**Politechnika Krakowska, projekt zespołowy grupa 3C**

Kamil Naglik
Wojciech Pajtel
Maciej Rybiński
Łukasz Niedzielski
Tomasz Rolek

Witamy w dokumentacji aplikacji PogoApp!
Dokumentacja ma na celu zaprezentowanie aplikacji pogodowej o nazwie PogoApp, która zapewnia łatwy dostęp do aktualnych informacji pogodowych dla różnych miast w Polsce. PogoApp została stworzona, aby dostarczyć użytkownikom nie tylko podstawowe dane pogodowe, ale również użyteczne informacje, takie jak prognoza na weekend oraz jakość powietrza w większych miastach.

Aplikacja PogoApp oferuje prosty i intuicyjny interfejs, który umożliwia szybkie wyszukiwanie miast i wyświetlanie bieżących danych pogodowych.

Główne funkcje aplikacji obejmują:

- Wyszukiwanie miast: Użytkownicy mogą wprowadzić nazwę miasta, aby uzyskać bieżące informacje pogodowe dla danej lokalizacji. Aplikacja oferuje szeroki zakres miast w Polsce, dzięki czemu każdy może znaleźć interesujące go miejsce.

- Prognoza na weekend: Oprócz aktualnych danych pogodowych, aplikacja dostarcza także prognozy na nadchodzący weekend. Użytkownicy będą mogli dowiedzieć się, jakie warunki atmosferyczne przewiduje się na najbliższe dni, aby dobrze zaplanować swoje aktywności na wolnym powietrzu.

- Ikony pogodowe: Aplikacja wykorzystuje ilustracje, które odzwierciedlają różne warunki atmosferyczne, takie jak słońce, chmury, deszcz czy śnieg. Dzięki temu użytkownicy mogą intuicyjnie zobaczyć, jaka jest obecna pogoda na podstawie odpowiedniej ikony.

- Jakość powietrza: Aplikacja dostarcza również informacje o jakości powietrza w większych miastach w Polsce. Użytkownicy będą mieli możliwość śledzenia poziomu zanieczyszczeń atmosferycznych, co jest szczególnie istotne dla osób zainteresowanych zdrowym trybem życia i środowiskiem naturalnym.

Niniejsza dokumentacja zawiera szczegółowe informacje dotyczące instalacji, konfiguracji, interfejsu użytkownika, funkcji aplikacji, architektury systemu, zarządzania danymi, testowania, obsługi oraz planów rozwoju. Przeznaczona jest dla użytkowników i deweloperów, aby zapewnić im pełne zrozumienie i wykorzystanie potencjału naszej aplikacji pogodowej.

Dokumentacja techniczna aplikacji

Wstęp:
Dokumentacja techniczna aplikacji PogoApp zawiera szczegółowe informacje dotyczące implementacji i funkcjonalności tej aplikacji. Aplikacja została stworzona w celu dostarczenia użytkownikom aktualnych danych pogodowych dla wybranych miejscowości. Dokumentacja techniczna obejmuje zarówno aspekty backendu, jak i frontendu, wraz z użytymi technologiami i bibliotekami.

**Backend**:
Aplikacja PogoApp wykorzystuje zaawansowany stack technologiczny w celu zapewnienia niezawodnej i efektywnej obsługi żądań oraz przetwarzania danych.
Użyte technologie:

- **Java 17**
- **Spring Boot 3.0.5**
- **Maven**
- **Servlety**
- **Spring Web MVC**
- **Apache Tomcat**
- **Jakarta Servlet**
- **JSP**
- **Apache Commons Lang**
- **Jsoup**
- **Jackson**

**Frontend**:
Frontend aplikacji PogoApp został zaprojektowany w sposób responsywny i estetyczny, aby zapewnić użytkownikom przyjemne doświadczenie użytkowania.
Użyte technologie:

- **HTML5**
- **CSS3**
- **JavaScript**
- **jQuery**
- **Font Awesome**
- **Bootstrap**
- **Lightbox**

**PogoAppApplication**

Klasa **PogoAppApplication** jest punktem startowym (Spring Boot) dla aplikacji PogoApp.

Metody:

- **main(String[] args)**: Metoda główna uruchamiająca aplikację. Wywołuje SpringApplication.run() do uruchomienia aplikacji Spring Boot.

Uwagi:

- Klasa jest oznaczona adnotacją @SpringBootApplication, która wskazuje, że jest to klasa konfiguracyjna Spring Boot.
- Wywołanie SpringApplication.run() inicjalizuje kontekst aplikacji Spring Boot i uruchamia aplikację.
- Wykorzystuje klasę WeatherService do pobierania danych pogodowych dla określonej lokalizacji.

**WeatherControler**

Klasa **WeatherControler** jest kontrolerem obsługującym żądania HTTP związane z funkcjonalnościami dotyczącymi pogody w aplikacji PogoApp.

Metody:

- **mappingOnWelcome(Model model, @CookieValue(defaultValue = "Kraków") String preferred_location)**: Obsługuje żądanie dostępu do strony głównej.

- **peaks(Model model)**: Obsługuje żądanie dostępu do strony z informacjami o szczytach.

- **weekendWeatherEntry(Model model)**: Obsługuje żądanie dostępu do strony z informacjami o pogodzie na weekend.

- **authors(Model model)**: Obsługuje żądanie dostępu do strony z informacjami o autorach aplikacji.

- **smogkrakow(Model model, @RequestParam(defaultValue = "Kraków") String name)**: Obsługuje żądanie dostępu do strony z informacjami o jakości powietrza w Krakowie lub innym podanym mieście.

- **mappingOnCity(Model model, String name)**: Obsługuje żądanie dostępu do strony z informacjami o pogodzie dla konkretnego miasta.

Klasa **WeatherController** jest oznaczona adnotacją **@Controller** i jest odpowiedzialna za przetwarzanie żądań związanych z pogodą w aplikacji PogoApp.

**City**

Klasa **City** jest modelem przechowującym informacje o aktualnie wyszukiwanym mieście.

Pola:
- **name**: Reprezentuje nazwę miasta.

Metody:
- **getName()**: Zwraca nazwę miasta.
- **setName(String name)**: Ustawia nazwę miasta.

Klasa **City** jest odpowiedzialna za przechowywanie informacji o aktualnie wyszukiwanym mieście w aplikacji PogoApp.

**Coordinates**

Klasa **Coordinates** reprezentuje współrzędne geograficzne oraz nazwę dla określonego miejsca.

Pola:
- **latitude**: Reprezentuje szerokość geograficzną.
- **longitude**: Reprezentuje długość geograficzną.
- **displayName**: Reprezentuje nazwę miejsca.

Konstruktory:
- **Coordinates(String latitude, String longitude, String displayName)**: Inicjalizuje obiekt **Coordinates** z podanymi wartościami szerokości geograficznej, długości geograficznej i nazwy miejsca.
- **Coordinates()**: Konstruktor domyślny.

Metody:
- **getLatitude()**: Zwraca szerokość geograficzną.
- **setLatitude(String latitude)**: Ustawia szerokość geograficzną.
- **getLongitude()**: Zwraca długość geograficzną.
- **setLongitude(String longitude)**: Ustawia długość geograficzną.
- **getDisplayName()**: Zwraca nazwę miejsca.
- **setDisplayName(String displayName)**: Ustawia nazwę miejsca.
- **toString()**: Zwraca tekstową reprezentację obiektu Coordinates.

Klasa **Coordinates** jest wykorzystywana w aplikacji PogoApp do przechowywania współrzędnych geograficznych i nazwy dla określonego miejsca.

**Smog**

Klasa **Smog** reprezentuje informacje o jakości powietrza dla konkretnego miejsca.

Pola:
- **stationName**: Nazwa stacji pomiarowej.
- **id**: Identyfikator.
- **stCalcDate**: Data pomiaru.
- **stIndexLevel**: Poziom jakości powietrza ogólny.
- **so2IndexLevel**: Poziom dwutlenku siarki (SO2).
- **no2IndexLevel**: Poziom dwutlenku azotu (NO2).
- **pm10IndexLevel**: Poziom pyłów zawieszonych PM10.
- **pm25IndexLevel**: Poziom pyłów zawieszonych PM2.5.
- **o3IndexLevel**: Poziom ozonu (O3).
- **stIndexStatus**: Status wskaźnika jakości powietrza.

Metody:
- **getStationName()**: Zwraca nazwę stacji pomiarowej.
- **getStCalcDate()**: Zwraca datę pomiaru.
- **getStIndexLevel()**: Zwraca poziom jakości powietrza ogólny.
- **getSo2IndexLevel()**: Zwraca poziom dwutlenku siarki (SO2).
- **getNo2IndexLevel()**: Zwraca poziom dwutlenku azotu (NO2).
- **getPm10IndexLevel()**: Zwraca poziom pyłów zawieszonych PM10.
- **getPm25IndexLevel()**: Zwraca poziom pyłów zawieszonych PM2.5.
- **getO3IndexLevel()**: Zwraca poziom ozonu (O3).
- **isStIndexStatus()**: Sprawdza status wskaźnika jakości powietrza.
- **toString()**: Zwraca tekstową reprezentację obiektu **Smog**.

Klasa **Smog** jest wykorzystywana w aplikacji PogoApp do przechowywania informacji o jakości powietrza dla konkretnego miejsca.

**Station**

Klasa **Station** reprezentuje informacje o stacji pomiarowej.

Pola:
- **id**: Identyfikator stacji.
- **stationName**: Opcjonalna nazwa stacji.
- **gegrLat**: Współrzędna geograficzna szerokości.
- **gegrLon**: Współrzędna geograficzna długości.
- **addressStreet**: Opcjonalny adres ulicy stacji.

Metody:
- Brak dodatkowych metod.

Klasa **Station** jest wykorzystywana w aplikacji PogoApp do przechowywania informacji o stacjach pomiarowych.

**Weather**

Klasa **Weather** reprezentuje dane pogodowe. Przechowuje informacje dotyczące pogody na podstawie lokalizacji geograficznej.

Pola

- **coords**: obiekt klasy Coordinates zawierający współrzędne geograficzne
- **name**: nazwa lokalizacji
- **sunriseDaily**: lista zawierająca godziny wschodu słońca dla każdego dnia
- **sunsetDaily**: lista zawierająca godziny zachodu słońca dla każdego dnia
- **dateDaily**: lista zawierająca daty dla każdego dnia
- **temperatureDaily**: lista zawierająca temperatury dla każdego dnia
- **pressureDaily**: lista zawierająca ciśnienie atmosferyczne dla każdego dnia
- **windDaily**: lista zawierająca prędkość wiatru dla każdego dnia
- **precipitationProbabilityDaily**: lista zawierająca prawdopodobieństwo opadów dla każdego dnia
- **timeHourly**: lista zawierająca godziny dla danych pogodowych na przestrzeni jednego dnia
- **temperatureHourly**: lista zawierająca temperatury dla danych pogodowych na przestrzeni jednego dnia
- **windHourly**: lista zawierająca prędkość wiatru dla danych pogodowych na przestrzeni jednego dnia
- **precipitationHourly**: lista zawierająca opady atmosferyczne dla danych pogodowych na przestrzeni jednego dnia
- **cloudCoverHourly**: lista zawierająca pokrycie chmurami dla danych pogodowych na przestrzeni jednego dnia

Metody:

- **Weather(coords)**: Tworzy nowy obiekt Weather na podstawie obiektu klasy Coordinates.
- **getCoords()**: Zwraca obiekt klasy Coordinates zawierający współrzędne geograficzne.
- **setCoords(coords)**: Ustawia nowe współrzędne geograficzne dla obiektu Weather.
- **getName()**: Zwraca nazwę lokalizacji.
- **setName(name)**: Ustawia nową nazwę lokalizacji.
- **getDateDaily()**: Zwraca listę zawierającą daty dla każdego dnia.
- **setDateDaily(dateDaily)**: Ustawia nową listę dat dla każdego dnia.
- **getTemperatureDaily()**: Zwraca listę zawierającą temperatury dla każdego dnia.

- **setTemperatureDaily(temperatureDaily)**: Ustawia nową listę temperatur dla każdego dnia.
- **getPressureDaily()**: Zwraca listę zawierającą ciśnienie atmosferyczne dla każdego dnia.
- **setPressureDaily(pressureDaily)**: Ustawia nową listę ciśnień atmosferycznych dla każdego dnia.
- **getWindDaily()**: Zwraca listę zawierającą prędkość wiatru dla każdego dnia.
- **setWindDaily(windDaily)**: Ustawia nową listę prędkości wiatru dla każdego dnia.
- **getTimeHourly():** Zwraca listę zawierającą godziny dla danych pogodowych na przestrzeni jednego dnia.
- **setTimeHourly(timeHourly)**: Ustawia nową listę godzin dla danych pogodowych na przestrzeni jednego dnia.
- **getTemperatureHourly():** Zwraca listę zawierającą temperatury dla danych pogodowych na przestrzeni jednego dnia.
- **setTemperatureHourly(temperatureHourly)**: Ustawia nową listę temperatur dla danych pogodowych na przestrzeni jednego dnia.
- **getWindHourly()**: Zwraca listę zawierającą prędkość wiatru dla danych pogodowych na przestrzeni jednego dnia.
- **setWindHourly(windHourly)**: Ustawia nową listę prędkości wiatru dla danych pogodowych na przestrzeni jednego dnia.
- **getPrecipitationHourly()**: Zwraca listę zawierającą opady atmosferyczne dla danych pogodowych na przestrzeni jednego dnia.
- **setPrecipitationHourly(precipitationHourly)**: Ustawia nową listę opadów atmosferycznych dla danych pogodowych na przestrzeni jednego dnia.
- **getCloudCoverHourly()**: Zwraca listę zawierającą pokrycie chmurami dla danych pogodowych na przestrzeni jednego dnia.
- **setCloudCoverHourly(cloudCoverHourly)**: Ustawia nową listę pokrycia chmurami dla danych pogodowych na przestrzeni jednego dnia.
- **getSunriseDaily()**: Zwraca listę zawierającą godziny wschodu słońca dla każdego dnia.
- **setSunriseDaily(sunriseDaily)**: Ustawia nową listę godzin wschodu słońca dla każdego dnia.
- **getSunsetDaily()**: Zwraca listę zawierającą godziny zachodu słońca dla każdego dnia.
- **setSunsetDaily(sunsetDaily)**: Ustawia nową listę godzin zachodu słońca dla każdego dnia.
- **getPrecipitationProbabilityDaily()**: Zwraca listę zawierającą prawdopodobieństwo opadów dla każdego dnia.
- **setPrecipitationProbabilityDaily(precipitationProbabilityDaily)**: Ustawia nową listę prawdopodobieństwa opadów dla każdego dnia.
- **toString()**: Zwraca tekstową reprezentację obiektu Weather.

**GPSCoordinates**

Klasa **GPSCoordinates** jest odpowiedzialna za pobieranie współrzędnych geograficznych na podstawie lokalizacji.

Pola:

- Brak pól w tej klasie.

Metody:

- **getCoordinatesForLocation(location)**: Metoda statyczna, która przyjmuje jako argument lokalizację w postaci ciągu znaków. Metoda wykonuje zapytanie do serwisu Nominatim OpenStreetMap w celu uzyskania danych geograficznych dla podanej lokalizacji. Na podstawie odpowiedzi JSON, metoda tworzy nowy obiekt klasy Coordinates i ustawia wartości współrzędnych geograficznych (szerokość i długość) oraz nazwę lokalizacji. Jeśli wystąpi błąd lub nie uda się znaleźć danych dla podanej lokalizacji, metoda zwraca obiekt Coordinates z wartościami domyślnymi oznaczającymi brak danych.

Zależności:

- Klasa GPSCoordinates korzysta z biblioteki Jackson do przetwarzania JSON oraz z klasy Coordinates.

Uwagi:

- Metoda ***getCoordinatesForLocation*** wykorzystuje serwis Nominatim OpenStreetMap do pobierania danych geograficznych. Upewnij się, że masz dostęp do internetu i że serwis jest dostępny.
- W przypadku wystąpienia błędu lub braku danych, metoda zwraca obiekt Coordinates z wartościami domyślnymi oznaczającymi brak danych (np. nazwa lokalizacji "NOT FOUND").

**SmogService**

Klasa **SmogService** jest odpowiedzialna za obsługę danych dotyczących zanieczyszczenia powietrza (smog).

Pola:

- **stations**: Lista stacji pomiarowych, które są zbierane przez metodę collectStations(). Pole jest statyczne, dzięki czemu jest współdzielone między różnymi instancjami klasy SmogService.

Metody:

- **fromJSON(type, jsonPacket)**: Metoda generyczna, która przyjmuje jako argumenty referencję do typu oraz URL z danymi w formacie JSON. Metoda

korzysta z biblioteki Jackson do deserializacji danych JSON na obiekt o podanym typie.
- **collectStations()**: Metoda prywatna, która zbiera informacje o stacjach pomiarowych z serwisu API GIOŚ (Główny Inspektorat Ochrony Środowiska).
- **findStations(query)**: Metoda statyczna, która przyjmuje jako argument zapytanie wyszukiwania (ciąg znaków) i zwraca listę stacji pomiarowych pasujących do zadanego zapytania. Metoda korzysta z danych zebranych przez metodę collectStations(). Wyszukiwanie jest nieczułe na wielkość liter. Metoda zwraca maksymalnie 15 pasujących stacji.
- **getSmogData(station)**: Metoda prywatna, która przyjmuje jako argument obiekt stacji pomiarowej i zwraca dane dotyczące zanieczyszczenia powietrza (smog) dla tej stacji. Metoda korzysta z serwisu API GIOŚ, aby pobrać dane o zanieczyszczeniu powietrza dla konkretnej stacji.
- **fillmodelwithSmogData(model, name)**: Metoda statyczna, która przyjmuje jako argument obiekt Model (z biblioteki Spring) oraz nazwę stacji pomiarowej. Metoda wypełnia model danymi dotyczącymi zanieczyszczenia powietrza dla pasujących stacji pomiarowych i dodaje te dane do modelu. Metoda korzysta z metod findStations() i getSmogData().

Uwagi:
- Metoda ***fromJSON()*** korzysta z biblioteki Jackson do deserializacji danych JSON. Upewnij się, że masz zainstalowaną odpowiednią bibliotekę i masz dostęp do niej.
- Metoda ***collectStations()*** pobiera dane o stacjach pomiarowych z serwisu API GIOŚ. Upewnij się, że masz dostęp do internetu i że serwis jest dostępny.
- Metoda ***findStations()*** i ***getSmogData()*** korzystają z danych zebranych przez metodę ***collectStations()***. Upewnij się, że metoda collectStations() zostanie wywołana przed użyciem tych metod.
- Metoda ***fillmodelwithSmogData()*** wypełnia model danymi dotyczącymi zanieczyszczenia powietrza i dodaje te dane do modelu. Model jest obiektem dostarczonym przez bibliotekę Spring. Upewnij się, że masz odpowiednie zależności i konfigurację, aby korzystać z biblioteki Spring.

**WeatherService**

Klasa **WeatherService** zawiera funkcje do pobierania danych pogodowych dla określonej lokalizacji. Kluczowe pola i metody w tej klasie to:

Pola:
- **objectMapper**: Obiekt ObjectMapper do przetwarzania danych JSON.
- **coords**: Obiekt Coordinates przechowujący współrzędne geograficzne lokalizacji.
- **link**: Adres URL do zewnętrznego API pogodowego.

Metody:

- **getWeatherDataForLocation(String location)**: Pobiera dane pogodowe dla określonej lokalizacji. Zwraca obiekt Weather.
- **fillModelWithWeatherData(Model model, String loc)**: Wypełnia obiekt Model danymi pogodowymi dla określonej lokalizacji.

Metody pomocnicze:

- **averagePressures(List<Double> pressureList)**: Oblicza średnie ciśnienie atmosferyczne na podstawie listy wartości ciśnienia.
- **fillWeatherIconsHourly(Weather weather)**: Wypełnia listę ikon pogodowych na podstawie danych godzinowych.
- **fillWeatherIconsWeekly(Weather weather)**: Wypełnia listę ikon pogodowych na podstawie danych tygodniowych.

Uwagi:

- Klasa wykorzystuje zewnętrzne API pogodowe do pobierania danych.
- Obsługuje błędy związane z brakiem danych dla określonej lokalizacji.
- Korzysta z biblioteki Jackson do przetwarzania danych JSON.

**WeekDayNaming**

Klasa **WeekDayNaming** zawiera funkcję do ustawiania nazw dni tygodnia na modelu. Kluczowe elementy w tej klasie to:

Pola:

- **polish**: Obiekt Locale reprezentujący język polski.

Metody:

- **setWeekDayNames(Model model)**: Ustawia nazwy dni tygodnia na modelu, zaczynając od aktualnego dnia i następnych sześciu dni.

Uwagi:

- Klasa korzysta z obiektu LocalDateTime do uzyskania aktualnej daty i czasu.
- Wykorzystuje obiekt Locale do ustalenia nazw dni tygodnia w języku polskim.
- Dodaje nazwy dni tygodnia do modelu, używając atrybutu "dayOfWeek".