

Projekt z przedmiotu Programowanie aplikacyjne: aplikacja webowa do wyświetlania prognozy pogody i serwer REST dostarczający dane pogodowe

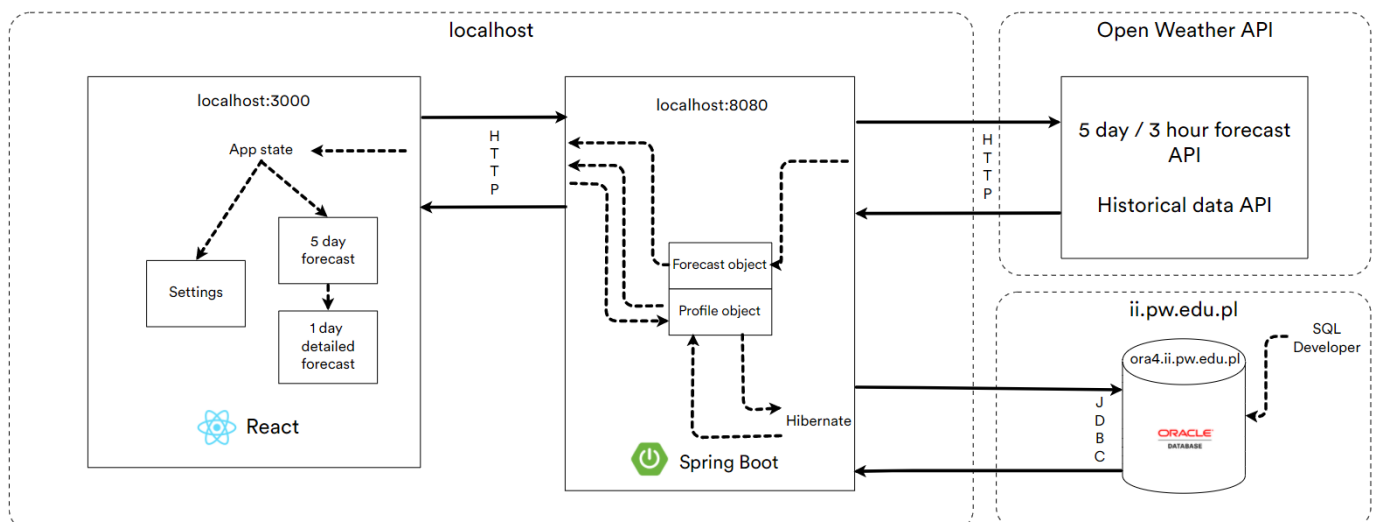
Bartłomiej Piktel, Jan Jeschke

Architektura projektu:

Zastosowanym wzorcem projektowym jest wzorzec Single Page Application. Aplikacja webowa składa się z pojedynczej strony internetowej, której zawartość jest dynamicznie generowana przez logikę napisaną w języku JavaScript z wykorzystaniem biblioteki React.js. Wszystkie elementy aplikacji wyświetlane są na tej samej stronie, a nawigacja jest zrealizowana poprzez mechanizmy logiki aplikacji, a nie poprzez przełączanie pomiędzy stronami.

Kluczowe warstwy projektu:

- warstwa persystencji danych:
 - baza relacyjna – Oracle dostępna na serwerze ii.pw.edu.pl
 - możliwość podglądu i zarządzania danymi przez SQLDeveloper
- warstwa logiki i dostępu do danych
 - serwer SpringBoot + Java
 - główna encja: prognoza pogody na 5 dni dla danego miasta (klasa Forecast)
 - dodatkowa encja: profil użytkownika przypisujący miasto do użytkownika (klasa Profile)
 - dodatkowe klasy odpowiedzialne za:
 - konfigurację kontrolerów do komunikacji zewnętrznej (klasy Controller);
 - zdefiniowanie repozytorium do przechowywania danych w bazie Oracle (klasy Repository);
 - parsowanie danych otrzymanych z zewnętrznego API (klasy Utility)
 - kontroler zrealizowany przez Spring Framework, dostęp do API OpenWeatherMap zrealizowany przy użyciu HttpURLConnection (Java)
 - dostęp do danych (repozytoria) zrealizowane przez Hibernate
- warstwa prezentacji
 - aplikacja webowa przy użyciu biblioteki React
 - prognoza pogody przechowywana w stanie aplikacji
 - komponenty bezstanowe WeatherCard (w wersji Small i Large) służące do wyświetlania danych pogodowych
 - komponent stanowy SettingsCard służący do zarządzania profilami i przechowujący w stanie listę profili



Wykorzystywane narzędzia:

Środowiskiem programistycznym wykorzystywanym do tworzenia projektu było Visual Studio Code z rozszerzeniami zapewniającymi obsługę języka Java oraz JavaScript. Do zarządzania zależnościami aplikacji serwera posłużył Maven, a do zależności aplikacji webowej wykorzystany został manager npm.

Do inicjalizacji projektu posłużyły narzędzia „Spring Initializr” (<https://start.spring.io/>) oraz „Create React App” (<https://create-react-app.dev/>).

Główne zależności aplikacji webowej:

- react – główna biblioteka do tworzenia interfejsów użytkownika
- react-apexcharts – biblioteka dostarczająca wykresy
- react-dropdown – biblioteka dostarczająca listę wyboru typu dropdown

Główne zależności serwera REST:

- org.springframework.boot – mechanizmy działania kontrolerów
- org.hibernate – metody dostępu do bazy danych
- com.oracle.database.jdbc – interfejsy wymagane do połączenia z bazą danych Oracle
- com.google.code.gson – parsowanie danych w formacie .json
- org.junit.jupiter – realizacja testów jednostkowych

Testy jednostkowe:

Testy jednostkowe pozwalają na sprawdzenie poprawności działania parsera. Do testów zostały wygenerowane pliki .json z przykładowymi danymi, które parsowane są w ramach testów i sprawdzana jest ich poprawność. Dodatkowo testy obejmują poprawność działania klasy Forecast (poprawność odczytywania danych z klasy, poprawne ilości danych na listach).

```
[INFO] Results:
[INFO]
[INFO] Tests run: 27, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 5.636 s
[INFO] Finished at: 2021-05-27T09:21:39+02:00
[INFO] -----
```


Raporty walidacji:

Raport checkstyle:

WeatherApp REST Server

Project Documentation

- Project Information
- Project Reports
 - Checkstyle



Checkstyle Results

The following document contains the results of Checkstyle 8.29 with checkstyle.xml ruleset.

[rss feed](#)

Summary

Files	Info	Warnings	Errors
21	0	0	0

Files

File	I	W	E
------	---	---	---

Rules

Category	Rule	Violations	Severity
----------	------	------------	----------

Details

Raporty walidacji plików CSS:

Wyniki Walidatora CSS W3C dla App.css (CSS wersja 3 + SVG)

Gratulacje! Nie znaleziono żadnych błędów.

Dokument ten jest poprawnie napisany arkuszem [CSS wersja 3 + SVG](#) !

Wyniki Walidatora CSS W3C dla SettingsCard.css (CSS wersja 3 + SVG)

Gratulacje! Nie znaleziono żadnych błędów.

Dokument ten jest poprawnie napisany arkuszem [CSS wersja 3 + SVG](#) !

Wyniki Walidatora CSS W3C dla WeatherCardLarge.css (CSS wersja 3 + SVG)

Gratulacje! Nie znaleziono żadnych błędów.

Dokument ten jest poprawnie napisany arkuszem [CSS wersja 3 + SVG](#) !

Wyniki Walidatora CSS W3C dla WeatherCardSmall.css (CSS wersja 3 + SVG)

Gratulacje! Nie znaleziono żadnych błędów.

Dokument ten jest poprawnie napisany arkuszem [CSS wersja 3 + SVG](#) !

Błędy znalezione przez linter eslint:

```
PS C:\PAP\pap211-z09\weatherapp> eslint ./src
PS C:\PAP\pap211-z09\weatherapp> |
```

Wykorzystane zewnętrzne API:

Do pobierania danych pogodowych zostało wykorzystane API firmy OpenWeather. Darmowa wersja pozwala na pobranie prognozy pogody na najbliższe 5 dni z danymi w odstępach co 3 godziny. Dane te muszą być uzupełnione o dane historyczne od początku dnia do pierwszej godziny prognozy w celu wyświetlenia pogody dla pełnego dnia.

Zmiany wprowadzone po etapie 3:

Jedyną zmianą wprowadzoną po etapie 3 jest zmiana struktury plików serwera REST. Zostały one podzielone na 4 grupy: Controller – kontrolery REST; Entity – encje programu (Forecast oraz Profile); Repository – interfejsy odpowiedzialne za kontakt z bazą danych; Utility – klasy pomocnicze, takie jak parsery danych i pomocnicze struktury danych używane podczas parsowania.

Analiza rozwiązania:

Celem projektu było stworzenie rozwiązania wykorzystującego różnorodne technologie. Z tego powodu zadanie wyświetlania prognozy pogody, które mogłoby zostać rozwiązane przy pomocy samej aplikacji webowej, zostało rozbite na aplikację webową oraz serwer REST. O ile takie rozwiązanie nie było konieczne ze strony technicznej, to pobieranie danych z serwera REST, a nie bezpośrednio z zewnętrznego API, rozwiązało problem ograniczonej liczby zapytań oferowanej przez darmowe API. Pośrednictwo serwera REST pozwala na optymalizację liczby zapytań, poprzez przechowywanie prognozy w bazie danych i korzystanie z nich dopóki nie pojawi się konieczność aktualizacji.

Sam moduł prognozy nie realizował wszystkich funkcji CRUD, dlatego konieczne było dodanie dodatkowej funkcjonalności, które wykorzystywałyby usuwanie rekordów z bazy danych. W tym celu powstały profile, które zastąpiły bezpośredni wybór miasta prognozy pogody. Profile pozwalają na wszystkie 4 operacje CRUD.

Jednym z podstawowych problemów projektu był problem prezentacji danych w formie 5 pełnych dni. Prognoza pogody pobierana z OpenWeather API zawiera dane na 5 dni, ale liczone od najbliższych pełnych 3 godzin. Z tego powodu konieczne było wykorzystanie dodatkowych historycznych danych pogodowych. Wprowadziło to konieczność stworzenia dodatkowego parsera, który pobiera dane historyczne dla obecnego dnia, a następnie dołącza dane w odstępach 3-godzinnych (dane historyczne dostępne są tylko w formie cogodzinnej) na początku pobranej prognozy. Oprócz tego zapytanie o dane prognozowe realizowane jest na podstawie miasta, a dane historyczne muszą być pobierane wykorzystując długość i szerokość geograficzną, która odczytywana jest z odpowiedzi na zapytanie o prognozę.

Innym problemem, wynikającym z rozbudowanej struktury, są stosunkowo długie czasy ładowania, zależne od szybkości odpowiedzi zewnętrznego API lub bazy danych. Zastosowanie profili zmniejszyło widoczność tego problemu, ponieważ dane są pobierane już w momencie wyboru profilu, a nie przejścia na ekran wyświetlania pogody, ale przy dużym obciążeniu serwerów wciąż są widoczne ekrany ładowania.

Największym ograniczeniem projektu jest to, że dane prognozowe, ze względu na ograniczenia darmowego API, są pobierane dla strefy czasowej UTC+0 i w takiej formie prezentowane, ponieważ nie ma możliwości dostosowania

danych do strefy czasowej UTC+2 (dane co 3 godziny). Problem ten mógłby być rozwiązany przez wykorzystanie płatnego API, pozwalającego na pobieranie danych pogodowych z danymi co godzinę.

Jedną z możliwości rozwinięcia aplikacji byłaby walidacja poprawności wybranego miasta na poziomie profilu. Wprowadza to jednak szereg problemów. Wykorzystywane API nie zapewnia zapytania o poprawność miasta, dlatego jedyną metodą weryfikacji jest wysłanie dowolnego zapytania i sprawdzenie kodu odpowiedzi. Dodanie kolejnego rodzaju zapytania prowadziłoby do szybszego wykorzystania puli zapytań darmowego API, a dodatkowo długi czas oczekiwania na odpowiedź sprawiłby, że taka walidacja zmniejszyłaby płynność obsługi aplikacji.