COURS

Écrivez du code back-end en .NET C#

P1C4 - Prenez en main ASP.NET Core

Vérifiez votre travail

Le DbContext et sa configuration. Le `DbContext` ne connaît plus sa base de données, il la reçoit :

```
public sealed class HamacDbContext : DbContext
{
    public HamacDbContext(DbContextOptions<HamacDbContext> options) : base(options)
    {
    }

    public DbSet<Employee> Employees => Set<Employee>();
    public DbSet<LeaveRequest> LeaveRequests => Set<LeaveRequest>();
}
```

Dans `appsettings.json`, ajoutez la chaîne de connexion :

```
"ConnectionStrings": {
  "Hamac": "Data Source=hamac.db"
}
```

Les politiques et le résolveur. Chaque politique déclare le type de contrat qu'elle traite, en s'appuyant sur les constantes de `ContractTypes` :

```
public sealed class FixedTermContractPolicy : ILeaveEntitlementPolicy
{
    private const double DaysPerFullMonth = 2.08;

    public string ContractType => ContractTypes.FixedTerm;

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

public sealed class LeavePolicyResolver
{
    private readonly IReadOnlyDictionary<string, ILeaveEntitlementPolicy> _policies;

    public LeavePolicyResolver(IEnumerable<ILeaveEntitlementPolicy> policies) =>
        _policies = policies.ToDictionary(policy => policy.ContractType);

    public ILeaveEntitlementPolicy For(string contractType) =>
        _policies.TryGetValue(contractType, out var policy)
            ? policy
            : throw new InvalidOperationException($"Unknown contract type: {contractType}");
}
```

`PermanentContractPolicy` et `ApprenticeshipPolicy` reçoivent la même propriété, avec `ContractTypes.Permanent` et `ContractTypes.Apprenticeship`. La méthode `For` ne change pas.

Program.cs. C'est désormais le seul fichier qui connaît les classes concrètes :

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

```
builder.Services.AddControllers();

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<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.MapControllers();

app.Run();
```

Le bloc de réinitialisation mérite une explication. Le `DbContext` est *scoped*, c'est-à-dire lié à une requête. Or, au démarrage, aucune requête n'est en cours. `CreateScope()` crée donc une portée temporaire, le temps de réinitialiser la base, puis la libère à la fin du bloc `using`, ce qui libère aussi le `DbContext`.

Le contrôleur. Son constructeur se réduit à une ligne :

```
public LeaveController(LeaveService leaveService) =>
    _leaveService = leaveService;
```

Les méthodes `GetBalance` et `Approve` ne changent pas.

Le filtrage. Dans `LeaveService.Approve`, la ligne et son commentaire provisoire sont remplacés par :

```
var request = _db.LeaveRequests.Find(requestId);
```

Le test du conteneur. Il demande au conteneur de l'application de construire le service, avec toutes ses dépendances :

```
[Fact]
public void Container_can_build_leave_service()
{
    using var scope = _factory.Services.CreateScope();

    var service = scope.ServiceProvider.GetRequiredService<LeaveService>();

    Assert.NotNull(service);
}
```

Si quelqu'un oublie un jour d'enregistrer une dépendance du service, ce test échouera avec le message du conteneur, avant toute mise en production.

La distinction attendue est la suivante : déplacer les `new` du contrôleur vers `Program.cs` aurait simplement **déplacé l'assemblage** ; en laissant le conteneur lire les constructeurs, vous l'avez **rendu déclaratif**. Chaque classe déclare ce dont elle a besoin, et `Program.cs` déclare comment le fournir. Pour ajouter les stages, une classe et une ligne suffiront.

Erreurs fréquentes

- **Enregistrer** `LeaveService` **en singleton** : il capture un seul `DbContext` pour toute l'application. En développement, le conteneur refuse de le résoudre.
- **Garder** `new HamacDbContext()` **dans** `Program.cs` **pour la réinitialisation** : ce `DbContext` n'a plus de constructeur sans paramètre, et le code ne compile plus. Passez par `CreateScope()`.
- **Remplacer** `ToList()` **par** `AsEnumerable()` : le nom change, mais la requête s'exécute toujours sans filtre et le tri se fait toujours en mémoire.
- **Tout faire en un seul commit** : si un test échoue, vous ne savez plus si la cause est l'injection ou le filtrage.