AI1 **LAB D**	Bancewicz Aleksandra Album ba34753 Grupa 1	**Wersja 1**

REST API CLIENT

SPIS TREŚCI

CEL ZAJĘĆ

Celem głównym zajęć jest zdobycie następujących umiejętności:

- pobieranie danych z zewnętrznych zasobów za pomocą REST API
- zdobywanie wiedzy na temat zewnętrznych API za pomocą dokumentacji typu Swagger
- wysyłanie asynchronicznych żądań z wykorzystaniem XMLHttpRequest i Fetch API

W praktycznym wymiarze uczestnicy stworzą dynamiczną stronę HTML pozwalającą na wyświetlanie bieżącej informacji pogodowej oraz prognoz dla zadanej przez użytkownika miejscowości.

ROZPOCZĘCIE

Rozpoczęcie zajęć. Powtórzenie wykonywania połączeń synchronicznych i asynchronicznych z poziomu JS na stornie.

Wejściówka?

UWAGA

Ten dokument aktywnie wykorzystuje niestandardowe właściwości. Podobnie jak w LAB A wejdź do `Plik` -> `Informacje` -> `Właściwości` -> `Właściwości zaawansowane` -> `Niestandardowe` i zaktualizuj pola. Następnie uruchom ten dokument ponownie lub `Ctrl+A` -> `F9`.

WYMAGANIA

W ramach LAB D przygotowane powinny zostać:

- pojedyncza strona HTML ze skryptem ładowanym z zewnętrznego pliku JS
- pole tekstowe (input typu „text”) do wprowadzania adresu
- przycisk „Pogoda”, po kliknięciu którego wykonywane jest zapytanie asynchroniczne:
  - do API Current Weather: https://openweathermap.org/current za pomocą XMLHttpRequest
  - do API 5 day forecast: https://openweathermap.org/forecast5 za pomocą Fetch API
- obsługa zwrotki z obu API – wypisanie pogody bieżącej oraz prognoz poniżej pola wyszukiwania.

Wygeneruj własny lub wykorzystaj gotowy klucz do API: 7ded80d91f2b280ec979100cc8bbba94

W przypadku blokady można posiłkować się filmem: https://www.youtube.com/watch?v=WoKp2qDFxKk jednakże spróbuj rozwiązać ten problem samodzielnie!

Prowadzący omówi powyższe wymagania. Upewnij się, czy wszystko rozumiesz.

Tu umieść swoje notatki:

…notatki…

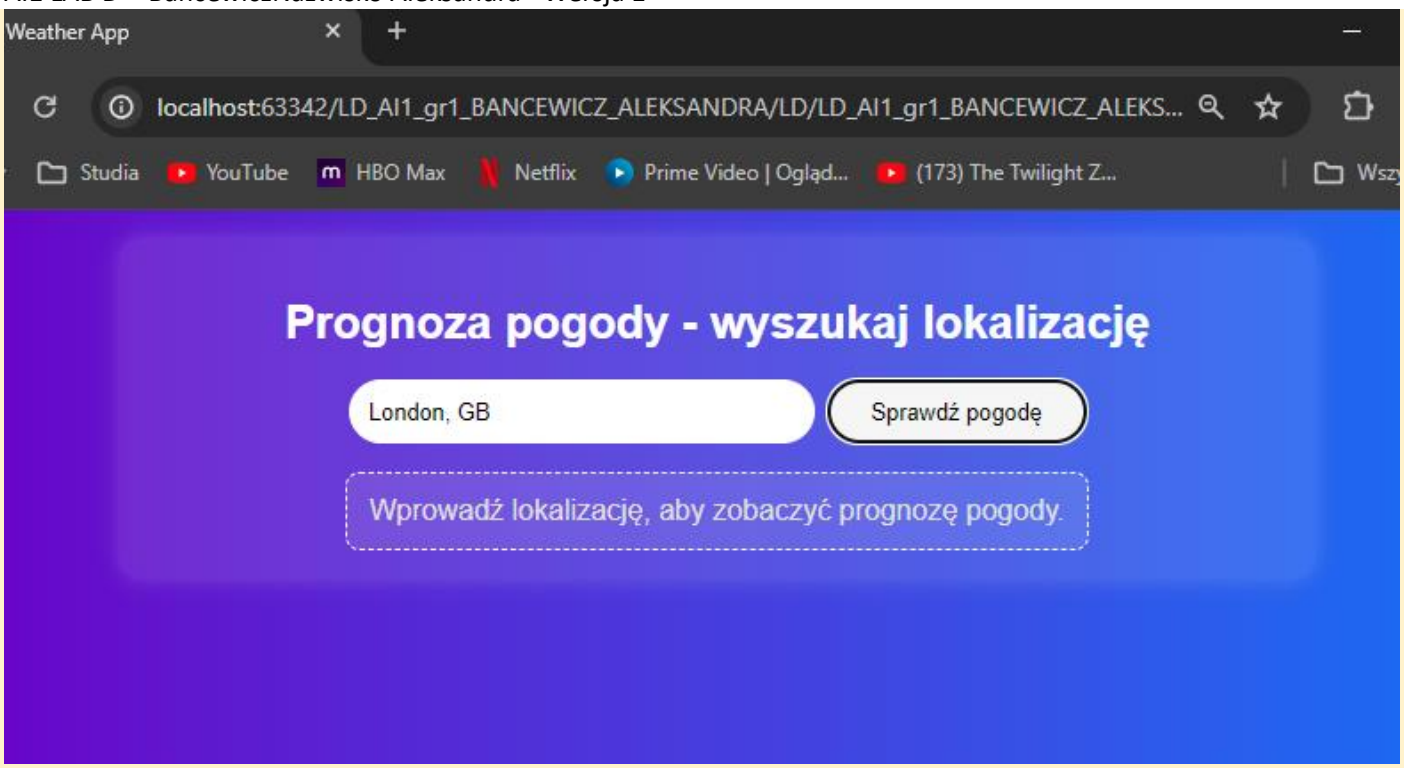
BADANIE API

Poświęć kilka minut na wykonanie przykładowych zapytań do API z poziomu pasku adresu przeglądarki. Podaj wymagane parametry dla osiągnięcia różnych wyników. Zbadaj odpowiedzi API, aby uzyskać pełen obraz wymagań i możliwości API.

IMPLEMENTACJA

Tradycyjnie implementację należy zacząć od zbudowania w HTML + CSS wszystkich wymaganych elementów / placeholderów na te elementy. Następnie krok po kroku należy implementować poszczególne zachowania.

Wstaw zrzut ekranu zawierającego stronę ze wszystkimi elementami, tj. pole tekstowe, przycisk, miejsce do wyświetlenia pogody i prognozy:

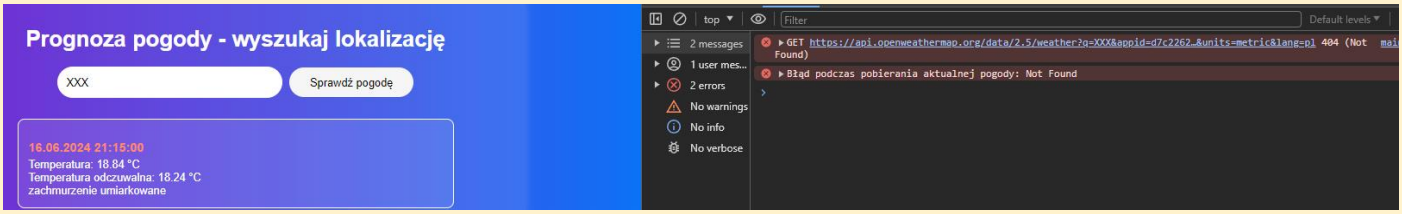


Punkty:	0	1

Wstaw zrzut ekranu kodu odpowiedzialnego za wysyłanie żądania do current za pomocą XMLHttpRequest:

```
async getCurrentWeather() : Promise<void>  {
    const city : string  = document.getElementById( elementId: "search-input").value.trim();
    if (!city) {
        alert("Wpisz nazwę miasta.");
        return;
    }
    const currentWeatherUrl = this.createUrl(CURRENT_WEATHER_URL, city);

    const request : XMLHttpRequest  = new XMLHttpRequest();
    request.open( method: "GET", currentWeatherUrl,  async: true);

    request.onload = () : void  => {
        if (request.status >= 200 && request.status < 400) {
            const data = JSON.parse(request.responseText);
            this.weatherData.push(data);
            this.getForecast();
            this.displayWeather();
        } else {
            console.error("Błąd podczas pobierania aktualnej pogody:", request.statusText);
        }
    };

    request.onerror = () : void  => {
        console.error("Błąd podczas zapytania XMLHttpRequest");
    };

    request.send();
}
```

Wstaw zrzut ekranu pokazujący otrzymaną odpowiedź za pomocą `console.log()` w przeglądarce.

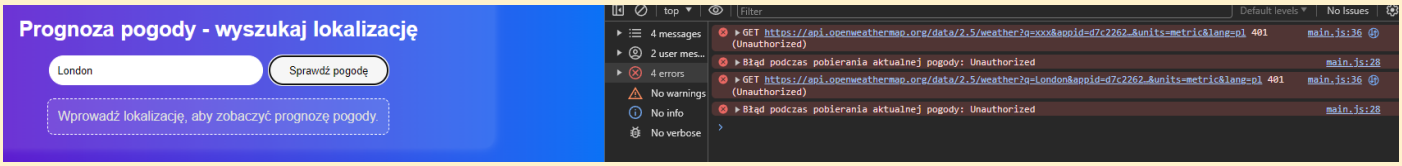


Punkty:	0	1

Wstaw zrzut ekranu kodu odpowiedzialnego za wysyłanie żądania do forecast za pomocą Fetch:

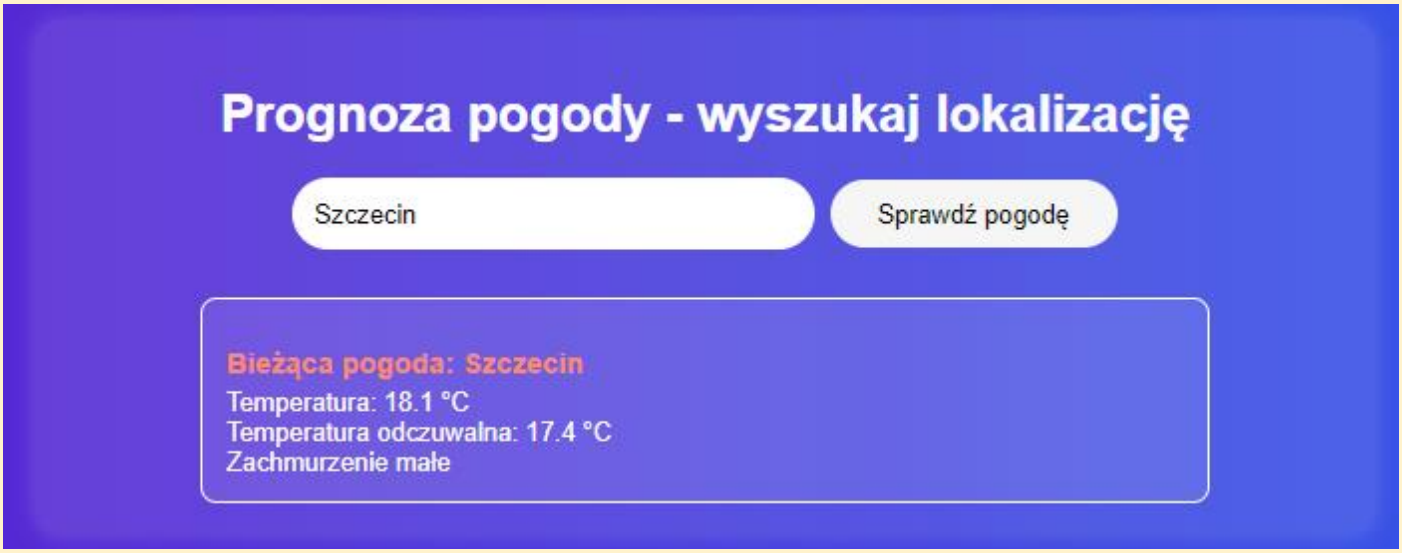
```
async getForecast() : Promise<void>  {
    const city : string  = document.getElementById( elementId: "search-input").value.trim();
    const forecastUrl = this.createUrl(FORECAST_URL, city);
    try {
        const response : Response  = await fetch(forecastUrl);
        const data = await response.json();
        this.weatherData.push(...data.list);
    } catch (error) {
        console.error("Błąd podczas pobierania prognozy pogody:", error);
    }
}
```

Wstaw zrzut ekranu pokazujący otrzymaną odpowiedź za pomocą `console.log()` w przeglądarce.



Punkty:	0	1

Wstaw zrzut ekranu przedstawiającego wizualizację prognoz pogody:



Upewnij się, że widoczne są pasek wyszukiwania ze wskazaną miejscowością, a także zarówno pogoda bieżąca jak i prognozy pogody.

Punkty:	0	1

COMMIT PROJEKTU DO GIT

Zacommituj i pushnij swoje rozwiązanie do repozytorium GIT.

Upewnij się, czy wszystko dobrze się wysłało. Jeśli tak, to z poziomu przeglądarki utwórz branch o nazwie `lab-c` na podstawie głównej gałęzi kodu.

Podaj link do brancha `lab-d` w swoim repozytorium:

…link, np. https://github.com/inazwisko/ai1-lab/tree/lab-d…

PODSUMOWANIE

W kilku zdaniach podsumuj zdobyte podczas tego laboratorium umiejętności.

…podsumowanie…

- **Rozumienie i Wykorzystanie API**:
  - W projekcie użyliśmy OpenWeatherMap API do pobierania danych pogodowych. Klucz API był niezbędny do autoryzacji żądań. Ważne jest, aby klucz API był poprawny i nie wygasł, a także aby dobrze zarządzać nim w kodzie, np. przechowywać go w zmiennych środowiskowych dla większego bezpieczeństwa.
- **Asynchroniczne Programowanie w JavaScript**:
  - Asynchroniczne funkcje `async` i `await` zostały wykorzystane do obsługi żądań API. To pozwala na wykonywanie kodu w sposób, który nie blokuje głównego wątku przeglądarki, co jest kluczowe dla responsywności aplikacji webowych.
- **Obsługa Błędów**:
  - Wprowadzenie bloków `try...catch` do obsługi błędów w czasie rzeczywistym pozwala na wyświetlanie przyjaznych komunikatów użytkownikowi w przypadku problemów z siecią lub błędów API, co poprawia doświadczenie użytkownika.
- **Przetwarzanie i Wyświetlanie Danych**:
  - Dane pogodowe zostały przetworzone w sposób umożliwiający wyświetlanie średnich wartości temperatury i opisu pogody dla najbliższych dni. Dzięki temu użytkownik otrzymuje klarowną

informację o prognozowanej pogodzie, zamiast surowych danych.

```
const dailyAverage : ({...})[]  = Object.keys(dailyForecast).slice(0, 3).map(date : string  => {
    const dayData = dailyForecast[date];
    const temps = dayData.map(d => d.main.temp);
    const feelsLikeTemps = dayData.map(d => d.main.feels_like);
    const descriptions = dayData.map(d => d.weather[0].description);

    const averageTemp : number  = temps.reduce((a, b) => a + b, 0) / temps.length;
    const averageFeelsLike : number  = feelsLikeTemps.reduce((a, b) => a + b, 0) / feelsLikeTemps.length;
    const mostCommonDescription = descriptions.sort((a, b) =>
        descriptions.filter(v => v === a).length - descriptions.filter(v => v === b).length
    ).pop();

    return {
        date,
        temp: averageTemp,
        feels_like: averageFeelsLike,
        description: mostCommonDescription
    };
});
```

- **Stylizacja CSS**:
  - Stylizacja strony za pomocą CSS pozwala na stworzenie estetycznej i przyjaznej dla użytkownika interfejsu. Użycie gradientów, cieni i przejść kolorów poprawia wizualne wrażenia z aplikacji.
- **Modularność i Czytelność Kodu**:
  - Kod został zorganizowany w klasę `WeatherApp`, co pozwala na lepszą organizację funkcjonalności i łatwiejsze zarządzanie kodem. Taka modularność ułatwia przyszłą rozbudowę i utrzymanie aplikacji.
- **Interaktywność Strony**:
  - Strona jest interaktywna dzięki elementom HTML takim jak przyciski i pola tekstowe oraz zdarzeniom JavaScript takim jak `click`, co pozwala użytkownikowi na dynamiczne wyszukiwanie i wyświetlanie pogody.

Zweryfikuj kompletność sprawozdania. Utwórz PDF i wyślij w terminie.