Rapport Personnel de Projet : Solution IoT ESEO

Auteur : Oussama JERIDI

Table des matières

Chapitre 1

Description Technique de l'Implémentation

Dans ce projet, ma mission principale a consisté à concevoir et implémenter la **couche de communication avec les objets connectés (Devices)** ainsi que les **services d'arrière-plan** pour l'automatisation de la collecte de données.

1.1 Architecture de Communication

J'ai mis en place un système de communication hybride capable de s'adapter aux capacités technologiques des différents capteurs :

- **Interrogation Active (HttpPull)** : Utilisation d'un `BackgroundService` .NET pour interroger cycliquement les sondes passives.
- **Réception Passive (HttpPush / Webhooks)** : Mise à disposition d'une API REST pour les sondes capables de pousser leurs données en temps réel.
- **Abstraction Métier** : Centralisation de la logique dans le `DeviceCommunicationService.cs`, permettant une validation et un traitement uniforme des données, quelle que soit leur source.

1.2 Focus sur des Implémentations Complexes

1.2.1 Gestion de la Durée de Vie des Services (Scope Management)

L'une des difficultés majeures a été l'injection de services "Scoped" (comme les repositories accédant à la base de données) au sein d'un service "Singleton" (`BackgroundService`).

```
1  protected override async Task ExecuteAsync(CancellationToken
       stoppingToken)
2  {
3      while (!stoppingToken.IsCancellationRequested)
4      {
5          // Cr ation manuelle d'un scope pour acc der aux services
               scoped
6          using (var scope = _scopeFactory.CreateScope())
7          {
8              var deviceComm =
                   scope.ServiceProvider.GetRequiredService<IDeviceCommunicationService>
9              var sondeService =
                   scope.ServiceProvider.GetRequiredService<ISondeService>();
10
```

```
            // Filtrage et interrogation des sondes configur es en
               mode Pull
            var pullSondes = (await sondeService.GetAllAsync())
                .Where(s => s.CanalCommunication ==
                   CanalCommunication.HttpPull && s.EstActif);

            foreach (var sonde in pullSondes)
            {
                await deviceComm.PullDataFromSondeAsync(sonde.Id);
            }
        }
        // D lai de 5 minutes entre chaque cycle de collecte
        await Task.Delay(TimeSpan.FromMinutes(5), stoppingToken);
    }
}
```

Listing 1.1 – Extrait de HttpPullBackgroundService.cs

Analyse technique : L'utilisation du `IServiceScopeFactory` est ici vitale. Elle garantit que chaque cycle d'interrogation dispose de sa propre instance de contexte de base de données, évitant ainsi les fuites de mémoire et les conflits d'accès aux données.

1.2.2 Stratégie de Résilience et d'Authentification

La communication avec des systèmes tiers nécessite une robustesse face à l'hétérogénéité des configurations.

```
private async Task<DeviceDataDto> GetDataFromDevice(string url, string
    credentials)
{
    var client = _httpClientFactory.CreateClient();

    // Support de l'authentification Basic Auth standard
    if (!string.IsNullOrEmpty(credentials))
    {
        var authValue =
            Convert.ToBase64String(Encoding.UTF8.GetBytes(credentials));
        client.DefaultRequestHeaders.Authorization = new
            AuthenticationHeaderValue("Basic", authValue);
    }

    // Strat gie de Fallback sur les points d'entr e (Endpoints)
    var response = await client.GetAsync($"{url}/data");
    if (response.StatusCode == HttpStatusCode.NotFound)
    {
        response = await client.GetAsync(url); // Tentative sur l'URL
            racine
    }

    response.EnsureSuccessStatusCode();
    // D s rialisation flexible (insensible    la casse du JSON)
    var json = await response.Content.ReadAsStringAsync();
    return JsonSerializer.Deserialize<DeviceDataDto>(json, new
        JsonSerializerOptions { PropertyNameCaseInsensitive = true });
}
```

Listing 1.2 – Extrait de la logique de résilience dans DeviceCommunicationService.cs

Analyse technique : Cette implémentation gère trois aspects critiques : la **sécurité** (Basic Auth), la **flexibilité** (fallback d'URL) et la **robustesse** (gestion de la casse JSON), ce qui rend le système compatible avec une large gamme de matériels IoT.

Chapitre 2

Suivi des Défis et Problèmes Rencontrés

— **Validation Temporelle des Données** : J'ai dû faire face à des décalages d'horloge sur certains périphériques simulant des données futures. J'ai implémenté une garde-fou rejetant tout relevé ayant plus de 5 minutes d'avance sur l'heure UTC du serveur pour garantir l'intégrité de l'historique.
— **Diagnostic de Connectivité** : L'absence de matériel physique a complexifié les tests. J'ai résolu cela en développant un simulateur (`DeviceSimulatorController.cs`) et en intégrant des mesures de latence (`Stopwatch`) pour fournir un retour visuel en temps réel sur l'état du réseau dans l'interface Blazor.
— **Couplage Cyclique** : J'ai dû refactorer la structure des DTOs pour éviter les dépendances circulaires entre la couche Application et les services de communication, un défi classique en Clean Architecture.

Chapitre 3

Environnement de Développement [Hors notation]

— **Outil choisi** : **VS Code**.
— **Raison du choix** : Sa légèreté et sa modularité. J'utilise VS Code pour la majorité de mes projets personnels, il était donc déjà configuré sur ma machine.

Avis Critique :

— **Points Positifs** : L'extension "REST Client" a été indispensable pour tester mes Webhooks via des fichiers `.http`. La rapidité de l'interface et la gestion fluide de Git via le terminal intégré sont de vrais atouts.
— **Points Négatifs** : Le débogage de processus asynchrones complexes (Background Tasks) est moins intuitif que sur Visual Studio "complet", nécessitant une manipulation plus fine des configurations de lancement.

Chapitre 4

Bilan de l'Apprentissage

Ce projet a été un catalyseur pour ma compréhension du développement .NET moderne. Les points clés que je retiens sont :

1. **Maîtrise des Background Tasks** : Comprendre comment orchestrer des traitements asynchrones de longue durée sans bloquer l'application web.
2. **Gestion de la Résilience HTTP** : Utilisation de `IHttpClientFactory` et mise en place de stratégies de contournement pour les erreurs réseau.
3. **Architecture de Données IoT** : L'importance cruciale de la validation et du nettoyage des données au point d'entrée (Webhook/Pull) pour préserver la qualité de la base de données décisionnelle.