COURS

Écrivez du code back-end en .NET C#

P1C6 - Connectez le front-end fourni et documentez votre application

Vérifiez votre travail

Le ViewModel. Un seul attribut aligne le front-end sur le contrat, sans toucher aux écrans :

```
public sealed class LeaveRequestViewModel
{
    public int Id { get; set; }
    public int EmployeeId { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }

    [JsonPropertyName("workingDays")]
    public int Days { get; set; }

    public string Status { get; set; } = "";
}
```

HttpLeaveApi. Chaque méthode correspond à une route, et toutes partagent la même gestion des erreurs :

```
public sealed class HttpLeaveApi : ILeaveApi
{
    private readonly HttpClient _http;

    public HttpLeaveApi(HttpClient http) => _http = http;

    public async Task<double> GetBalanceAsync(int employeeId)
    {
        var response = await _http.GetAsync($"api/employees/{employeeId}/balance");
        var balance = await ReadAsync<BalanceResponse>(response);
        return balance.Balance;
    }

    public async Task<IReadOnlyList<LeaveRequestViewModel>> GetRequestsAsync(string? status)
    {
        var url = status is null
            ? "api/leave-requests"
            : $"api/leave-requests?status={Uri.EscapeDataString(status)}";

        return await ReadAsync<List<LeaveRequestViewModel>>(await _http.GetAsync(url));
    }

    public async Task<LeaveRequestViewModel> CreateRequestAsync(NewLeaveRequestViewModel request) =>
        await ReadAsync<LeaveRequestViewModel>(await _http.PostAsJsonAsync("api/leave-requests", request));

    public async Task<LeaveRequestViewModel> ApproveAsync(int requestId) =>
        await ReadAsync<LeaveRequestViewModel>(await _http.PostAsync($"api/leave-requests/{requestId}/approval", null));

    public async Task<LeaveRequestViewModel> RefuseAsync(int requestId) =>
        await ReadAsync<LeaveRequestViewModel>(await _http.PostAsync($"api/leave-requests/{requestId}/refusal", null));

    private static async Task<T> ReadAsync<T>(HttpResponseMessage response)
    {
        if (response.IsSuccessStatusCode)
        {
            return (await response.Content.ReadFromJsonAsync<T>())!;
        }

        throw new LeaveApiException(await ReadErrorMessageAsync(response));
    }
```

```
    private static async Task<string> ReadErrorMessageAsync(HttpResponseMessage response)
    {
        var fallback = $"Erreur {(int)response.StatusCode}";

        if (response.Content.Headers.ContentType?.MediaType != "application/problem+json")
        {
            return fallback;
        }

        var problem = await response.Content.ReadFromJsonAsync<ApiProblem>();

        return problem?.Errors?.Values.SelectMany(messages => messages).FirstOrDefault()
            ?? problem?.Title
            ?? fallback;
    }

    private sealed record BalanceResponse(double Balance);

    private sealed record ApiProblem(string? Title, Dictionary<string, string[]>? Errors);
}
```

Le record `ApiProblem` ne lit que les deux champs utiles de Problem Details. La classe `ProblemDetails` d'ASP.NET Core n'est pas disponible dans un projet Blazor WebAssembly, et il serait inutile d'ajouter une dépendance pour deux propriétés. La vérification du type de contenu protège contre une réponse d'erreur qui ne serait pas du JSON, par exemple une page renvoyée par un proxy.

Le Program.cs du front-end. L'adresse vient de la configuration, et la ligne de `MockLeaveApi` a disparu :

```
var apiBaseUrl = builder.Configuration["Api:BaseUrl"]
    ?? throw new InvalidOperationException("Api:BaseUrl is missing in wwwroot/appsettings.json.");

builder.Services.AddScoped(_ => new HttpClient { BaseAddress = new Uri(apiBaseUrl) });
builder.Services.AddScoped<ILeaveApi, HttpLeaveApi>();
```

Conservez `MockLeaveApi` dans le projet : l'équipe front-end pourra encore s'en servir pour travailler sur un écran sans lancer l'API.

Le Program.cs de l'API. Voici les ajouts du chapitre, à leur place dans le fichier :

```
var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();
builder.Services.AddProblemDetails();
builder.Services.AddOpenApi();

var frontOrigins = builder.Configuration.GetSection("Front:Origins").Get<string[]>() ?? [];

builder.Services.AddCors(options =>
    options.AddPolicy("Front", policy => policy
        .WithOrigins(frontOrigins)
        .WithMethods("GET", "POST")
        .WithHeaders("Content-Type")));

builder.Services.AddDbContext<HamacDbContext>(options =>
    options.UseSqlite(builder.Configuration.GetConnectionString("Hamac")));

builder.Services.AddSingleton<ILeaveEntitlementPolicy, PermanentContractPolicy>();
builder.Services.AddSingleton<ILeaveEntitlementPolicy, FixedTermContractPolicy>();
builder.Services.AddSingleton<ILeaveEntitlementPolicy, ApprenticeshipPolicy>();
builder.Services.AddSingleton<ILeaveEntitlementPolicy, InternshipPolicy>();
builder.Services.AddSingleton<LeavePolicyResolver>();

builder.Services.AddScoped<ILeaveNotifier, NotificationService>();
builder.Services.AddScoped<LeaveService>();

var app = builder.Build();
```

```
using (var scope = app.Services.CreateScope())
{
    var db = scope.ServiceProvider.GetRequiredService<HamacDbContext>();
    DemoData.Reset(db);
}

app.UseExceptionHandler();
app.UseCors("Front");

if (app.Environment.IsDevelopment())
{
    app.MapOpenApi();
}

app.MapControllers();

app.Run();
```

La politique des stages. `Math.Max` garantit le contrat commun des politiques, même avant l'arrivée de la personne :

```
public sealed class InternshipPolicy : ILeaveEntitlementPolicy
{
    private const double DaysPerFullMonth = 2;

    public string ContractType => ContractTypes.Internship;

    public double ComputeEntitlement(Employee employee, DateTime today)
    {
        var fullMonths = Math.Floor((today - employee.HireDate).TotalDays / 30);
        return Math.Max(0, fullMonths) * DaysPerFullMonth;
    }
}
```

Les tests. Les deux tests unitaires reprennent les dates du test de la politique des CDD, pour faciliter la comparaison :

```
[Fact]
public void Internship_policy_grants_two_days_per_full_month()
{
    var policy = new InternshipPolicy();
    var employee = new Employee { HireDate = new DateTime(2026, 3, 15) };

    var entitlement = policy.ComputeEntitlement(employee, new DateTime(2026, 10, 20));

    Assert.Equal(14.0, entitlement);
}

[Fact]
public void Internship_policy_grants_nothing_before_the_internship_starts()
{
    var policy = new InternshipPolicy();
    var employee = new Employee { HireDate = new DateTime(2026, 11, 2) };

    var entitlement = policy.ComputeEntitlement(employee, new DateTime(2026, 10, 26));

    Assert.Equal(0.0, entitlement);
}
```

Entre le 15 mars et le 20 octobre, il s'écoule 219 jours, soit 7 mois complets de 30 jours, et donc 14 jours de droits. Le test d'API vérifie que toute la chaîne fonctionne, du conteneur jusqu'au DTO. Il échoue avec un `500` tant que `InternshipPolicy` n'est pas enregistrée :

```
[Fact]
public async Task Balance_of_intern_is_available_before_arrival()
{
    var client = _factory.CreateClient();

    var balance = await client.GetFromJsonAsync<BalanceDto>("/api/employees/6/balance");

    Assert.Equal(0.0, balance!.Balance);
}
```

Le README. Voici une version qui répond aux questions de Tomás sans recopier le code :

```
# Hamac

Application interne de gestion des congés : les salariés consultent leur solde
et déposent leurs demandes, les managers les approuvent ou les refusent.

## Démarrer

Prérequis : SDK .NET 10.

    dotnet run --project src/Hamac.Api      # API sur http://localhost:5110
    dotnet run --project src/Hamac.Front    # Front-end sur http://localhost:5200

La base SQLite de démonstration est réinitialisée à chaque démarrage de l'API.

## Configuration

| Clé | Fichier | Rôle |
| --- | --- | --- |
| ConnectionStrings:Hamac | src/Hamac.Api/appsettings.json | Base de données |
| Front:Origins | src/Hamac.Api/appsettings.Development.json | Origines autorisées par CORS |
| Api:BaseUrl | src/Hamac.Front/wwwroot/appsettings.json | Adresse de l'API |

## Architecture

- src/Hamac.Api : contrôleurs (HTTP uniquement), LeaveService (règles métier),
  politiques de congés (une classe par type de contrat), HamacDbContext (EF Core).
- src/Hamac.Front : Blazor WebAssembly. Les écrans dépendent de ILeaveApi ;
  HttpLeaveApi appelle l'API, MockLeaveApi fournit des données simulées.
- tests/Hamac.Tests : tests xUnit (API et politiques).

Ajouter un type de contrat : une constante dans ContractTypes, une classe qui
implémente ILeaveEntitlementPolicy, une ligne dans Program.cs, un test unitaire.

## API

Contrat complet : http://localhost:5110/openapi/v1.json (en développement).
Exemples prêts à l'emploi : src/Hamac.Api/Hamac.Api.http.

| Verbe | Route | Rôle |
| --- | --- | --- |
| GET | api/employees/{id}/balance | Solde d'un salarié |
| GET | api/leave-requests?status= | Liste des demandes, filtrable par statut |
| GET | api/leave-requests/{id} | Détail d'une demande |
| POST | api/leave-requests | Dépôt d'une demande (201) |
| POST | api/leave-requests/{id}/approval | Approbation (409 si déjà traitée) |
| POST | api/leave-requests/{id}/refusal | Refus (409 si déjà traitée) |

## Tests

    dotnet test

## Règles métier et limites connues

Les règles de calcul des droits sont validées par Samuel Okafor (RH) :
ne modifiez pas une règle sans son accord.

- Les jours fériés ne sont pas déduits du nombre de jours ouvrés.
- Les messages d'erreur de l'API sont en français, quelle que soit la langue de l'écran.
```

```
- La politique des CDD renvoie un droit négatif pour une date d'entrée future (à confirmer avec Samuel).
```

La distinction attendue est la suivante : une application qui **fonctionne** tourne sur le poste de la personne qui l'a écrite ; une application **livrée** peut être lancée, comprise et modifiée par quelqu'un d'autre, sans l'aide de son auteur. Le README, les tests et le contrat documenté sont ce qui fait passer Hamac de l'une à l'autre.

Erreurs fréquentes

- **Renommer** `workingDays` **dans l'API** pour l'aligner sur le ViewModel : vous cassez le contrat publié, les tests et le fichier `.http`.
- **Régler l'erreur CORS avec** `AllowAnyOrigin()` : l'erreur disparaît, mais n'importe quel site peut appeler l'API depuis le navigateur de vos utilisateurs.
- **Écrire l'origine avec une barre oblique finale**, comme `http://localhost:5200/` : le navigateur envoie l'origine sans barre oblique, elle ne correspond pas, et l'erreur CORS reste affichée dans la console.
- **Écrire les chemins avec une barre oblique initiale**, comme `/api/leave-requests` : l'appel fonctionne ici, mais il casserait le jour où l'API serait publiée sous un sous-chemin.
- **Oublier** `Math.Max` **dans** `InternshipPolicy` : le solde de Zoé vaut -2 jours, et le test du stage qui n'a pas commencé échoue.
- **Modifier** `LeaveService` **pour gérer les stages** : le résolveur et le conteneur rendent cette modification inutile.