Instrukcja dla Frontendowca - Praca z .NET Blazor i IHttpClientFactory

Cel zadania: Twoim zadaniem jest stworzenie komponentu w aplikacji .NET Blazor, który pobierze dane z dwóch różnych endpointów API i wyświetli je użytkownikowi w kreatywny sposób. Wykorzystasz do tego celu IHttpClientFactory do zarządzania klientami HTTP.

Technologie: .NET Blazor (C#, HTML, CSS)

Wymagania:

1. **Konfiguracja HttpClient:** Skonfiguruj HttpClient w usłudze IServiceCollection w pliku Program.cs (lub Startup.cs w starszych wersjach Blazor). Użyj IHttpClientFactory do tworzenia nazwanych lub podstawowych klientów HTTP.
2. **Pobieranie danych z API:** Stwórz serwis (lub użyj komponentu Razor Pages) do pobierania danych z dwóch podanych endpointów API:
   - https://my-wonderful-api-acgvdvaaa5d4b9ez.northeurope-01.azurewebsites.net/api/sensor/gettemperature
   - https://my-wonderful-api-acgvdvaaa5d4b9ez.northeurope-01.azurewebsites.net/api/sensor/getcurrentlockstate
3. **Model danych:** Zdefiniuj klasy C# (np. TemperatureData, LockStateData) odpowiadające strukturom odpowiedzi JSON z obu endpointów. Użyj atrybutów JsonPropertyName z przestrzeni nazw System.Text.Json.Serialization (jeśli używasz domyślnego serializatora JSON) lub odpowiednich atrybutów z biblioteki Newtonsoft.Json (jeśli jej używasz), aby poprawnie zmapować pola JSON na właściwości klas C#.
4. **Wyświetlanie danych:** Stwórz komponent Blazor (.razor), który będzie:
   - Wstrzykiwał utworzony serwis do pobierania danych.
   - Wywoływał metody serwisu w celu pobrania danych z obu endpointów.
   - Przechowywał pobrane dane w odpowiednich właściwościach.
   - **Kreatywnie** wyświetlał pobrane dane na stronie. Sposób prezentacji danych jest Twoją inwencją twórczą. Możesz użyć tabel, kart, ikon, wskaźników, wykresów (jeśli masz taką wiedzę i chcesz ją wykorzystać - nie jest to wymagane), itp. Ważne, aby dane z obu endpointów były czytelnie przedstawione użytkownikowi.
5. **Obsługa błędów (opcjonalnie, ale zalecane):** Zaimplementuj podstawową obsługę błędów, np. wyświetlenie komunikatu, gdy nie uda się pobrać danych z API.
6. **Asynchroniczność:** Pamiętaj o używaniu słów kluczowych async i await podczas

wykonywania operacji asynchronicznych, takich jak wywołania HTTP.

Kroki do wykonania:

1. **Utwórz projekt Blazor:** Utwórz nowy projekt Blazor WebAssembly.
2. **Zdefiniuj modele danych:** Utwórz klasy C# w osobnym folderze (np. Models) odpowiadające strukturom JSON z API.

```
C#
// Models/TemperatureData.cs
using System.Text.Json.Serialization;

public class TemperatureData
{
    [JsonPropertyName("value")]
    public double Value { get; set; }

    [JsonPropertyName("sensorId")]
    public string SensorId { get; set; }

    [JsonPropertyName("name")]
    public string Name { get; set; }

    [JsonPropertyName("enqueuedTime")]
    public DateTime EnqueuedTime { get; set; }
}

// Models/LockStateData.cs
using System.Text.Json.Serialization;

public class LockStateData
{
    [JsonPropertyName("value")]
    public string Value { get; set; }

    [JsonPropertyName("sensorId")]
    public string SensorId { get; set; }

    [JsonPropertyName("name")]
    public string Name { get; set; }
```

```
    [JsonPropertyName("enqueuedTime")]
    public DateTime EnqueuedTime { get; set; }
}
```

3. **Zainstaluj Microsoft.Extensions.Http w NuGet Manager**

4. **Skonfiguruj IHttpClientFactory:** Otwórz plik Program.cs i dodaj konfigurację HttpClient. Możesz użyć nazwanego klienta lub podstawowego.
   Przykład Program.cs dla Blazor WebAssembly:

```
C#
// Program.cs
using Microsoft.AspNetCore.Components.Web;
using Microsoft.AspNetCore.Components.WebAssembly.Hosting;
// ... inne usingi

var builder = WebAssemblyHostBuilder.CreateDefault(args);

builder.RootComponents.Add<App>("#app");
builder.RootComponents.Add<HeadOutlet>("head::after");

builder.Services.AddHttpClient("MyApiClient", client =>
{
  client.BaseAddress = new
Uri("https://my-wonderful-api-acgvdvaaa5d4b9ez.northeurope-01.azurewebsites.net/api/sensor/"
);
});


await builder.Build().RunAsync();
```

5. **Utwórz serwis do pobierania danych (opcjonalnie, ale zalecane dla lepszej organizacji kodu):**

```
C#
// Services/ApiService.cs
using System.Net.Http;
using System.Net.Http.Json;
using System.Threading.Tasks;
using YourProjectName.Models; // Zmień na przestrzeń nazw Twojego projektu

public class ApiService
```

```
{
    private readonly HttpClient _httpClient;

    public ApiService(IHttpClientFactory httpClientFactory)
    {
        _httpClient = httpClientFactory.CreateClient("MyApiClient");
    }

  public async Task<List<TemperatureData>?> GetTemperatureDataAsync()
  {
      return await
_httpClient.GetFromJsonAsync<List<TemperatureData>>("https://my-wonderful-api-acgvdvaaa5d4b9ez.northeurope-01.azurewebsites.net/api/sensor/gettemperature");
  }

  public async Task<LockStateData?> GetLockStateDataAsync()
  {
      return await
_httpClient.GetFromJsonAsync<LockStateData>("https://my-wonderful-api-acgvdvaaa5d4b9ez.northeurope-01.azurewebsites.net/api/sensor/getcurrentlockstate");
  }
}
```

Zarejestruj serwis w Program.cs (lub Startup.cs):

```
C#
// Program.cs
// ...
builder.Services.AddScoped<ApiService>();
// ...
```

6. **Utwórz komponent Blazor do wyświetlania danych:**

```
Razor CSHTML
// Pages/SensorDataDisplay.razor
@page "/sensordata"
@using YourProjectName.Models // Zmień na przestrzeń nazw Twojego projektu
@using YourProjectName.Services // Zmień na przestrzeń nazw Twojego serwisu (jeśli go utworzyłeś)
```

```
@inject ApiService ApiService

<h1>Dane z Sensorów</h1>

@if (temperatureData == null && lockStateData == null && !errorOccurred)
{
    <p>Ładowanie danych...</p>
}
else if (errorOccurred)
{
    <p class="text-danger">Wystąpił błąd podczas pobierania danych.</p>
}
else
{
    <h2>Temperatura</h2>
    @if (temperatureData != null && temperatureData.Any())
    {
        // Twoja kreatywna implementacja wyświetlania danych temperatury
        <ul>
            @foreach (var temp in temperatureData)
            {
                <li>Czujnik: @temp.Name (@temp.SensorId), Temperatura: @temp.Value
°C, Czas: @temp.EnqueuedTime</li>
            }
        </ul>
    }
    else
    {
        <p>Brak danych temperatury.</p>
    }

    <h2>Stan Zamka</h2>
    @if (lockStateData != null)
    {
        // Twoja kreatywna implementacja wyświetlania danych stanu zamka
        <div>
            Czujnik: @lockStateData.Name (@lockStateData.SensorId)
            <br/>
            Stan: <strong>@lockStateData.Value</strong>
```

```
            <br/>
            Czas: @lockStateData.EnqueuedTime
        </div>
    }
    else
    {
        <p>Brak danych stanu zamka.</p>
    }
}

@code {
    private List<TemperatureData>? temperatureData;
    private LockStateData? lockStateData;
    private bool errorOccurred = false;

    protected override async Task OnInitializedAsync()
    {
        try
        {
            temperatureData = await ApiService.GetTemperatureDataAsync();
            lockStateData = await ApiService.GetLockStateDataAsync();
        }
        catch (Exception)
        {
            errorOccurred = true;
        }
    }
}
```

7. **Dodaj link do komponentu w nawigacji (jeśli używasz Blazor WebAssembly lub Blazor Server z domyślnym layoutem):**

```
Razor CSHTML
// Shared/NavMenu.razor
<div class="nav-item px-3">
    <NavLink class="nav-link" href="sensordata">
        <span class="oi oi-list-rich" aria-hidden="true"></span> Dane Sensorów
    </NavLink>
</div>
```

Inwencja Twórcza - Wyświetlanie Danych:

Poniżej kilka sugestii, jak możesz kreatywnie wyświetlić dane:

- **Temperatura:**
  - Użyj wskaźników (np. progress bar) wizualizujących wartość temperatury w określonym zakresie.
  - Wyświetlaj kolor tła lub ikonę zmieniającą się w zależności od temperatury (np. niebieski dla niskiej, czerwony dla wysokiej).
  - Stwórz proste karty z ikonami termometru.
- **Stan Zamka:**
  - Użyj ikon kłódki (otwartej/zamkniętej) w zależności od wartości.
  - Wyświetlaj stan zamka dużym, czytelnym tekstem z odpowiednim kolorem (np. zielony dla "Open", czerwony dla "Closed").
  - Dodaj animację zmiany stanu zamka.
- **Ogólne:**
  - Użyj tabel z niestandardowym stylem CSS.
  - Wyświetlaj dane w formie kart z cieniem i zaokrąglonymi rogami.
  - Zgrupuj dane według typu czujnika.
  - Dodaj informacje o czasie od ostatniego odczytu.

Pamiętaj:

- Zastąp "YourProjectName" rzeczywistą nazwą Twojego projektu.
- Upewnij się, że Twoja aplikacja ma dostęp do internetu, aby pobrać dane z API.
- Testuj swój komponent, aby upewnić się, że dane są poprawnie pobierane i wyświetlane.

Powodzenia w realizacji zadania! Bądź kreatywny i pokaż, jak interesująco można przedstawić dane z API w Blazor