Національний технічний університет України  
«Київський політехнічний інститут імені Ігоря Сікорського»  
Факультет інформатики та обчислювальної техніки Кафедра обчислювальної техніки

**Лабораторна робота № 9**з дисципліни «Інженерія програмного забеспечення» на тему:

**Проєктування та реалізація програмного продукту з використанням архітектурної моделі “Кліент-Сервер”**

Виконав:  
Душко Роман  
Група ІО-32  
Залікова книжка № 3208

Перевірила:

Васильєва М. Д.

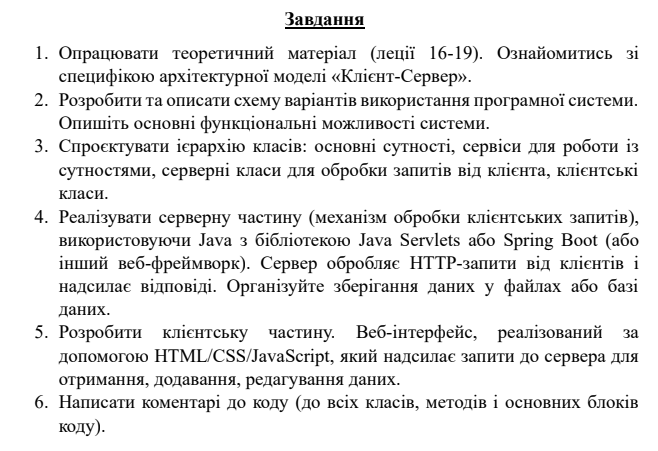
Київ – 2024

**Лабораторна робота № 9**

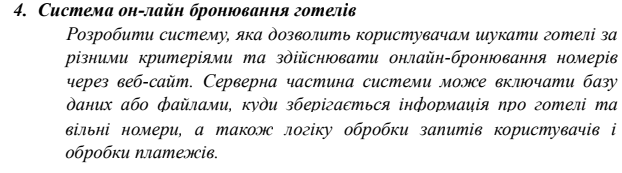
**Проєктування та реалізація програмного продукту з використанням**

**архітектурної моделі «Клієнт-Сервер»**

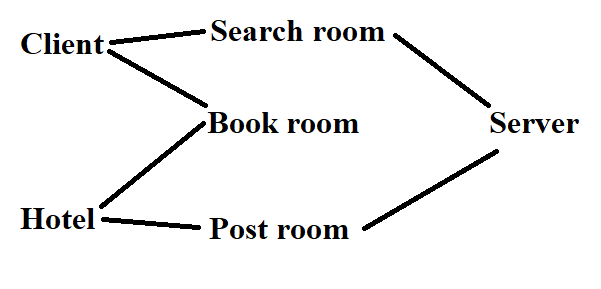
**Мета**: Вивчити реалізацію та специфіку використання архітектури «клієнт-сервер». Спроєктувати програмну систему (або її частину), архітектура якої відповідала б програмній моделі «Клієнт-Сервер».



Номер варіанту — 3208 % 6 = 4



**Схема варіантів використання:**



**Роздруківка коду:**

Main.java

package work9;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

/\*\*

\* The class that contains the entry point of the program

\*/

public class Main {

/\*\*

\* The entry point of the program that initializes the Spring Boot application

\* @param args THe arguments of the program

\*/

public static void main(String[] args) {

SpringApplication.run(Main.class, args);

}

}

HotelRoom.java

package work9;

/\*\*

\* A class that describes a hotel room

\*/

public class HotelRoom {

/\*\*

\* The internal id of the room

\*/

public int id;

/\*\*

\* Id setter

\*/

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

/\*\*

\* Id getter

\*/

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

/\*\*

\* The name of the hotel

\*/

public String hotelName;

/\*\*

\* hotelname setter

\*/

public void setHotelName(String hotelName) { this.hotelName = hotelName; }

/\*\*

\* hotelName getter

\*/

public String getHotelName() { return hotelName; }

/\*\*

\* The city where the room is located at

\*/

public String city;

/\*\*

\* city setter

\*/

public void setCity(String city) { this.city = city; }

/\*\*

\* city getter

\*/

public String getCity() { return city; }

/\*\*

\* The amout of stars the room has

\*/

public int stars;

/\*\*

\* stars setter

\*/

public void setStars(int stars) { this.stars = stars; }

/\*\*

\* stars getter

\*/

public int getStars() { return stars; }

/\*\*

\* The price for the room

\*/

public int price;

/\*\*

\* price setter

\*/

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

/\*\*

\* price getter

\*/

public int getPrice() { return price; }

/\*\*

\* Whether the room has already been booked

\*/

public boolean isBooked;

/\*\*

\* isBooked setter

\*/

public void setIsBooked(boolean isBooked) { this.isBooked = isBooked; }

/\*\*

\* isBooked getter

\*/

public boolean getIsBooked() { return this.isBooked; }

/\*\*

\* The number of the room in the hotel

\*/

public int roomNumber;

/\*\*

\* roomNumber setter

\*/

public void setRoomNumber(int roomNumber) { this.roomNumber = roomNumber; }

/\*\*

\* roomNumber getter

\*/

public int getRoomNumber() { return roomNumber; }

}

Filter.java

package work9;

/\*\*

\* A class that describes a filter that checks whether a hotel room passes the requirements

\*/

public abstract class Filter {

/\*\*

\* The next filter in the chain

\*/

private Filter next;

/\*\*

\* A method that combines two filters

\* @param f The next filter

\* @return The filter you passed to the method

\*/

public Filter then(Filter f) { next = f; return next; }

/\*\*

\* A method to check whether the hotelroom passes the whole chain of the filters

\* @param room The room to be checked

\* @return True if the room passes

\*/

public boolean filter(HotelRoom room) {

return check(room) && (next == null || next.filter(room));

}

/\*\*

\* The method that checks whether the room passes this specific filter

\* @param room The room to be checked

\* @return True if the room passes

\*/

public abstract boolean check(HotelRoom room);

}

FilterAlways.java

package work9;

/\*\*

\* A filter that always passes

\*/

public class FilterAlways extends Filter {

/\*\*

\* A filter that always passes the room

\* @param room The room in question

\* @return Always true

\*/

public boolean check(HotelRoom room) { return true; }

}

FilterCity.java

package work9;

/\*\*

\* A filter that filters the rooms based on their city

\*/

public class FilterCity extends Filter {

/\*\*

\* The city to be checked against

\*/

private String city;

/\*\*

\* The constructor for the city filter

\* @param city The city of the filter

\*/

public FilterCity(String city) {

this.city = city;

}

/\*\*

\* The method that checks whether the room's city fits

\* @param room The room to be checked

\* @return True if the city of the room is the same as filter's

\*/

public boolean check(HotelRoom room) {

return room.city.equals(city);

}

}

FilterId.java

package work9;

/\*\*

\* A filter that checks whether the ids are the same

\*/

public class FilterId extends Filter {

/\*\*

\* The id to be compared to

\*/

private int id;

/\*\*

\* A constructor for the id filter

\* @param id The id

\*/

public FilterId(int id) {

this.id = id;

}

/\*\*

\* The method that checks whether the ids are the same

\* @param room The room to be checked

\* @return True if the room id is the same

\*/

public boolean check(HotelRoom room) {

return room.id == id;

}

}

FilterIsBooked.java

package work9;

/\*\*

\* The filter that checks the status of room being booked

\*/

public class FilterIsBooked extends Filter {

/\*\*

\* The required status

\*/

private boolean isBooked;

/\*\*

\* A constructor for the filter

\* @param isBooked The asked status

\*/

public FilterIsBooked(boolean isBooked) {

this.isBooked = isBooked;

}

/\*\*

\* The method that checks whether the room passes the test

\* @param room The room to be checked

\* @return True if room's booked status is the same

\*/

public boolean check(HotelRoom room) {

return room.isBooked == this.isBooked;

}

}

FilterPrice.java

package work9;

/\*\*

\* The filter that checks whether the price fits in range

\*/

public class FilterPrice extends Filter {

/\*\*

\* The minimum allowed price

\*/

private int minPrice;

/\*\*

\* The maximum allowed price

\*/

private int maxPrice;

/\*\*

\* The constructor for the filter

\* @param minPrice The min price

\* @param maxPrice The max price

\*/

public FilterPrice(int minPrice, int maxPrice) {

this.minPrice = minPrice;

this.maxPrice = maxPrice;

}

/\*\*

\* The method that checks whether the room passes the price requirements

\* @param room The room to be checked

\* @return True if room's price fits the specified range

\*/

public boolean check(HotelRoom room) {

return room.price >= minPrice && room.price <= maxPrice;

}

}

FilterStars.java

package work9;

/\*\*

\* The filter that checks whether the room fits the range of stars

\*/

public class FilterStars extends Filter {

/\*\*

\* The minimum allowed amount of stars

\*/

private int minStars;

/\*\*

\* The maximum allowed amount of stars

\*/

private int maxStars;

/\*\*

\* The cosntructor for the filter

\* @param minStars The min stars

\* @param maxStars The max stars

\*/

public FilterStars(int minStars, int maxStars) {

this.minStars = minStars;

this.maxStars = maxStars;

}

/\*\*

\* The method that checks whether the room fits the stars requirement

\* @param room The room

\* @return True if room's star amount fits the specified range

\*/

public boolean check(HotelRoom room) {

return room.stars >= minStars && room.stars <= maxStars;

}

}

Iterator.java

package work9;

import java.util.List;

import java.util.ArrayList;

/\*\*

\* The class that describes the way to iterate over the hotel rooms

\*/

public abstract class Iterator {

/\*\*

\* The internal list

\*/

protected List<HotelRoom> list;

/\*\*

\* The method to get the next room

\* @return The next room

\*/

public abstract HotelRoom next();

/\*\*

\* The method to check whether there is a next room

\* @return True if there is next

\*/

public abstract boolean hasNext();

/\*\*

\* The method that gets n or less next items

\* @param n The max amount to get

\* @return A list of at most n rooms

\*/

public List<HotelRoom> nextn(int n) {

List<HotelRoom> result = new ArrayList<>();

while(hasNext() && (n--) > 0) { result.add(next()); }

return result;

}

/\*\*

\* The method to skip next n items

\* @param n The amount of items to skip

\*/

public void skipn(int n) {

nextn(n);

}

}

IteratorPrice.java

package work9;

import java.util.List;

import java.util.Collections;

/\*\*

\* The class that describes an iterator that goes from lowest price to highest

\*/

public class IteratorPrice extends Iterator {

/\*\*

\* The current index of the iterator

\*/

private int index;

/\*\*

\* The constructor that orders rooms to increase price

\* @param list The list of data

\*/

public IteratorPrice(List<HotelRoom> list) {

Collections.sort(list, (a, b) -> Integer.compare(a.price, b.price));

this.list = list;

}

/\*\*

\* Checks whether there is a next room

\* @return True if there is

\*/

public boolean hasNext() { return index < list.size(); }

/\*\*

\* Returns the next room is there is one

\* @return The next item

\*/

public HotelRoom next() { return list.get(index++); }

}

IteratorStars.java

package work9;

import java.util.List;

import java.util.Collections;

/\*\*

\* The iterator that orders the rooms from highest stars to lowest

\*/

public class IteratorStars extends Iterator {

/\*\*

\* The internal index

\*/

private int index;

/\*\*

\* The constructor for the iterator that orders rooms as stars decrease

\* @param list The list of data

\*/

public IteratorStars(List<HotelRoom> list) {

Collections.sort(list, (a, b) -> Integer.compare(b.stars, a.stars)); // descending

this.list = list;

}

/\*\*

\* The method that checks if there is a next item

\* @return True if yes

\*/

public boolean hasNext() { return index < list.size(); }

/\*\*

\* The method that returns the next item

\* @return The next item

\*/

public HotelRoom next() { return list.get(index++); }

}

VeryCoolController.java

package work9;

import java.util.List;

import java.util.ArrayList;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.ModelAttribute;

import java.util.Random;

@Controller

/\*\*

\* The class that routes the requests

\*/

public class VeryCoolController {

/\*\*

\* The amount of rooms that you can see per page

\*/

private static final int pageSize = 10;

@GetMapping("/search")

/\*\*

\* A method that routes the request to the search page, does the search idk i dont care

\* @param minStars Min stars

\* @param maxStars Max stars

\* @param minPrice Min price

\* @param maxPrice Max price

\* @param city City

\* @param id Id (if specified ignore everything else)

\* @param page Page (skip page\*pageSize items)

\* @param order The way to order

\* @param model Model

\*/

public String search(

@RequestParam(name="minStars", required=false, defaultValue="0") int minStars,

@RequestParam(name="maxStars", required=false, defaultValue="9999999") int maxStars,

@RequestParam(name="minPrice", required=false, defaultValue="0") int minPrice,

@RequestParam(name="maxPrice", required=false, defaultValue="9999999") int maxPrice,

@RequestParam(name="city", required=false, defaultValue="") String city,

@RequestParam(name="id", required=false, defaultValue="-1") int id,

@RequestParam(name="page", required=false, defaultValue="0") int page,

@RequestParam(name="order", required=false, defaultValue="stars") String order,

Model model

) {

Filter f = new FilterAlways();

Filter filter = f;

filter = filter.then(new FilterIsBooked(false));

filter = filter.then(new FilterStars(minStars, maxStars));

filter = filter.then(new FilterPrice(minPrice, maxPrice));

if(city.length() != 0) filter = filter.then(new FilterCity(city));

if(id != -1) filter = filter.then(new FilterId(id));

Database db = Database.get();

List<HotelRoom> result = db.getFiltered(f);

Iterator iterator;

if(order.equals("stars")) iterator = new IteratorStars(result);

else if(order.equals("price")) iterator = new IteratorPrice(result);

else iterator = new IteratorStars(result);

iterator.skipn(pageSize \* page);

List<HotelRoom> rooms = iterator.nextn(pageSize);

String component = "<div class=\"listRoom\"><a href=\"/room?id=%d\"><h2>Hotel %s at city %s</h2></a> <p>★ %d</p> <p>$ %d</p></div>";

String roomComponents = "";

for(HotelRoom room : rooms) {

roomComponents += String.format(component, room.id, room.hotelName, room.city, room.stars, room.price);

}

model.addAttribute("rooms", roomComponents);

return "search";

}

@GetMapping("/room")

/\*\*

\* The route to check the details of the room

\* @param id The room id

\* @param model Model

\*/

public String room(

@RequestParam(name="id", required=false, defaultValue="0") int id,

Model model

) {

if(id == 0) return "index"; // TODO: error

Filter filter = new FilterId(id);

Database db = Database.get();

List<HotelRoom> result = db.getFiltered(filter);

if(result.size() <= 0) {

return "index"; // TODO: error?

}

HotelRoom room = result.get(0);

model.addAttribute("number", room.roomNumber);

model.addAttribute("stars", room.stars);

model.addAttribute("price", room.price);

model.addAttribute("city", room.city);

model.addAttribute("hotelName", room.hotelName);

return "room";

}

@GetMapping("/book")

/\*\*

\* A route that books the room and redirects you to home page

\* @param id The id of the room to book

\* @param model Model

\*/

public String book(

@RequestParam(name="id", required=false, defaultValue="0") int id,

Model model

) {

if(id == 0) return "index"; // TODO: error

Filter filter = new FilterId(id);

Database db = Database.get();

List<HotelRoom> result = db.getFiltered(filter);

if(result.size() <= 0) {

return "index"; // TODO: error?

}

HotelRoom room = result.get(0);

if(room.isBooked) return "index";

// NOTE: contact the hotel here

room.isBooked = true;

db.saveDatabase();

return "index";

}

@GetMapping("/posthotel")

/\*\*

\* A route to the form to post your room

\* @param model Model

\*/

public String postHotelGet(Model model) {

model.addAttribute("hotelRoom", new HotelRoom());

return "posthotel";

}

@PostMapping("/posthotel")

/\*\*

\* A route to post your room to the database

\* @param hotelRoom The room to be posted

\* @param model Model

\*/

public String postHotelSubmit(@ModelAttribute HotelRoom hotelRoom, Model model) {

Database db = Database.get();

hotelRoom.isBooked = false;

db.addRoom(hotelRoom);

return "index";

}

@GetMapping("/home")

/\*\*

\* The route to the home page

\*/

public String home() {

return "index";

}

@GetMapping("/DEBUG\_POPULATE")

/\*\*

\* The debug route to generate some data

\*/

public String debugGenerate() {

Random rng = new Random();

Database db = Database.get();

String[] hotelWords = { "Aboba", "Kamaz", "Pivo", "Chupa", "Zebra", "Kaban", "Cool", "Amogus", "Guga", "Krab", "Cabra", "Kavun", "Zhizha" };

String[] cities = { "Kyiv", "New York", "Odessa", "Tokyo", "Beijing", "Berlin", "Rome", "Madrid", "Chernobyl", "Kabanograd", "Aboba town" };

for(int hotel = 0; hotel < 30; hotel++) {

String word1 = hotelWords[rng.nextInt(hotelWords.length)];

String word2 = hotelWords[rng.nextInt(hotelWords.length)];

String hotelName = word1 + word2;

int amountOfRooms = rng.nextInt(5) + 5;

int maxStars = rng.nextInt(3) + 6;

int priceMultiplier = rng.nextInt(3) + 10;

int roomOffset = rng.nextInt(30) + 30;

for(int roomIndex = 0; roomIndex < amountOfRooms; roomIndex++) {

HotelRoom room = new HotelRoom();

room.hotelName = hotelName;

room.city = cities[rng.nextInt(cities.length)];

room.stars = rng.nextInt(maxStars) + 1;

room.price = (rng.nextInt(4) + 20) \* priceMultiplier;

room.isBooked = false;

room.roomNumber = roomOffset++;

db.addRoom(room);

}

}

return "index";

}

}

Database.java

package work9;

import com.google.gson.\*;

import java.util.List;

import java.util.Arrays;

import java.util.ArrayList;

import org.apache.commons.io.FileUtils;

import java.io.File;

import java.lang.Exception;

import java.nio.charset.Charset;

import java.util.Random;

/\*\*

\* A class that decribes a database stored in a file

\*/

public class Database {

/\*\*

\* A random number generator for internal use

\*/

private static Random random;

/\*\*

\* A singleton instance of the database

\*/

private static Database instance;

/\*\*

\* A private constructor to ensure the singleton

\*/

private Database() {}

/\*\*

\* The path where the database is stored

\*/

private static final String databasePath = "database.json";

/\*\*

\* A method to get (or load) the database

\* @return The database

\*/

public static Database get() {

if(instance == null) loadDatabase();

return instance;

}

/\*\*

\* A method to load the database

\*/

private static void loadDatabase() {

try{

random = new Random();

instance = new Database();

File file = new File(databasePath);

file.createNewFile();

String json = FileUtils.readFileToString(file);

if(json.length() == 0) { json = "[]"; }

Gson g = new Gson();

HotelRoom[] data = g.fromJson(json, HotelRoom[].class);

instance.data = new ArrayList<>(Arrays.asList(data));

}catch(Exception e) { System.out.println("I couldn't care less"); System.exit(1); }

}

/\*\*

\* The internal representation of the database

\*/

private List<HotelRoom> data;

/\*\*

\* The method to get entries from the database that pass the filter

\* @param filter The filter being used

\* @return The data that passed the filter

\*/

public List<HotelRoom> getFiltered(Filter filter) {

List<HotelRoom> result = new ArrayList<>();

for(HotelRoom room : data) {

if(filter.filter(room)) result.add(room);

}

return result;

}

/\*\*

\* A method to add the room to the data and back up the database

\* @param room The room to be added

\*/

public void addRoom(HotelRoom room) {

room.id = random.nextInt(1000000) + 1;

data.add(room);

saveDatabase();

}

/\*\*

\* A method to save the database

\*/

public void saveDatabase() {

File file = new File(databasePath);

Gson g = new Gson();

HotelRoom[] array = new HotelRoom[data.size()];

data.toArray(array);

String json = g.toJson(array);

try{

FileUtils.writeStringToFile(file, json, Charset.forName("UTF-8"));

}catch(Exception e) { System.out.println(e.toString()); System.exit(1); }

}

}

static/book.js

function book() {

const urlParams = new URLSearchParams(window.location.search);

const roomId = urlParams.get("id");

location.href = `/book?id=${roomId}`;

}

static/index.html

<!DOCTYPE HTML>

<html>

<head>

<title>Very cool booking website</title>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

<link rel="stylesheet" href="style.css">

</head>

<body>

<a href="/home"><h1>Home</h1></a>

<h1>Book only from this website!!!</h1>

<div><a href="/search">Search here</a></div>

<div><a href="/posthotel">If you are a hotel click here</a></div>

</body>

</html>

static/search.js

let minPrice = undefined;

let maxPrice = undefined;

let minStars = undefined;

let maxStars = undefined;

let city = undefined;

let id = undefined; // room id, ignore all other args

let page = localStorage.getItem("page");

if(page === null) page = 0;

let order = undefined; // sorting order

function populateInputs() {

minPrice = document.getElementById("minPrice").value;

if(isNaN(parseFloat(minPrice))) minPrice = undefined;

maxPrice = document.getElementById("maxPrice").value;

if(isNaN(parseFloat(maxPrice))) maxPrice = undefined;

minStars = document.getElementById("minStars").value;

if(isNaN(parseFloat(minStars))) minStars = undefined;

maxStars = document.getElementById("maxStars").value;

if(isNaN(parseFloat(maxStars))) maxStars = undefined;

city = document.getElementById("city").value;

if(city.length === 0) city = undefined;

id = document.getElementById("roomId").value;

if(isNaN(parseInt(id))) id = undefined;

}

function applyFilters() {

populateInputs();

let link = "/search?";

let selectedOrder = document.querySelector('input[name="searchOrder"]:checked');

let order = "";

if(selectedOrder === null) order = undefined;

else order = selectedOrder.value;

console.log(order);

if(id !== undefined) link += `id=${id}`;

else {

if(minPrice !== undefined) link += `minPrice=${minPrice}&`;

if(maxPrice !== undefined) link += `maxPrice=${maxPrice}&`;

if(minStars !== undefined) link += `minStars=${minStars}&`;

if(maxStars !== undefined) link += `maxStars=${maxStars}&`;

if(city !== undefined) link += `city=${city}&`;

if(page !== undefined) link += `page=${page}&`;

if(order !== undefined) link += `order=${order}&`;

}

location.href = link;

}

function search() {

applyFilters();

localStorage.setItem("page", 0);

}

function nextPage() {

page++;

localStorage.setItem("page", page);

applyFilters();

}

function prevPage() {

page--;

if(page < 0) page = 0;

localStorage.setItem("page", page);

applyFilters();

}

static/style.css

.searchFilterBlock {

display: flex;

flex-direction: row;

}

.roomInfoItem {

display: flex;

flex-direction: row;

}

#searchRadios {

display: flex;

flex-direction: row;

}

.searchFilterBlock > \* {

padding: 10px;

}

.listRoom {

border: solid black;

padding: 20px;

margin-top: 10px;

margin-bottom: 10px;

}

#searchButtons {

max-width: fit-content;

margin-left: auto;

margin-right: auto;

}

#searchRadios {

margin-top: 20px;

margin-bottom: 20px;

max-width: fit-content;

margin-left: auto;

margin-right: auto;

}

templates/index.html

<!DOCTYPE HTML>

<html>

<head>

<title>Very cool booking website</title>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

<link rel="stylesheet" href="style.css">

</head>

<body>

<a href="/home"><h1>Home</h1></a>

<h1>Book only from this website!!!</h1>

<div><a href="/search">Search here</a></div>

<div><a href="/posthotel">If you are a hotel click here</a></div>

</body>

</html>

templates/posthotel.html

<!DOCTYPE HTML>

<html xmlns:th="http://www.thymeleaf.org">

<head>

<title>Post room</title>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

<link rel="stylesheet" href="style.css">

</head>

<body>

<a href="/home"><h1>Home</h1></a>

<h1>Post your very good room here!</h1>

<form method="post" action="#" th:action="@{/posthotel}" th:object="${hotelRoom}">

<p>name</p> <input type="text" th:field="\*{hotelName}"/>

<p>city</p> <input type="text" th:field="\*{city}"/>

<p>price</p> <input type="text" th:field="\*{price}"/>

<p>number</p> <input type="text" th:field="\*{roomNumber}"/>

<input type="submit" value="post"/>

</form>

</body>

</html>

templates/room.html

<!DOCTYPE HTML>

<html>

<head>

<title>Your very good room</title>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

<script src="book.js"></script>

<link rel="stylesheet" href="style.css">

</head>

<body>

<a href="/home"><h1>Home</h1></a>

<h1>A very cool room</h1>

<div class="roomInfoItem">Room number: <div th:utext="${number}" /></div>

<div class="roomInfoItem">★ <div th:utext="${stars}" /></div>

<div class="roomInfoItem">$ <div th:utext="${price}" /></div>

<div class="roomInfoItem">City: <div th:utext="${city}" /></div>

<div class="roomInfoItem">Hotel: <div th:utext="${hotelName}" /></div>

<button onclick="book()">Book</button>

</body>

</html>

templates/search.html

<!DOCTYPE HTML>

<html xmlns:th="http://www.thymeleaf.org">

<head>

<title>Search</title>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

<script src="search.js"></script>

<link rel="stylesheet" href="style.css">

</head>

<body>

<a href="/home"><h1>Home</h1></a>

<h1>Search for your room here!</h1>

<div class = "searchFilterBlock">

<div class="searchFilterBlock"><p>Min price</p><input id="minPrice" type="number"></input></div>

<div class="searchFilterBlock"><p>Max price</p><input id="maxPrice" type="number"></input></div>

</div>

<div class = "searchFilterBlock">

<div class="searchFilterBlock"><p>Min stars</p><input id="minStars" type="number"></input></div>

<div class="searchFilterBlock"><p>Max stars</p><input id="maxStars" type="number"></input></div>

</div>

<div class = "searchFilterBlock">

<div class="searchFilterBlock"><p>City</p><input id="city" type="text"></input></div>

<div class="searchFilterBlock"><p>Room Id</p><input id="roomId" type="number"></input></div>

</div>

<div id="searchRadios">

<div>

<input type="radio" id="searchRadio\_stars" name="searchOrder" value="stars"/>

<label for="searchRadio\_stars">Sort by stars</label>

</div>

<div>

<input type="radio" id="searchRadio\_price" name="searchOrder" value="price"/>

<label for="searchRadio\_price">Sort by price</label>

</div>

</div>

<div id="searchButtons">

<button onclick="prevPage()"> &lt; </button>

<button onclick="search()">Search</button>

<button onclick="nextPage()"> &gt; </button>

</div>

<div>

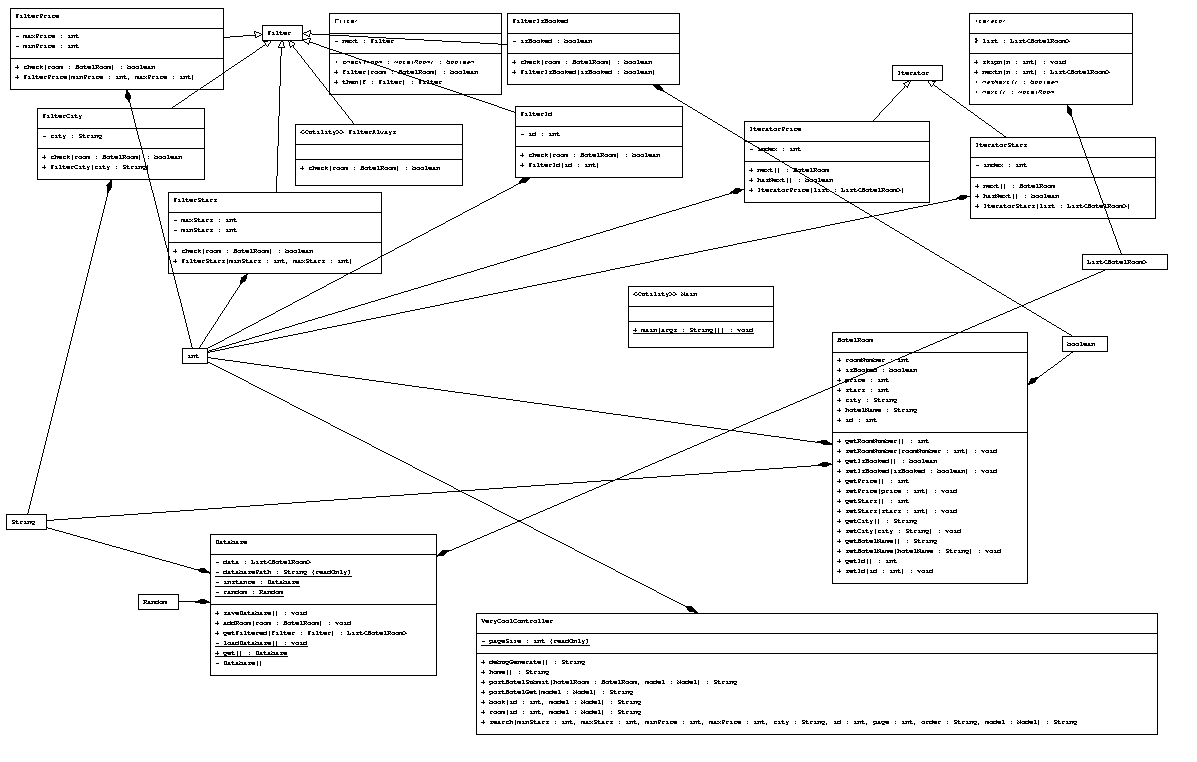
<div th:utext="${rooms}" />

</div>

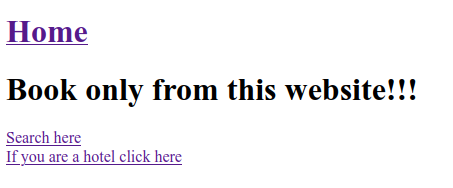
</body>

</html>

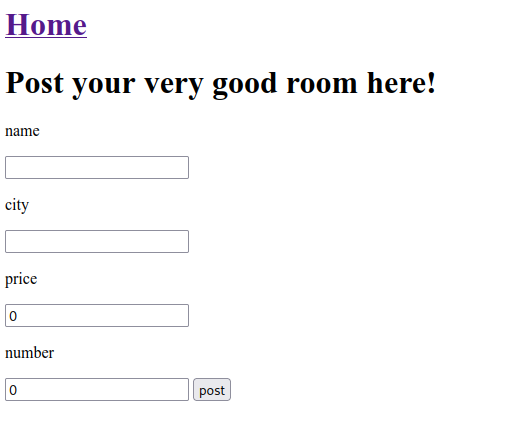
**Згенерована за кодом діаграма:**

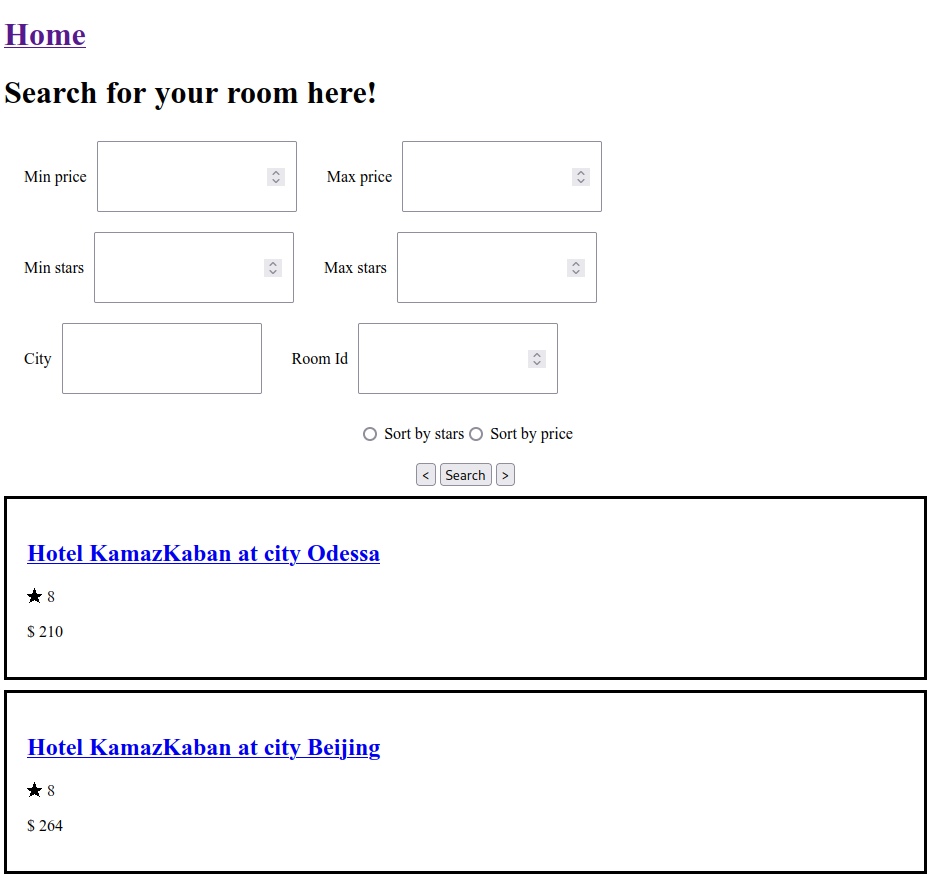
  
У цьому проекті були застосовані три паттерни — Singleton (підтримання лише однієї бази даних), Iterator (різні способи сортування кімнат, за зірками, отзивами або за ціною), та Chain of Responsibility (фільтрування кімнат та композиція окремих фільтрів у ланцюг)

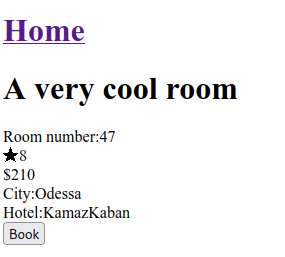
**Згенерована документація:**



**Приклади роботи програми:**







**Висновки:** Під час даного розробленого проекту, навчився працювати з фреймворком Spring Boot, а також навчився реалізовувати архітектуру «клієнт-сервер». Була спроектована програмна система, підключена та добудована база даних, архітектура (всього проекту) відповідає програмній моделі «Клієнт-Сервер», тож можна змінювати та модифікувати в подальшому проект відповідно до завдання.