# SPRAWOZDANIE PLATFORMY PROGRAMISTYCZNE .NET I JAVA – LAB6


POLITECHNIKA WROCŁAWSKA
WYDZIAŁ INFORMATYKI I TELEKOMUNIKACJI
INFORMATYCZNE SYSTEMY AUTOMATYKI

MICHAŁ WYRZYKOWSKI

INDEKS 264228

# 1. Opis ogólny projektu aplikacji

### 1.1. Opis aplikacji

WeatherApp jest aplikacją desktopową napisaną w języku Java, wykorzystującą framework Swing do interfejsu graficznego oraz JPA (Java Persistence API) do zarządzania bazą danych. Aplikacja ta pozwala użytkownikom na pobieranie i wyświetlanie danych pogodowych z API OpenWeatherMap na podstawie podanych współrzędnych geograficznych (szerokości i długości geograficznej). Ponadto, dane pogodowe są przechowywane w bazie danych, co umożliwia ich późniejsze przeglądanie i filtrowanie według daty oraz nazwy miasta.

### 1.2. Funkcjonalności aplikacji

Pobieranie danych pogodowych:

- Użytkownik wprowadza szerokość i długość geograficzną, a następnie naciska przycisk „Fetch Weather".
- Aplikacja łączy się z API OpenWeatherMap, pobiera dane pogodowe i zapisuje je w bazie danych.

Wyświetlanie wszystkich danych pogodowych:

- Użytkownik może wyświetlić wszystkie zapisane dane pogodowe naciskając przycisk „Show All Weather Data".
- Dane są prezentowane w oknie tekstowym.

Filtrowanie danych pogodowych:

- Użytkownik może wprowadzić zakres dat oraz nazwę miasta, aby filtrować zapisane dane pogodowe.
- Wyniki są wyświetlane w oknie tekstowym po naciśnięciu przycisku „Filter Weather Data".

### 1.3. Podsumowanie

Aplikacja WeatherApp dostarcza podstawowe funkcje do pobierania, zapisywania i wyświetlania danych pogodowych z API OpenWeatherMap. Użytkownik może również filtrować dane według daty i nazwy miasta. Interfejs użytkownika jest prosty i intuicyjny, umożliwiając łatwą obsługę aplikacji. Całość jest napisana w języku Java, korzystając z technik i narzędzi takich jak Swing, JPA oraz Gson do parsowania JSON.

# 2. Opis klasy WeatherApp

### 2.1. Struktura klasy

Klasa WeatherApp jest główną klasą aplikacji, która dziedziczy po JFrame (klasa Swingowa odpowiedzialna za tworzenie okien w aplikacjach desktopowych). Jej zadaniem jest zarządzanie interfejsem użytkownika oraz interakcjami z bazą danych i zewnętrznym API pogodowym.

### 2.2. Pola klasy

- textArea, latitudeField, longitudeField, startDateField, endDateField, cityField: Pola tekstowe i obszar tekstowy używane do interakcji z

użytkownikiem.
- emf: Obiekt EntityManagerFactory używany do zarządzania połączeniami z bazą danych.

### 2.3. Konstruktor

Konstruktor klasy inicjalizuje EntityManagerFactory oraz wywołuje metodę initUI(), która odpowiada za ustawienie interfejsu użytkownika.

### 2.4. Metoda initUI()

Metoda ta konfiguruje wygląd i układ elementów interfejsu użytkownika. Wykorzystuje GridBagLayout do ustawienia komponentów w siatce:

- Panel tekstowy: textArea jest umieszczony w JScrollPane i dodany do panelu, aby wyświetlać wyniki operacji.
- Pola tekstowe: Pola tekstowe dla szerokości i długości geograficznej, daty początkowej i końcowej oraz miasta są dodane wraz z odpowiednimi etykietami.
- Przyciski: Trzy przyciski są dodane do panelu:
- Fetch Weather: Pobiera dane pogodowe z API.
- Show All Weather Data: Wyświetla wszystkie dane pogodowe zapisane w bazie danych.
- Filter Weather Data: Filtruje i wyświetla dane pogodowe na podstawie wprowadzonych kryteriów.

### 2.5. Metoda fetchWeatherData()

Metoda ta odpowiada za pobieranie danych pogodowych z API OpenWeatherMap:

- Pobiera współrzędne geograficzne od użytkownika.
- Tworzy URL do API na podstawie podanych współrzędnych.
- Otwiera połączenie HTTP i pobiera dane w formacie JSON.
- Mapuje dane JSON na obiekt WeatherResponse za pomocą biblioteki Gson.
- Tworzy obiekt WeatherBaseInfo i wypełnia go danymi z odpowiedzi API.
- Zapisuje dane do bazy danych za pomocą EntityManager.
- Wywołuje metodę updateWeatherInfo() do wyświetlenia najnowszych danych pogodowych.

### 2.6. Metoda displayAllWeatherData()

Metoda ta pobiera wszystkie zapisane dane pogodowe z bazy danych i wyświetla je w textArea. Używa EntityManager do wykonania zapytania SELECT w FROM WeatherBaseInfo w.

### 2.7. Metoda displayFilteredWeatherData()

Metoda ta pozwala na filtrowanie danych pogodowych według daty i nazwy miasta:

- Sprawdza, czy pola daty i nazwy miasta są wypełnione.
- Tworzy zapytanie SQL z odpowiednimi filtrami.
- Wykonuje zapytanie i wyświetla wyniki w textArea.

### 2.8. Metoda parseDate()

Metoda pomocnicza, która parsuje ciąg znaków reprezentujący datę w formacie dd-MM-yyyy do obiektu LocalDateTime.

### 2.9. Metoda updateWeatherInfo()

Metoda ta wyświetla najnowsze dane pogodowe dla podanych współrzędnych geograficznych:

- Wykonuje zapytanie do bazy danych, aby pobrać dane pogodowe dla podanych współrzędnych.
- Wyświetla dane w textArea.

### 2.10 Metoda main()

Metoda main() jest punktem wejścia aplikacji. Wywołuje SwingUtilities.invokeLater(), aby utworzyć i wyświetlić instancję WeatherApp w wątku EDT (Event Dispatch Thread), co jest standardową praktyką dla aplikacji Swing.

Podsumowując, klasa WeatherApp zarządza interfejsem użytkownika oraz logiką aplikacji, w tym pobieraniem danych pogodowych z zewnętrznego API, zapisywaniem ich w bazie danych oraz wyświetlaniem tych danych użytkownikowi. Wszystko to jest realizowane z użyciem komponentów Swing oraz JPA.

# 3. Opis klasy WeatherResponse

### 3.1. Struktura klasy

WeatherResponse jest klasą, która reprezentuje odpowiedź JSON z API OpenWeatherMap. Klasa ta zawiera pola i klasy wewnętrzne, które odpowiadają strukturze danych zwracanych przez API. Jest to standardowy sposób mapowania danych JSON na obiekty Javy, co ułatwia pracę z danymi w aplikacji.

### 3.2 Pola klasy WeatherResponse

- coord: Pole typu Coord, które przechowuje współrzędne geograficzne (szerokość i długość).
- weather: Lista obiektów typu Weather, które zawierają informacje o stanie pogody.
- main: Pole typu Main, które przechowuje główne informacje o pogodzie, takie jak temperatura, ciśnienie, wilgotność.
- wind: Pole typu Wind, które zawiera informacje o wietrze.
- clouds: Pole typu Clouds, które zawiera informacje o zachmurzeniu.
- sys: Pole typu Sys, które zawiera dodatkowe informacje systemowe, takie jak kraj, czas wschodu i zachodu słońca.
- visibility: Widoczność w metrach.
- timezone: Przesunięcie strefy czasowej w sekundach od UTC.
- dt: Znacznik czasu dla danych pogodowych.
- id: Identyfikator miasta.
- name: Nazwa miasta.

- cod: Kod odpowiedzi API.

### 3.3. Klasy wewnętrzne

- Coord
  - lon: Długość geograficzna.
  - lat: Szerokość geograficzna.
- Weather
  - id: Identyfikator stanu pogody.
  - main: Główna kategoria pogody (np. deszcz, słońce).
  - description: Opis stanu pogody.
  - icon: Kod ikony reprezentującej stan pogody.
- Main
  - temp: Temperatura.
  - feels_like: Temperatura odczuwalna.
  - temp_min: Minimalna temperatura.
  - temp_max: Maksymalna temperatura.
  - pressure: Ciśnienie atmosferyczne.
  - humidity: Wilgotność.
- Wind
  - speed: Prędkość wiatru.
  - deg: Kierunek wiatru w stopniach.
  - gust: Porywy wiatru.
- Clouds
  - all: Procentowe zachmurzenie.
- Sys
  - type: Typ systemu.
  - id: Identyfikator systemu.
  - country: Kraj.
  - sunrise: Czas wschodu słońca.
  - sunset: Czas zachodu słońca.

### 3.4. Zastosowanie

Klasa WeatherResponse jest używana do mapowania odpowiedzi z API OpenWeatherMap na obiekt Javy. Dzięki temu dane pogodowe mogą być łatwo przetwarzane w aplikacji. Klasa ta jest wykorzystywana w metodzie fetchWeatherData() klasy WeatherApp do parsowania odpowiedzi JSON.

### 3.5. Podsumowanie

Klasa WeatherResponse służy do mapowania odpowiedzi z API OpenWeatherMap na strukturę obiektową w Javie. Zawiera pola reprezentujące różne elementy danych pogodowych oraz klasy wewnętrzne, które dokładnie odwzorowują strukturę danych zwracanych przez API. Dzięki temu możliwe jest łatwe przetwarzanie i wyświetlanie danych pogodowych w aplikacji.

# 4. Opis klasy WeatherBaseInfo

### 4.1. Struktura klasy

WeatherBaseInfo jest klasą reprezentującą encję JPA (Java Persistence API), która mapuje dane pogodowe na rekordy w tabeli bazy danych. Klasa ta przechowuje informacje o pogodzie, które są pobierane z API OpenWeatherMap i zapisywane w bazie danych.

### 4.2. Anotacje JPA

- @Entity: Oznacza, że klasa jest encją JPA i będzie mapowana na tabelę w bazie danych.
- @Table(name = "weather_data"): Określa nazwę tabeli w bazie danych, do której będzie mapowana ta encja.

### 4.3. Pola klasy

- @Id: Oznacza, że pole id jest kluczem głównym.
- @GeneratedValue(strategy = GenerationType.IDENTITY): Określa, że wartość id będzie automatycznie generowana przez bazę danych przy użyciu strategii AUTO_INCREMENT.
- private int id: Unikalny identyfikator rekordu.
- private String cityName: Nazwa miasta.
- private String country: Nazwa kraju.
- private float latitude: Szerokość geograficzna.
- private float longitude: Długość geograficzna.
- private int timezone: Przesunięcie strefy czasowej w sekundach od UTC.
- private float temperature: Temperatura.
- private int pressure: Ciśnienie atmosferyczne.
- private int humidity: Wilgotność.
- private float windSpeed: Prędkość wiatru.
- private int windDirection: Kierunek wiatru w stopniach.
- private String weatherCondition: Główna kategoria pogody (np. deszcz, słońce).
- private LocalDateTime dateAdded: Data i czas dodania rekordu do bazy danych.

### 4.4. Gettery i setery

Każde pole posiada odpowiednie metody getter i setter, które umożliwiają odczyt i modyfikację wartości tych pól. Są one standardowymi elementami każdej encji JPA, umożliwiającymi dostęp do danych w sposób bezpieczny i kontrolowany.

### 4.5. Zastosowanie

Klasa WeatherBaseInfo jest używana w aplikacji WeatherApp do:

- Zapisywania danych pogodowych: W metodzie fetchWeatherData(), dane pobrane z API są mapowane na obiekt WeatherBaseInfo i zapisywane w bazie danych.
- Pobierania danych pogodowych: W metodach displayAllWeatherData() i displayFilteredWeatherData(), dane są pobierane z bazy danych i wyświetlane w interfejsie użytkownika

### 4.6. Podsumowanie

Klasa WeatherBaseInfo jest kluczowym elementem w aplikacji WeatherApp, umożliwiającym mapowanie danych pogodowych na rekordy w bazie danych. Dzięki niej możliwe jest przechowywanie, pobieranie i filtrowanie danych pogodowych, co stanowi podstawę funkcjonalności aplikacji. Anotacje JPA i odpowiednie metody getter i setter zapewniają, że dane są zarządzane w sposób efektywny i zgodny z najlepszymi praktykami w zakresie ORM (Object-Relational Mapping).