює тимчасовий *Сервлет* з кодом, заснованим на вмісті *JSP-файлу*, при цьому *Контейнер* транслює оператори *Java* в метод **\_jspService()**, визначений в інтерфейсі **javax.servlet.jsp.JspPage**. Якщо не виявлено помилок компіляції, то цей метод викликається для безпосередньої обробки запита. Отриманий *Сервлет* є відповідальним за виконання статичних елементів *JSP*, визначених під час розробки, а також за створення динамічних елементів.
4. Отриманий код компілюється в **\*.class**.
5. Викликається метод **init()** та **\_jspService()**, і *Сервлет* виконується.
6. Комбінація статичного *HTML* та графіки разом із динамічними елементами, визначеними в оригіналі *JSP-сторінки*, надсилаються браузеру через вихідний потік об’єкта відповіді **ServletResponse**.

Наступне звернення до *JSP-файлу* просто викликає метод **\_jspService()** *Сервлета*. Він, в свою чергу, використовується доти, доки сервер не буде зупинено та *Сервлет* не буде вивантажений вручну, або доки не буде змінено *JSP-файл*.

*JSP-код* вкладається в спеціальні теги, які вказують *Контейнеру*, щоб він використовував цей код для генерації *Сервлета* або його частини. Таким чином, маємо документ, який одночасно містить і статичну сторінку, і теги *Java*, які управляють цією сторінкою. Статичні частини *HTML-сторінки* надсилаються у вигляді рядків в метод **write()**. Динамічні частини вкладаються прямо в код *Сервлета*. З цього моменту сторінка поводить себе як звичайна *HTML-сторінка* з асоційованим *Сервлетом*.

Усі JSP-файли даного проекту розміщено в каталозі **src\main\webapp\WEB-INF\jsp**.

§2.4.2. JSTL (JSP Standard Tag Library)

*JSTL* – розширення специфікації *JSP*, яке додає бібліотеку *JSP-тегів* для загальних потреб, таких як, наприклад, розбір *XML-даних*, створення циклів та підтримка інтернаціоналізації. Саме за допомогою *JSTL* в даному проекті реалізована інтернаціоналізація.

Створення *JSP-сторінок* із застосуванням *JSTL* дозволяє спростити розробку та відмовитися від використання *Java-коду* в *JSP* (так звані *Скриптлети*, використання яких зазвичай є поганою практикою, адже змішується логіка та представлення).

У *JSTL* вводиться поняття *Expression Language* (*EL*), який використовується для спрощення доступу до даних, що зберігаються в різних областях видимості (**page-scope**, **request-scope**, **application-scope**) та для обчислення простих виразів.

Всередині *JSP-сторінки* *EL* викликається за допомогою конструкції **“${**ім’я**}”**. Дані програми, як правило, складаються з об’єктів, що відповідають специфікації *JavaBeans*, або представляють собою колекції, такі як **List**, **Map**, **Array** тощо. *EL* надає доступ до цих об’єктів за допомогою операторів “**.**” та “**[]**”.

Бібліотека тегів *JSTL* складається з чотирьох груп тегів:

* **core** (основні теги) – зокрема містить **if/then** вирази, **switch** конструкції, вивід інформації на сторінку, створення та видалення контекстних змінних, управління властивостями *JavaBeans* компонентів, перехоплення виключень, ітерування по колекціям, створення *URL* та імпортування їхнього змісту.
* **formatting** (теги форматування) – зокрема, містить встановлення локалізації, локалізацію тексту та структури повідомлень, форматування та аналіз чисел, відсотків, грошей, дат тощо.
* **sql** (теги для роботи з *SQL*) – опис джерела даних, виконання запитів, поновлення даних і транзакцій, обробка результату запиту.
* **xml** (теги для обробки *XML*) – доступ і перетворення *XML*.

§2.4.3. Користувацькі теги

Починаючи з версії *JSP 1.1* з’явилася можливість визначення власних користувацьких тегів. У даному проекті такий функціонал використовується для виведення помилок при заповненні форм, або введенні помилкових даних у деякі поля.

Для створення користувацьких тегів необхідно визначити клас-обробник тегу (в даному проекті це клас **ua.fantotsy.tags.ErrorTag**, який наслідує клас **javax.servlet.jsp.tagext.TagSupport**). Цей клас визначає поведінку нового тегу. Також потрібно створити *дескрипторний файл бібліотеки тегів* (тобто файл **.tld**), в якій описуються один або декілька тегів, які встановлюють співвідношення між іменами *XML-елементів* та реалізацією тегів. У даному проекті цей файл міститься за шляхом **src\main\webapp\WEB-INF\TLDs\errorTag.tld**:

*<?***xml version="1.0" encoding="ISO-8859-1"***?>*<**taglib xmlns="http://java.sun.com/xml/ns/javaee"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://java.sun.com/xml/ns/javaee   
 http://java.sun.com/xml/ns/javaee/web-jsptaglibrary\_2\_1.xsd"  
 version="2.1"**>  
  
 <**tlib-version**>1.0</**tlib-version**>  
 <**short-name**>errorTag</**short-name**>  
 <**uri**>/WEB-INF/TLDs/errorTag</**uri**>  
  
 <**tag**>  
 <**name**>error</**name**>  
 <**tag-class**>ua.fantotsy.tags.ErrorTag</**tag-class**>  
 <**body-content**>empty</**body-content**>  
 <**attribute**>  
 <**name**>errorType</**name**>  
 <**required**>true</**required**>  
 <**rtexprvalue**>true</**rtexprvalue**>  
 </**attribute**>  
 <**attribute**>  
 <**name**>locale</**name**>  
 <**required**>true</**required**>  
 <**rtexprvalue**>true</**rtexprvalue**>  
 </**attribute**>  
 </**tag**>  
</**taglib**>

Коли виявляється початковий елемент тегу, викликається перевизначений метод **doStartTag()**, який містить основну логіку роботи тегу. В даному проекті цей метод повертає константу **SKIP\_BODY**, яка дає вказівку системі ігнорувати будь-який вміст між початковим та кінцевим елементами тегу. Іншими словами, кастомний тег, який обробляється класом **ErrorTag** не має тіла. Однак, він має два атрибута, вказані в файлі **errorTag.tld**: **errorType** та **locale**, які є типом помилки та мовою, якою потрібно її відобразити відповідно.

§2.4.4. Неявні об’єкти, доступні на JSP-сторінці

*JSP-сторінка* завжди має доступ до багатьох функціональних можливостей *Сервлета*, який створюється *Web-контейнером* за замовчуванням. Далі наведено основні неявні об’єкти, якими може маніпулювати *JSP-сторінка*:

* **request** – запит клієнта. Область видимості в межах сторінки.
* **response** – відповідь клієнту. Область видимості в межах сторінки.
* **pageContext** – визначає контекст *JSP-сторінки*. Область видимості в межах сторінки.
* **session** – створюється *Контейнером* у відповідності до протоколу *HTTP* та є екземпляром класу **javax.servlet.http.HttpSession**, надає інформацію про *Cесію* клієнта, якщо вона була створена. Область видимості в межах *Cесії*.

Детальніше про поняття *Cесії* буде йтися в Темі 6.

* **application** – контейнер, в якому виконується *JSP-сторінка*, є екземпляром класу **javax.servlet.ServletContext**. Область видимості в межах програми.
* **out** – містить вихідний потік *Сервлета*. Інформація, яка передається в цей потік, надходить до клієнта. Об’єкт є екземпляром класу **javax.servlet.jsp.JspWriter**. Область видимості в межах сторінки.
* **config** – містить параметри конфігурації *Сервлета* і є екземпляром класу **javax.servlet.ServletConfig**. Область видимості в межах сторінки.
* **page** – посилання **this** для поточного екземпляра даної сторінки, є об’єктом **java.lang.Object**. Область видимості в межах сторінки.
* **exception** – являє собою виключення одного з підкласів класу **java.lang.Throwable**, яке передає сторінці повідомлення про помилки. Область видимості в межах сторінки.

**Тема 2.5. Взаємодія Сервлета й JSP**

Взагалі кажучи, *Сервлет* та *JSP* виконують принципово різні ролі в програмі. *JSP-сторінка* відповідальна за формування інтерфейсу, призначеного для користувача, та відображає інформацію, яка передана з сервера. *Сервлет* виконує роль контролера запитів та відповідей, тобто приймає запити від усіх поєднаних з ним *JSP-сторінок* (або *URL-паттернів*), викликає відповідну бізнес-логіку для обробки запитів і в залежності від результату виконання вирішує яку *JSP-сторінку* поставити цьому результату у відповідність.

**Тема 2.6. Sessions**

При відвідуванні клієнтом *Web-ресурсу* та виконанні запитів, контекстна інформація про клієнта не зберігається. Протокол *HTTP* не має можливостей для збереження та зміни інформації про попередні відвідування клієнта. Таку задачу може виконувати *Cесія*.

*Сесія* (*Сеанс*) – з’єднання між клієнтом та сервером, яке встановлюється на деякий час, за який клієнт може відправити на сервер скільки завгодно запитів. *Сесія* встановлюється безпосередньо між клієнтом та *Web-сервером*, причому для кожного клієнта існує своя *Сесія*.

Щоб відкрити нову *Сесію*, використовується метод **getSession()** інтерфейса **HttpServletRequest**. Метод виймає з переданого в *Сервлет* запиту об’єкт **HttpSession**, який відповідає даному користувачеві. *Сесія*, зокрема, містить інформацію про дату та час створення останнього звернення до неї.

Якщо для метода **getSession(boolean param)** вхідним параметром є значення **true**, то *Сервлет-Контейнер* перевіряє наявність активного *Сеанса*, встановленого з даним клієнтом. У випадку успіху метод повертає дескриптор цього *Сеанса* (існуючу *Сесію*). Інакше, метод встановлює новий *Сеанс*, після чого починається збір інформації про клієнта.

Щоб зберігати значення змінної в поточній *Сесії*, використовується метод **setAttribute()** класу **HttpSession**. Для зчитування значення використовується метод **getAttribute()**.

Також слід зазначити, що у *Дескрипторі Розгортання* **web.xml** можна зазначається час інвалідації *Сесії* у хвилинах, тобто час її існування під час бездіяльності користувача:

<**session-config**>  
 <**session-timeout**>20</**session-timeout**>  
</**session-config**>

У даному проекті *Сесії* використовуються, зокрема, для визначення ролі залогіненого користувача (користувач чи адміністратор).

**Тема 2.7. Filters**

Реалізація інтерфейсу **javax.servlet.Filter** дозволяє створити об’єкт, який може трансформувати заголовок та зміст запиту клієнта чи відповіді сервера. *Фільтри* не створюють запит або відповідь, а лише модифікують їх. *Фільтр* виконує попередню обробку запиту, перш ніж він потрапить в *Cервлет*, з подальшою обробкою відповіді, яка надходить з *Cервлета* (за потреби).

*Фільтр* визначається реалізацією вищезазначеного інтерфейсу **Filter**. Основним методом цього інтерфейсу є метод **doFilter(ServletRequest request, ServletResponse response, FilterChain chain)**, якому передаються об’єкти запиту, відповіді та ланцюга *Фільтрів*. Він викликається щоразу, коли запит або відповідь проходить через список *Фільтрів* **FilterChain**.

§2.7.1. Життєвий цикл Фільтра

Життєвий цикл фільтра починається з одноразового виклику методу **init(FilterConfig config)**, який приймає параметр ініціалізації та налаштовує конфігураційний об’єкт *Фільтра* **FilterConfig**. Потім викликається метод **doFilter()**, описаний вище. Метод **destroy()** викликається при відключенні *Фільтра*. В ньому вивільняються використовувані ресурси.

§2.7.2. Налаштування Фільтрів

Щоб до *Фільтра* надходило звернення, необхідно ввімкнути спеціальну конфігурацію в *Дескрипторі Розгортання* **web.xml**. Наведемо приклад з даного проекту:

<**filter**>  
 <**filter-name**>AuthFilter</**filter-name**>  
 <**filter-class**>ua.fantotsy.filters.AuthorizationFilter</**filter-class**>  
</**filter**>  
  
<**filter-mapping**>  
 <**filter-name**>AuthFilter</**filter-name**>  
 <**url-pattern**>/\*</**url-pattern**>  
</**filter-mapping**>  
  
<**filter**>  
 <**filter-name**>CsrfFilter</**filter-name**>  
 <**filter-class**>ua.fantotsy.filters.CsrfFilter</**filter-class**>  
</**filter**>  
  
<**filter-mapping**>  
 <**filter-name**>CsrfFilter</**filter-name**>  
 <**url-pattern**>/order\_valid</**url-pattern**>  
 <**url-pattern**>/guest</**url-pattern**>  
 <**url-pattern**>/apartments</**url-pattern**>  
</**filter-mapping**>

<**filter**>  
 <**filter-name**>EncodingFilter</**filter-name**>  
 <**filter-class**>ua.fantotsy.filters.EncodingFilter</**filter-class**>  
 <**init-param**>  
 <**param-name**>defaultEncoding</**param-name**>  
 <**param-value**>UTF-8</**param-value**>  
 </**init-param**>  
</**filter**>  
  
<**filter-mapping**>  
 <**filter-name**>EncodingFilter</**filter-name**>  
 <**url-pattern**>/\*</**url-pattern**>  
</**filter-mapping**>

Бачимо, що в даному проекті налаштовано 3 *Фільтра*, які будуть виконуватися в порядку їх появи у файлі **web.xml**.

**ua.fantotsy.filters.AuthorizationFilter** використовується для попередження нелегального доступу. Наприклад, не дозволяє незалогіненому користувачеві перейти до сторінки перегляду поточних резервацій, просто зазначивши *URL* даної сторінки в адресному рядку.

**ua.fantotsy.filters.CsrfFilter** використовується для попередження *CSRF-атак*, використовуючи так званий токен, який є зашифрованим значенням *ID Сесії*.

**ua.fantotsy.filters.EncodingFilter** використовується для встановлення визначеного кодування перед тим як запит потрапить до *Сервлета*.

**Тема 2.8. JDBC (Java Database Connectivity)**

*JDBC* – це стандартний прикладний інтерфейс (*API*) мови *Java* для організації взаємодії між програмою та *СУБД*. Ця взаємодія здійснюється за допомогою драйверів *JDBC*, які забезпечують реалізацію загальних інтерфейсів для конкретних *СУБД* та конкретних протоколів.

За допомогою *JDBC* відправляються *SQL-запити* лише до реляційних баз даних, для яких існують драйвери, що знають спосіб спілкування з реальним сервером бази даних.

§2.8.1. Послідовність дій та процесів при роботі з JDBC

1. Завантаження драйвера бази даних, якщо екземпляр цього класу відсутній. Після цього виконується завантаження драйвера в пам’ять, що дає можливість з’єднання з *СУБД*.
2. Встановлення з’єднання з *СУБД*. Для цього викликається статичний метод **getConnection()** класу **java.sql.DriverManager**. В якості параметрів методу передаються *URL* бази даних, *логін* та *пароль* користувача бази даних. В свою чергу, метод повертає об’єкт **java.sql.Connection**.
3. Створення об’єкта для передачі запитів. Для виконання запитів застосовується об’єкт **java.sql.Statement**. Він використовується для виконання *SQL-запита* без його попередньої підготовки. Можуть також застосовуватися об’єкти **java.sql.PreparedStatement** та **java.sql.CallableStatement** для виконання підготовлених запитів та збережених процедур.

У даному проекті в основному використовується **PreparedStatement**, а підготовлені *SQL-запити* знаходяться у файлі **src\main\resources\properties\SQL\SQLs.properties**.

1. Виконання запиту. Результат виконання запиту поміщається в об’єкт **java.sql.ResultSet**.
2. Обробка результатів виконання запиту проводиться методами інтерфейса **ResultSet**, де найрозповсюдженішими є **next()** та **getString(int pos)**, а також аналогічні методи для отримання об’єктів, відмінних від об’єктів класу **java.lang.String**. При першому виклику метода **next()** вказівник переміщується на таблицю результатів вибірки в позицію першого рядка таблиці відповіді. Коли рядки закінчуються, метод повертає значення **false**.
3. Закриття з’єднання після того, як база вже не потрібна.

§2.8.2. Пул з’єднань

При великій кількості клієнтів, які працюють з програмою, до її бази даних виконується велика кількість запитів. Операція встановлення з’єднання з БД є досить витратною. Ефективним способом вирішення цієї проблеми є організація пулу використовуваних з’єднань, які не закриваються фізично, а зберігаються в черзі та надаються повторно для інших запитів.

У даному проекті використовується саме такий підхід. Знаходження та запуск пулу з’єднань забезпечується за допомогою *JNDI*. Розподілений доступ до джерела даних організовується шляхом статичної змінної типу **java.sql.DataSource**, що надає з’єднання з *СУБД*.

Клас **javax.naming.InitialContext**, як частина *JNDI API*, забезпечує роботу з каталогом іменованих об’єктів. У цьому каталозі можна зв’язати об’єкт джерела даних **DataSource** з деяким іменем, попередньо створивши об’єкт **DataSource**.

Потім створений об’єкт можна отримати за допомогою метода **lookup()** за його іменем.

Цей функціонал міститься в класі **ua.fantotsy.datasource.ConnectionPool**, який реалізований на основі паттерна *Singleton*:

**public final class** ConnectionPool {  
 **private static** ConnectionPool *instance*;  
 **private** DataSource **dataSource**;  
  
 **private** ConnectionPool() {  
 **try** {  
 Context context = **new** InitialContext();  
 Context envContext = (Context) context.lookup(**"java:/comp/env"**);  
 **dataSource** = (DataSource) envContext.lookup(**"/jdbc/hoteldb"**);  
 } **catch** (NamingException e) {  
 Logger logger = Logger.*getLogger*(ConnectionPool.**class**.getName());  
 logger.error(e);  
 }  
 }  
  
 **public static synchronized** ConnectionPool getInstance() {  
 **if**(*instance* == **null**){  
 *instance* = **new** ConnectionPool();  
 }  
 **return** *instance*;  
 }  
  
 **public** Connection getConnection() {  
 Connection connection = **null**;  
 **try** {  
 connection = **dataSource**.getConnection();  
 } **catch** (SQLException e) {  
 Logger logger = Logger.*getLogger*(ConnectionPool.**class**.getName());  
 logger.error(e);  
 }  
 **return** connection;  
 }  
}

Після виконання запиту з’єднання завершується, а його об’єкт повертається в пул викликом методу **close()**.

У даному проекті використовується *СУБД* *MySQL*, яка, як і багато інших *СУБД*, є сумісною з *JDBC*.

Для з’єднання з базою в даному проекті використовується дескриптор розгортання програми **src\main\webapp\META-INF\context.xml**, який має бути розміщений в директорії **target\tomcat\conf** після підняття *Apache Tomcat 7*. Основне призначення файлу – визначити джерело даних та помістити його в *JNDI* під певним іменем.

Файл **context.xml** даного проекту:

*<?***xml version="1.0" encoding="UTF-8"***?>*<**Context antiResourceLocking="true" path="/Hotel" docBase="myApp"  
 crossContext="true" reloadable="true"**>  
 <**Resource name="jdbc/hoteldb"  
 driverClassName="com.mysql.jdbc.Driver"  
 maxTotal="32"  
 maxIdle="8"  
 maxWaitMillis="10000"  
 username="root"  
 password="root"  
 type="javax.sql.DataSource"  
 url="jdbc:mysql://localhost:3306/hoteldb?useSSL=false&amp;  
 useUnicode=true&amp;  
 useJDBCCompliantTimezoneShift=true&amp;  
 useLegacyDatetimeCode=false&amp;  
 serverTimezone=UTC"  
 validationQuery="SELECT 1"  
 auth="Container"**/>  
</**Context**>

Бачимо, що *JDBC* драйвер вказано саме для *MySQL* *СУБД* (**com.mysql.jdbc.Driver**). Також слід звернути увагу на такі атрибути як **username**, **password**, **maxIdle** (максимальна кількість з’єднань у пулі, які знаходяться в очікуванні), **maxTotal** (максимальна кількість активних та неактивних з’єднань загалом).

Також, пул з’єднань треба додати в ресурси в *Дескрипторі Розгортання* **web.xml**. Це робиться наступним чином:

<**resource-ref**>  
 <**description**>MySQL DB Connection Pool</**description**>  
 <**res-ref-name**>jdbc/hoteldb</**res-ref-name**>  
 <**res-type**>javax.sql.DataSource</**res-type**>  
 <**res-auth**>Container</**res-auth**>  
</**resource-ref**>

Слід зауважити, що значення всередині тегу **<res-ref-name></res-ref-name>** має співпадати зі значенням **name**, вказаним в файлі **context.xml**. Також відповідність має бути між параметрами **<res-type></res-type>**, **<res-auth></res-auth>** та **type**, **auth** відповідно.

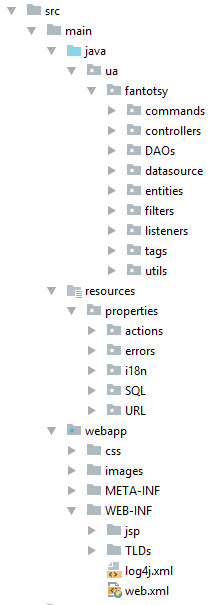
**Тема 2.9. Log4j**

*Log4j* – бібліотека логування *Java* програм. Дозволяє автоматизувати операції по збереженню інформації під час роботи програми, такі як трасування виконання конструкторів, методів, блоків обробки критичних ситуацій тощо.

У даному проекті *Log4j* використовується для логування виключень, які виникають в процесі роботи програми.

**Тема 2.10. Структура проекту**

Даний проект має наступну структуру пакетів, яка узгоджена зі специфікацією *Maven*:



*Java-код* міститься в пакеті **src.main.java.ua.fantotsy**. Розглянемо за яким принципом проводиться подальший розподіл по пакетам:

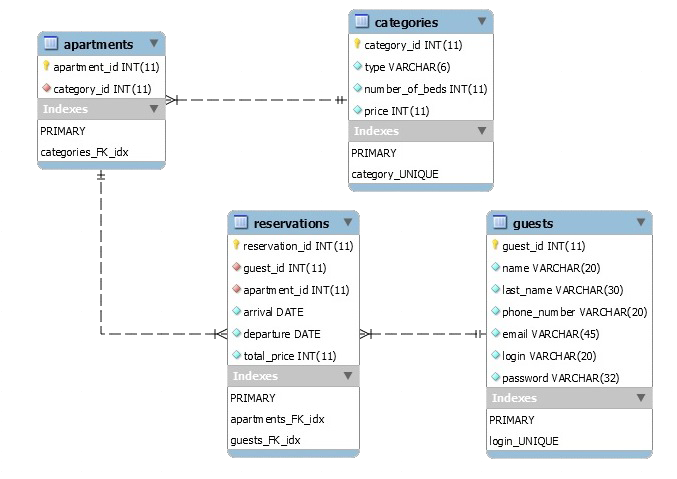
* **commands** містить класи, які реалізують інтерфейс **ICommand**. Ці команди реалізують згаданий раніше паттерн *Command* і названі виходячи з того, який функціонал вони містять.
* **controllers** зокрема,містить *Сервлет*, який обробляє всі *HTTP-запити* (адже проект побудований за односервлетною архітектурою), клас **CommandGetter**, який вирішує яку саме команду використати, виходячи з поточного запиту, а також **SessionRequestWrapper**, що являє собою обгортку над об’єктом **HttpServletRequest**.
* **DAOs** містить інтерфейси доступу до таблиць бази даних, а також їх імплементації.
* **datasource** містить класи, призначені для роботи пулу з’єднань до бази даних.
* **entities** містить класи, які являють собою структури об’єктів, які зберігаються в базі даних.
* **filters** містить *Фільтри*, що були розглянуті в Темі 2.7.
* **listeners** містить клас-слухач для роботи логера Log4j.
* **tags** містить клас для користувацького тегу, розглянутий в §2.4.3.
* **utils** містить класи-утиліти, які, зокрема, використовуються для отримання значень з *.properties-файлів*.

У пакеті **resources.properties** знаходяться *.properties-файли*, які використовуються для інтернаціоналізації, *SQL-запитів*, мапінга *URL* на команди, повідомлення помилок та шляхи до ресурсів, використовуваних у проекті.

Пакет **webapp** містить, зокрема, *JSP-сторінки*, *CSS-файли*, зображення та *Дескриптор Розгортання* проекту **web.xml**.

**Тема 2.11. Структура бази даних**

База даних даного проекту має наступну схему:



Отже, база даних містить 4 таблиці, пов’язані між собою відповідними зв’язками.

**Список використаних джерел**

1. [Java Platform Enterprise Edition API Specification](http://docs.oracle.com/javaee/6/api/?javax/servlet/package-summary.html)\_20.05.2017;
2. [Apache Tomcat 7 Documentation](http://tomcat.apache.org/tomcat-7.0-doc/)\_20.05.2017;
3. [MySQL Documentation](https://dev.mysql.com/doc/)\_20.05.2017;
4. [Apache Maven Documentation](http://maven.apache.org/guides/)\_20.05.2017;
5. [Wikipedia­](https://en.wikipedia.org/wiki/Main_Page)\_20.05.2017.