**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**

**імені ІГОРЯ СІКОРСЬКОГО»**

факультет інформатики та обчислювальної техніки

(повна назва інституту/факультету)

кафедра інформаційних систем та технологій

(повна назва кафедри)

|  |  |
| --- | --- |
|  |  |

**Курсова робота**

з дисципліни «Програмування-2»

на тему: Сайт прогноз погоди

Виконав : студент \_\_1\_ курсу, групи \_\_\_\_ІА-14\_\_\_\_\_

(шифр групи)

Паршин Марк Віталійович \_\_\_\_\_\_\_\_\_

(прізвище, ім’я, по батькові) (підпис)

Науковий керівник

асистент інформаційних систем та технологій В.О. Міщенко \_\_\_\_\_\_\_\_\_

(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Члени комісії \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_

(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_

(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.

Студент

Паршин.М.В. \_\_\_\_\_\_\_\_\_

(підпис)

Київ – 2020

ЗМІСТ

[ВСТУП 3](#_Toc105442119)

[1 ВИМОГИ ДО СИСТЕМИ 4](#_Toc105442120)

[1.1 Функціональні вимоги: 4](#_Toc105442121)

[1.2 Нефункціональні вимоги до системи 4](#_Toc105442122)

[2.1 Діаграма прецедентів 5](#_Toc105442123)

[2.2 Опис сценаріїв використання системи 6](#_Toc105442124)

[3 АРХІТЕКТУРА СИСТЕМИ 10](#_Toc105442125)

[4 РЕАЛІЗАЦІЯ КОМПОНЕНТІВ СИСТЕМИ 12](#_Toc105442126)

[4.1 Загальна структура проекту 12](#_Toc105442127)

[4.2 Компоненти рівня доступу до даних 13](#_Toc105442128)

[4.3 Компоненти рівня бізнес-логіки 14](#_Toc105442129)

[4.4 Компоненти рівня інтерфейсу користувача 15](#_Toc105442130)

[ВИСНОВКИ 16](#_Toc105442131)

[ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ 18](#_Toc105442132)

[ДОДАТКИ 19](#_Toc105442133)

# ВСТУП

Прогнози синоптиків про зміну температури, можливих опадах дозволяють людям вибрати одяг для вулиці, планувати заміські поїздки, інші захід, які проводяться на свіжому повітрі. Найпопулярнішим інформаційним джерелом зараз є інтернет, тому метеослужби різних країн розміщують дані про погоду на спеціальних веб-порталах. Для кожного користувача важлива достовірність наданого матеріалу, специфіка його подачі.

Зручність і швидкість таких сайтів завжди у пріорітеті для користувача, адже всі хочуть бачити прогноз не тільки на сьогодні, а й на інші дні, на інші періоди, які вкаже користувач.

Метою роботи є створення сайту, що дозволяє людям виконувати такий перелік дій та вирішувати такі задачі:

* Вхід у власний користувацький аккаунт
* Перегляд певної погоди за вказаний день чи період, у певній локації, місті, області.

# 1 ВИМОГИ ДО СИСТЕМИ

## 1.1 Функціональні вимоги:

* Вхід у аккаунт
* Пошук погоди у вказаній локації за вказаний день
* Пошук погоди у вказаній локації за вказаний період

## 1.2 Нефункціональні вимоги до системи

* Система має відповідати наступним функціональним вимогам:
* система повинна мати відкриту архітектуру;
* система повинна мати веб-інтерфейс;
* інтерфейс користувача має бути зручним та інтуїтивно-зрозумілим;
* система повинна бути крос-платформною.

2 СЦЕНАРІЙ ВИКОРИСТАННЯ

## 2.1 Діаграма прецедентів

Діаграма прецедентів системи представлена на рис. 2.1.

Акторами виконання є користувач та адміністратор, взаємодії наведені нижче.

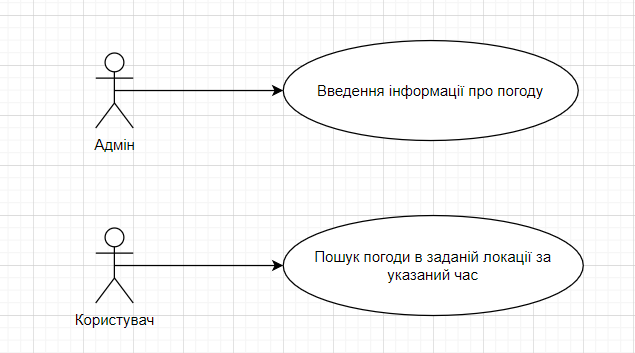


Рис 2.1. Діаграма прецендентів

## 2.2 Опис сценаріїв використання системи

Детальні описи сценаріїв використання наведено у таблицях 2.1 – 2.3

Таблиця 2.1 – Сценарій використання «Вхід у аккаунт»

|  |  |
| --- | --- |
| Назва | Вхід в аккаунт |
| ID | 1 |
| Опис | Використовуючи форму для входу у аккаунт взаємодіє з входом для входу відповідно. |
| Актори | Гість |
| Вигоди компанії | Якщо користувач не зайшов у аккаунт він не зможе взаємодіяти з функціоналом сайту |
| Частота користування | Постійно |
| Тригери | Користувач вводить дані у форму |
| Передумови | Форма доступна на початковому екрані |
| Пост-умови | Користувач потрапляє на сторінку де може взаємодіяти зі всім доступним функціоналом |
| Основний розвиток | Гість натискає вхід та вводить свої дані для входу у аккаунт |
| Альтернативні розвитки | – |
| Виняткові ситуації | Неправильно введені дані, потраплення до сторінки помилок з відповідним статусом |

Таблиця 2.2 – Сценарій використання «Пошук погоди»

|  |  |
| --- | --- |
| Назва | Пошук погоди |
| ID | 2 |
| Опис | Користувач шукає погоду |
| Актори | Користувач |
| Вигоди компанії | Головна ціль сайту – пошук погоди |
| Частота користування | Постійно |
| Тригери | Користувач вводить дані у форму для пошуку |
| Передумови | Введення даних у форму, та нажаття кнопки пошуку |
| Пост-умови |  |
| Основний розвиток | Користувач натискає на пошук, вводить назву та період і отримує список погоди по запиту |
| Альтернативні розвитки | – |
| Виняткові ситуації | Якщо погода не вказана за певний період показується «Немає даних» |

Таблиця 2.3 – Сценарій використання «Додавання погоди»

|  |  |
| --- | --- |
| Назва | Додавання погоди |
| ID | 3 |
| Опис | Адміністратор додає погоду |
| Актори | Адміністратор |
| Вигоди компанії | Швидке наповнення сайту потрібною інформацією про погоду |
| Частота користування | Постійно |
| Тригери | Адміністратор відкриває сторінку Адмін панель |
| Передумови | Сторінка доступна при нажатті кнопки Адмін панель(Тільки якщо користувач є адміністратором) |
| Пост-умови | - |
| Основний розвиток | Адміністратор натискає адмін панель, заповняє дані та натискає додати |
| Альтернативні розвитки | – |
| Виняткові ситуації | - |

# 3 АРХІТЕКТУРА СИСТЕМИ

Загальна архітектура системи наведена на Рис 3.1

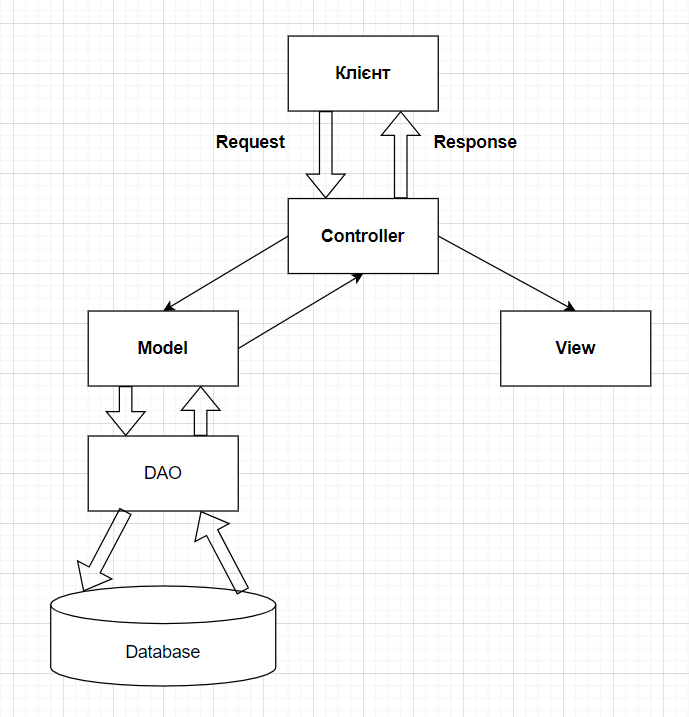


Рис 3.1 Загальна архітектура системи

Система складається з наступних елементів:

- графічний інтерфейс;

- серверна частина;

- база даних.

Графічний інтерфейс необхідний для взаємодії з користувачем. HTTP запит надходить до серверної частини, де оброблюється і повертається відповідь. На серверній частині виконується основна логіка системи. Дані, отриманні з графічного інтерфейсу конвертуються. Також, серверна частина формує запит до бази даних та оброблює відповідь і передає її до графічного інтерфейсу. База даних зберігає дані, які були сформовані на серверній частині та повертає їх у разі запиту. Схема зображена на Рис 3.1.

До серверної частини належать наступні елементи:

- Контролер;

- Модель та вигляд;

-D AO;

- База даних.

На контролер надходять дані з графічного інтерфейсу. З контролеру, дані формуються для запиту в ДАО. З репозиторію дані надсилаються до бази даних і зберігаються або виймаються. Також в контролері формується вид, тобто об’єкт і його ім’я для відображення на графічному інтерфейсі.

# 4 РЕАЛІЗАЦІЯ КОМПОНЕНТІВ СИСТЕМИ

## 4.1 Загальна структура проекту

Загальна структура проекту представлена на рис.4.1

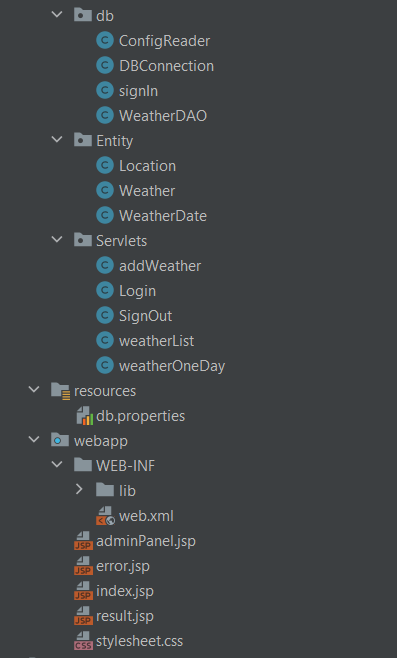


Рисунок 4.1 – Загальна структура проекту

Проект складається з веб-ресурсів, бібліотек, та вихідного коду, який в свою чергу можна поділити на компоненти рівня доступу до даних, компоненти бізнес-логіки та веб-компоненти.

## 4.2 Компоненти рівня доступу до даних

Основні сутності та інтерфейси рівня доступу до даних наведені на рис. 4.2

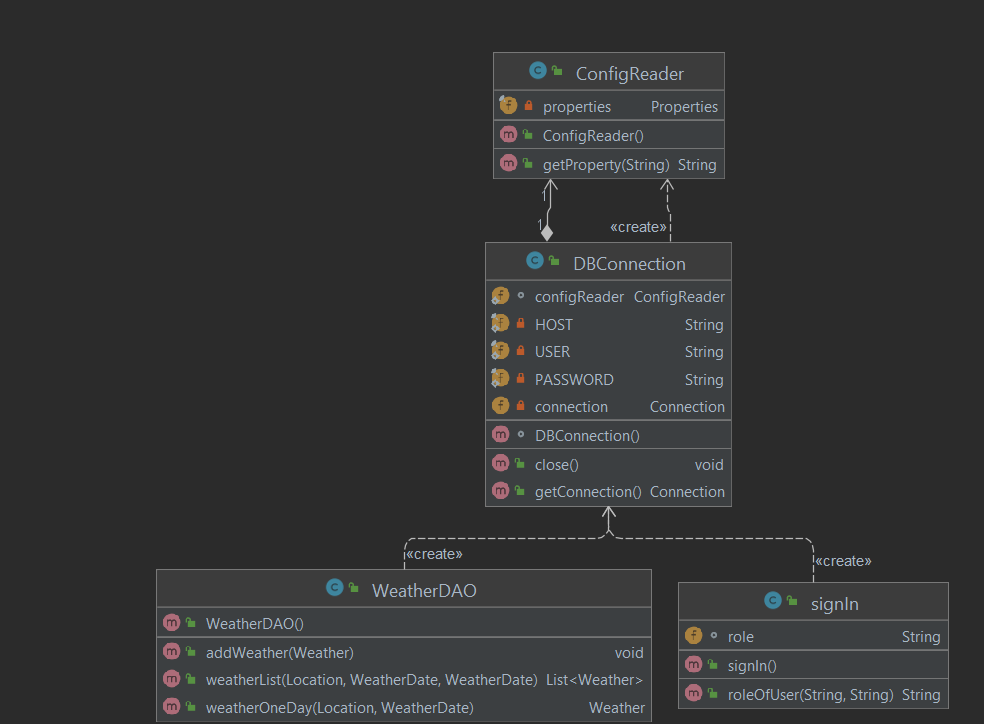
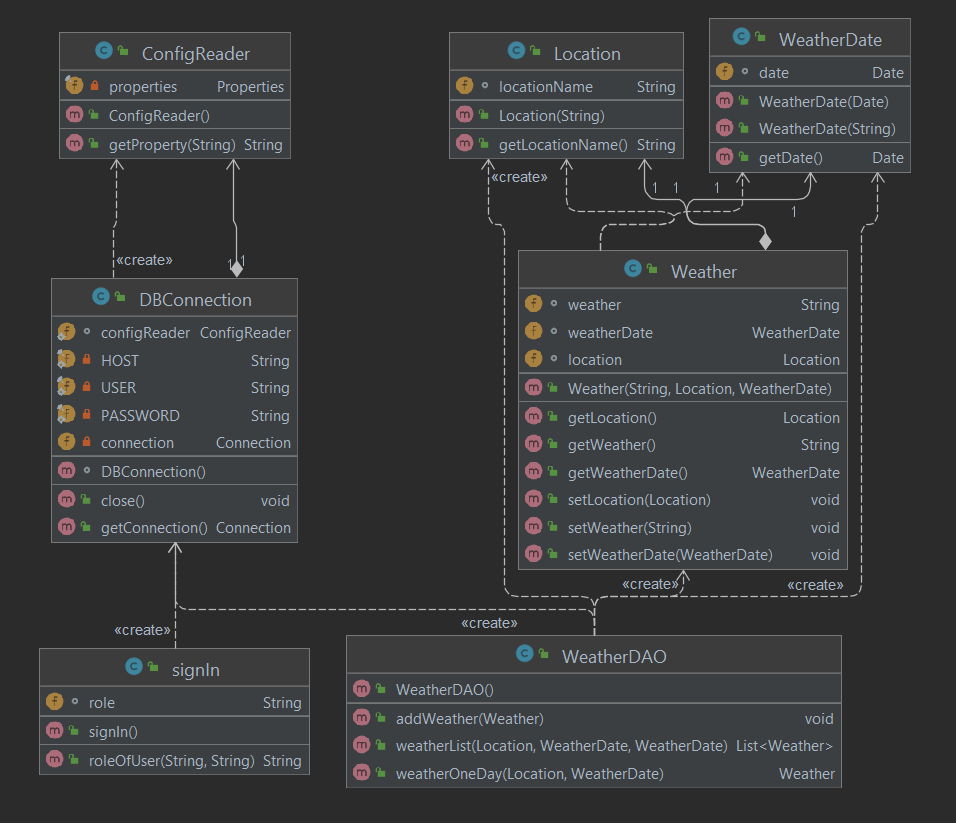
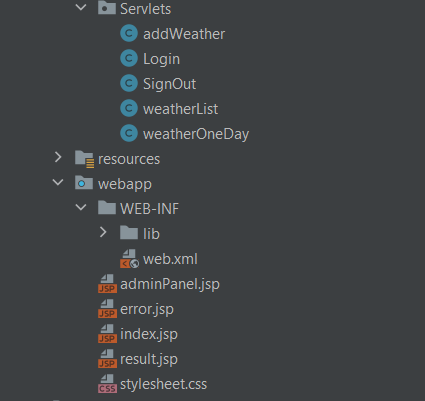


Рисунок 4.2 – Основні сутності та інтерфейси рівня доступу до даних

## 4.3 Компоненти рівня бізнес-логіки



## 4.4 Компоненти рівня інтерфейсу користувача



JSP + JSTL використовується для виведення результатів для користувача

Також прикріплений файл стилів, що дозволяє користувачу бачити приємніший інтерфейс



# ВИСНОВКИ

Під час виконання роботи було використано ідею сайту sinoptik, проаналізовано переваги та недоліки цієї ідеї та задумки, виконання основної роботи та проаналізований подальший розвиток проекту.

Є ще багато рішень для зручнішого користування сайтом, тому у подальшому розвитку проекту планується додати вибір погоди у певні години, показання погоди на карті та інші зручні функції, також проект має ще багато способів для доповнення та покращення.

Сам розвиток був за такими пунктами:

1. Формулювння функціональні та нефункціональні вимоги до системи, що визначило очікувану поведінку системи.
2. Обрано технології на яких буде написана система. Java була обрана в якості мови програмування, середа розробки – IntelliJ IDEA за її можливості та потужність. В якості системи управління базами даних було вирішено використовувати MySQL за легкість її налаштування та підтримки.
3. Обдумані сценарії використання, та можливі нові функції для подальшого збільшення проекту.
4. Розраховуючи загальну архітектуру було вибранно використання шаблону проектування такого як – MVC pattern, через його зручність та безпечність за рахунок розділення логічних рівнів.В моєму випадку таких як Controller(Сервлети для взаємодії користувача з сервером), Model(Для взаємодії з DAO, що в свою чергу взаємодіє з базою даних), View(Для взаємодії з користувацьким інтерфейсом), та наявністю 2 інтерфейсів, для можливості зручної заміни модуля(DAO).
5. В кінці виконання всіх попередніх пунктів розвитку проекту стало підбирання стилів, для приємного відображення користувацького інтерфейсу. Використовуючи стилі CSS.

# ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1.Что такое MVC [Електронний ресурс] – https://ru.hexlet.io/blog/posts/chto-takoe-mvc-rasskazyvaem-prostymi-slovami

2. Руководство Java JSP для начинающих [Електронний ресурс] –https://betacode.net/10263/java-jsp

3. Інформація про сайти погод та аналітика - https://pishohid.org.ua/13934/кращі-сайти-точного-прогнозу-погоди/

# ДОДАТКИ

package db;  
  
import java.io.IOException;  
import java.io.InputStream;  
import java.util.Properties;  
  
public class ConfigReader {  
  
 private final Properties properties = new Properties();  
  
 public ConfigReader() {  
 String filePath = "db.properties";  
 try (InputStream out = getClass().getClassLoader().getResourceAsStream(filePath)) {  
 properties.load(out);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
 }  
  
 public String getProperty(String propertyName){  
 return properties.getProperty(propertyName);  
 }  
}

package db;  
  
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.SQLException;  
  
public class DBConnection implements AutoCloseable{  
 static ConfigReader configReader = new ConfigReader();  
 private static final String HOST = configReader.getProperty("HOST");  
 private static final String USER = configReader.getProperty("USER");  
 private static final String PASSWORD = configReader.getProperty("PASSWORD");  
 private Connection connection = null;  
  
 DBConnection() {  
 try {  
 connection = DriverManager.getConnection(HOST, USER, PASSWORD);  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public Connection getConnection() {  
 return connection;  
 }  
  
  
 @Override  
 public void close() throws Exception {  
 connection.close();  
 }  
}

package db;  
  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
  
public class signIn {  
 String role;  
  
 public String roleOfUser(String login, String password) {  
 try (DBConnection db = new DBConnection()) {  
 String query = "SELECT role FROM sinoptik.users where login = ? and password = ?;";  
 PreparedStatement preparedStatement = db.getConnection().prepareStatement(query);  
 preparedStatement.setString(1, login);  
 preparedStatement.setString(2, password);  
 ResultSet result = preparedStatement.executeQuery();  
 if (result.next()) {  
 role=result.getString("role");  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 return role;  
 }  
  
}

package db;  
  
import Entity.Location;  
import Entity.Weather;  
import Entity.WeatherDate;  
  
import java.sql.Connection;  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
import java.util.ArrayList;  
import java.util.List;  
  
public class WeatherDAO {  
 public List<Weather> weatherList(Location location, WeatherDate date1, WeatherDate date2) {  
 List<Weather> weatherList = new ArrayList<>();  
 String query = "SELECT l.locationName, wd.date, w.weather FROM sinoptik.weather w\n" +  
 "INNER JOIN sinoptik.location l On w.idlocation=l.idlocation\n" +  
 "INNER JOIN sinoptik.weatherdate wd on w.idweatherDate=wd.idweatherDate\n" +  
 "where l.locationName=? and wd.date between ? and ?" +  
 "order by wd.date;";  
 try (DBConnection db = new DBConnection()) {  
 Connection connection = db.getConnection();  
 PreparedStatement preparedStatement = connection.prepareStatement(query);  
 preparedStatement.setString(1, location.getLocationName());  
 preparedStatement.setDate(2, date1.getDate());  
 preparedStatement.setDate(3, date2.getDate());  
 ResultSet resultSet = preparedStatement.executeQuery();  
 while (resultSet.next()) {  
 Location locationName = new Location(resultSet.getString(1));  
 WeatherDate date = new WeatherDate(resultSet.getDate(2));  
 String weather = resultSet.getString(3);  
 Weather weatherToList = new Weather(weather, locationName, date);  
 weatherList.add(weatherToList);  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 return weatherList;  
 }  
  
 public Weather weatherOneDay(Location location, WeatherDate date1) {  
 String query = "SELECT l.locationName, wd.date, w.weather FROM sinoptik.weather w\n" +  
 "INNER JOIN sinoptik.location l On w.idlocation=l.idlocation\n" +  
 "INNER JOIN sinoptik.weatherdate wd on w.idweatherDate=wd.idweatherDate\n" +  
 "where l.locationName=? and wd.date=?";  
 try (DBConnection db = new DBConnection()) {  
 Connection connection = db.getConnection();  
 PreparedStatement preparedStatement = connection.prepareStatement(query);  
 preparedStatement.setString(1, location.getLocationName());  
 preparedStatement.setDate(2, date1.getDate());  
 ResultSet resultSet = preparedStatement.executeQuery();  
 if (resultSet.next()) {  
 Location locationName = new Location(resultSet.getString(1));  
 WeatherDate date = new WeatherDate(resultSet.getDate(2));  
 String weather = resultSet.getString(3);  
 Weather weatherToReturn = new Weather(weather, locationName, date);  
 return weatherToReturn;  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 return null;  
 }  
  
 public void addWeather(Weather weather) {  
 String query1="INSERT IGNORE INTO sinoptik.location (locationName) VALUES(?);";  
 String query2="INSERT IGNORE INTO sinoptik.weatherdate (date) VALUES(?);";  
 String query3 = "INSERT INTO sinoptik.weather(idlocation, idweatherDate, weather) VALUES\n" +  
 " ((SELECT idlocation From sinoptik.location where locationName=?)," +  
 "(SELECT idweatherDate From sinoptik.weatherdate where date=?),?);";  
 try (DBConnection db = new DBConnection()) {  
 Connection connection = db.getConnection();  
 PreparedStatement preparedStatement1 = connection.prepareStatement(query1);  
 PreparedStatement preparedStatement2 = connection.prepareStatement(query2);  
 PreparedStatement preparedStatement3 = connection.prepareStatement(query3);  
 preparedStatement1.setString(1, weather.getLocation().getLocationName());  
 preparedStatement2.setDate(1, weather.getWeatherDate().getDate());  
 preparedStatement3.setString(1, weather.getLocation().getLocationName());  
 preparedStatement3.setDate(2,weather.getWeatherDate().getDate());  
 preparedStatement3.setString(3, weather.getWeather());  
 preparedStatement1.executeUpdate();  
 preparedStatement2.executeUpdate();  
 preparedStatement3.executeUpdate();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
}

package Entity;  
  
public class Location {  
 String locationName;  
  
 public String getLocationName() {  
 return locationName;  
 }  
  
 public Location(String locationName) {  
 this.locationName = locationName;  
 }  
}

package Entity;  
  
public class Weather {  
 String weather;  
 WeatherDate weatherDate;  
 Location location;  
  
  
 public String getWeather() {  
 return weather;  
 }  
  
 public void setWeather(String weather) {  
 this.weather = weather;  
 }  
  
 public Location getLocation() {  
 return location;  
 }  
  
 public void setLocation(Location location) {  
 this.location = location;  
 }  
  
 public WeatherDate getWeatherDate() {  
 return weatherDate;  
 }  
  
 public void setWeatherDate(WeatherDate weatherDate) {  
 this.weatherDate = weatherDate;  
 }  
  
 public Weather(String weather, Location location, WeatherDate weatherDate) {  
 this.weather = weather;  
 this.location = location;  
 this.weatherDate = weatherDate;  
 }  
}

package Entity;  
  
  
import java.sql.Date;  
  
public class WeatherDate {  
 Date date;  
  
 public java.sql.Date getDate() {  
 return date;  
 }  
  
 public WeatherDate(String date) {  
 this.date = java.sql.Date.valueOf(date);  
 }  
 public WeatherDate(Date date) {  
 this.date = date;  
 }  
  
}

package Servlets;  
  
import Entity.Location;  
import Entity.Weather;  
import Entity.WeatherDate;  
import db.WeatherDAO;  
  
import javax.servlet.\*;  
import javax.servlet.http.\*;  
import javax.servlet.annotation.\*;  
import java.io.IOException;  
  
@WebServlet(name = "addWeather", value = "/addWeather")  
public class addWeather extends HttpServlet {  
 @Override  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
  
 }  
  
 @Override  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 if (request.getSession().getAttribute("role").equals("admin")) {  
 Location location = new Location(request.getParameter("location"));  
 WeatherDate weatherDate = new WeatherDate(request.getParameter("date"));  
 Weather weather = new Weather(request.getParameter("weather"), location, weatherDate);  
 WeatherDAO weatherDAO = new WeatherDAO();  
 weatherDAO.addWeather(weather);  
 request.getRequestDispatcher("adminPanel.jsp").forward(request,response);  
 } else{  
 request.getSession().setAttribute("status", "Not admin, plz login");  
 request.getRequestDispatcher("error.jsp").forward(request,response);  
 }  
 }  
}

package Servlets;  
  
import db.signIn;  
  
import javax.servlet.\*;  
import javax.servlet.http.\*;  
import javax.servlet.annotation.\*;  
import java.io.IOException;  
  
@WebServlet(name = "Login", value = "/Login")  
public class Login extends HttpServlet {  
 @Override  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
  
 }  
  
 @Override  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 String login = request.getParameter("login");  
 String password = request.getParameter("password");  
 signIn signIn = new signIn();  
 String role = signIn.roleOfUser(login, password);  
 if (role != null) {  
 request.getSession().setAttribute("role", role);  
 request.getSession().setAttribute("logged", true);  
 request.getRequestDispatcher("index.jsp").forward(request, response);  
 } else {  
 request.getSession().setAttribute("status", "BAD LOGIN");  
 request.getRequestDispatcher("error.jsp").forward(request, response);  
 }  
  
 }  
}

package Servlets;  
  
import javax.servlet.\*;  
import javax.servlet.http.\*;  
import javax.servlet.annotation.\*;  
import java.io.IOException;  
  
@WebServlet(name = "SignOut", value = "/SignOut")  
public class SignOut extends HttpServlet {  
 @Override  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 request.getSession().setAttribute("logged", false);  
 request.getSession().setAttribute("role", null);  
 request.getRequestDispatcher("/index.jsp").forward(request,response);  
 }  
  
 @Override  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
  
 }  
}

package Servlets;  
  
import Entity.Location;  
import Entity.WeatherDate;  
import db.WeatherDAO;  
  
import javax.servlet.\*;  
import javax.servlet.http.\*;  
import javax.servlet.annotation.\*;  
import java.io.IOException;  
  
@WebServlet(name = "weatherList", value = "/weatherList")  
public class weatherList extends HttpServlet {  
 @Override  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
  
 }  
  
 @Override  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 String role = (String) request.getSession().getAttribute("role");  
 if (role != null) {  
 Location location = new Location(request.getParameter("location"));  
 WeatherDate date1 = new WeatherDate(request.getParameter("date1"));  
 WeatherDate date2 = new WeatherDate(request.getParameter("date2"));  
 WeatherDAO weatherDAO = new WeatherDAO();  
 request.getSession().setAttribute("weatherList", weatherDAO.weatherList(location, date1, date2));  
 request.getRequestDispatcher("index.jsp").forward(request, response);  
 } else {  
 request.getSession().setAttribute("status", "BAD LOGIN");  
 request.getRequestDispatcher("error.jsp").forward(request, response);  
 }  
  
 }  
}

package Servlets;  
  
import Entity.Location;  
import Entity.WeatherDate;  
import db.WeatherDAO;  
  
import javax.servlet.\*;  
import javax.servlet.http.\*;  
import javax.servlet.annotation.\*;  
import java.io.IOException;  
  
@WebServlet(name = "weatherOneDay", value = "/weatherOneDay")  
public class weatherOneDay extends HttpServlet {  
 @Override  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
  
 }  
  
 @Override  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 String role = (String) request.getSession().getAttribute("role");  
 if (role != null) {  
 Location location = new Location(request.getParameter("location"));  
 WeatherDate date = new WeatherDate(request.getParameter("date"));  
 WeatherDAO weatherDAO = new WeatherDAO();  
 request.getSession().setAttribute("weatherOneDay", weatherDAO.weatherOneDay(location, date));  
 request.getRequestDispatcher("index.jsp").forward(request, response);  
 } else {  
 request.getSession().setAttribute("status", "BAD LOGIN");  
 request.getRequestDispatcher("error.jsp").forward(request, response);  
 }  
 }  
}

<%@ **taglib** prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>  
  
<%@ **page** contentType="text/html;charset=UTF-8" language="java" %>  
<html>  
<head>  
 <title>Admin Panel</title>  
 <link rel="stylesheet" href="stylesheet.css">  
</head>  
<body>  
<**c:choose**>  
 <**c:when** test="${role=='admin'}">  
 <a class="button" href="index.jsp">Home page</a>  
 <div class="form-container">  
 <form action="/addWeather" method="post">  
 <label>Location</label><br>  
 <input type="text" required name="location" pattern="^[A-Z][a-z- ]{1,}"><br>  
 <label>Date</label><br>  
 <input type="date" required name="date" pattern="\d{4}-\d{2}-\d{2}"><br>  
 <label>Weather</label><br>  
 <input type="text" required name="weather"><br>  
 <button type="submit">Add</button>  
 </form>  
 </div>  
 </**c:when**>  
 <**c:otherwise**>  
 <**jsp:include** page="error.jsp"></**jsp:include**>  
 </**c:otherwise**>  
</**c:choose**>  
</body>  
</html>

<%@ **taglib** prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>  
<%@ **page** contentType="text/html;charset=UTF-8" language="java" %>  
<html>  
<head>  
 <title>Error</title>  
</head>  
<body>  
<**c:choose**>  
 <**c:when** test="${status!=null}">  
 <p>${status}</p>  
 </**c:when**>  
 <**c:otherwise**>  
 <p>Smth wrong</p>  
 </**c:otherwise**>  
 </**c:choose**>  
${status=null}  
<a href="index.jsp">Click here to go Home Page</a>  
</body>  
</html>

<%@ **taglib** prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>  
<%@ **page** contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>  
<!DOCTYPE html>  
<html>  
<head>  
 <title>Sinoptik</title>  
 <link rel="stylesheet" href="stylesheet.css">  
</head>  
<body>  
<**c:choose**>  
 <**c:when** test="${logged==true}">  
 <script>  
 function showMore() {  
 document.getElementById('oneDay').parentElement.classList.add('close');  
 document.getElementById('list').parentElement.classList.remove('close');  
 }  
  
 function showLess() {  
 document.getElementById('list').parentElement.classList.add('close');  
 document.getElementById('oneDay').parentElement.classList.remove('close');  
 }  
 </script>  
 <div class="allForms">  
 <**c:if** test="${role == 'admin'}">  
 <a href="adminPanel.jsp" class="button">Admin panel</a>  
 </**c:if**>  
 <a href="/SignOut" class="button">SignOut</a>  
 <div class="form-container">  
 <form id="oneDay" method="post" action="/weatherOneDay">  
 <label>Enter Location</label><br>  
 <input type="text" required name="location"><br>  
 <label>Enter Date</label><br>  
 <input type="date" name="date" required pattern="\d{4}-\d{2}-\d{2}"><br>  
 <button type="submit">Submit</button>  
 </form>  
 <button onclick="showMore()">Search on list</button>  
 </div>  
 <div class="form-container close">  
 <form id="list" method="post" action="/weatherList">  
 <label>Enter Location</label><br>  
 <input type="text" required name="location"><br>  
 <label>Enter Date Between</label><br>  
 <input type="date" name="date1" required pattern="\d{4}-\d{2}-\d{2}"><br>  
 <input type="date" name="date2" required pattern="\d{4}-\d{2}-\d{2}"><br>  
 <button type="submit">Submit</button>  
 <br>  
 </form>  
 <button onclick="showLess()">Search one day</button>  
 </div>  
 </**c:when**>  
 <**c:otherwise**>  
 <div class="form-container">  
 <form action="/Login" method="post">  
 <label>Enter Login</label><br>  
 <input name="login" required type="text"><br>  
 <label>Enter password</label><br>  
 <input name="password" required type="password"><br>  
 <button type="submit">Login</button>  
 <br>  
 </form>  
 </div>  
 </**c:otherwise**>  
</**c:choose**>  
 </div>  
<**jsp:include** page="result.jsp"></**jsp:include**>  
</body>  
</html>

<%@ **taglib** prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>  
<**c:if** test="${weatherOneDay!=null}"><div class="weather">  
 <p>${weatherOneDay.location.locationName} </p>  
 <p>${weatherOneDay.weatherDate.date} </p>  
 <p>${weatherOneDay.weather} </p>  
</div>  
 ${weatherOneDay=null}  
  
</**c:if**>  
<**c:if** test="${weatherList!=null}"><div class="weatherAll">  
 <**c:forEach** items="${weatherList}" var="list">  
 <div class="weather">  
 <p>${list.location.locationName} </p>  
 <p>${list.weatherDate.date} </p>  
 <p>${list.weather} </p>  
 </div>  
 </**c:forEach**>  
</div>  
 ${weatherList=null}  
</**c:if**>

div.form-container.close{  
 display: none;  
}  
div.form-container {  
 display: block;  
 width: 100%;  
 text-align: -webkit-center;  
 float: left;  
}  
a.button{  
 border-radius: 25px;  
 border: 4px solid #dcdcdc;  
 display: block;  
 cursor: pointer;  
 color: #666666;  
 padding: 1% 2%;  
 text-decoration: none;  
 background-color: wheat;  
 width: 10%;  
 float: left;  
}  
body{  
 background-color: seashell;  
}  
form{  
 padding: 3%;  
 border-radius: 25px;  
 border: 4px solid #dcdcdc;  
 width: 20%;  
}  
div.allForms{  
 display: block;  
 width: 100%;  
 float: left;  
}  
div.weatherAll{  
 position: relative;  
 display: inline-flex;  
 flex-direction: row;  
 padding-top: 5%;  
 overflow: auto;  
 width: 100%;  
}  
div.weather{  
 display: block;  
 float: left;  
 width: 87.5%;  
 align-self: center;  
 text-align: center;  
 border: 4px solid;  
 padding-left: 6%;  
 padding-right: 6%;  
 border-color: darkgray;  
}