

1. Explication des défauts de notre code

Le scaffolding, que nous avons utilisé en partie 1 pour créer le contrôleur utilisateur, manipule directement la base de données en utilisant la classe de contexte.

```
public async Task<ActionResult<Utilisateur>> GetUtilisateurById(int id)
{
    var utilisateur = await _context.Utilisateurs.FindAsync(id);

    if (utilisateur == null)
    {
        return NotFound();
    }

    return utilisateur;
}
```

Bien que pratique le code généré par le scaffolding impose un couplage fort entre le contrôleur et la base de données. Ainsi, si nous souhaitons utiliser un autre ORM, comme Dapper, par exemple, il sera nécessaire de modifier le code des contrôleurs.

Une bonne pratique est d'utiliser une couche intermédiaire qui permettra d'assurer un couplage léger entre les contrôleurs et les données, ce qui nous permettra de modifier la couche des données sans modifier celle des contrôleurs.

Il existe (au moins) 2 solutions pour créer cette couche intermédiaire :

1. Appliquer le design pattern DAL (Data Access Layer) spécifié par Microsoft. Exemple de code ici : <https://nathanaelmarchand.developpez.com/tutoriels/dotnet/architecture-couches-decouplage-et-injection-dependances-avec-unity/>

Remarque : comme indiqué dans ce tutorial, ASP.Net Web API (.NET framework) nécessitait l'installation du package NuGet Unity pour gérer l'injection de dépendance. ASP.Net Core gérant déjà nativement le DI, l'installation d'Unity n'est plus nécessaire (même si on peut quand même toujours l'utiliser).

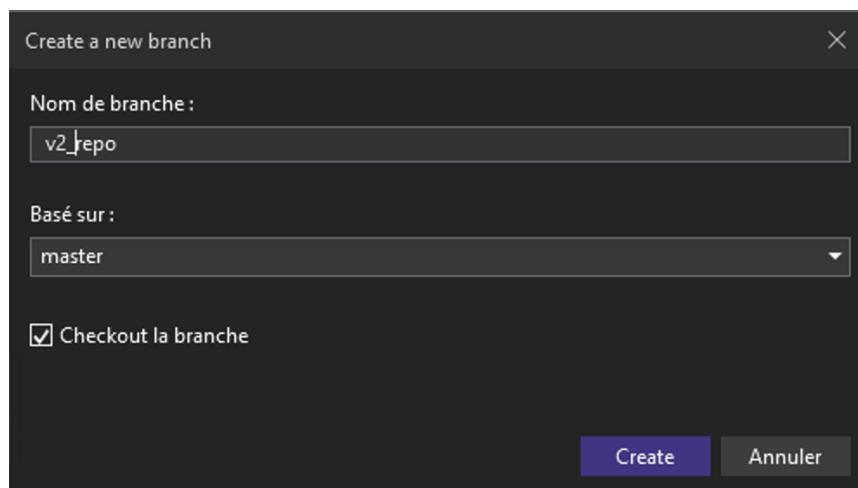
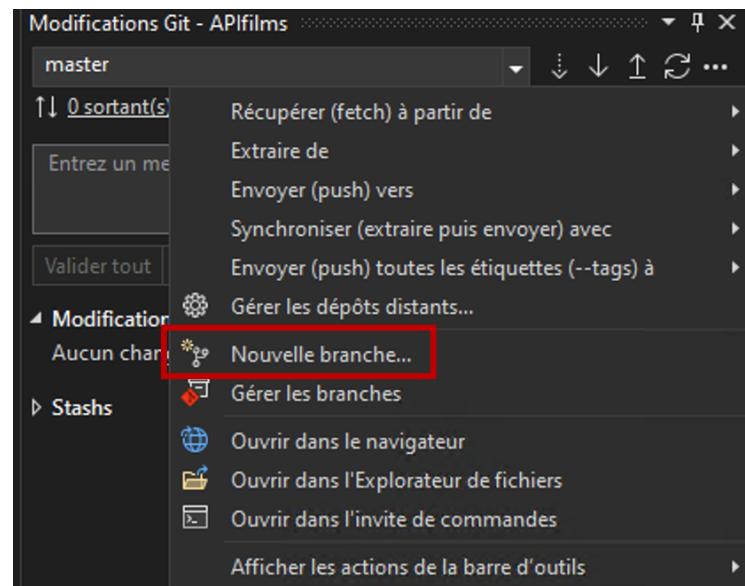
2. Appliquer le design pattern Repository (Dépôt). Ce pattern a été spécifié par Martin Fowler dans son (excellent) ouvrage « Patterns of Enterprise Application Architecture » : <https://martinfowler.com/books/eaa.html>. Principes de base de ce pattern : <https://martinfowler.com/eaaCatalog/repository.html>

Ce pattern fait partie d'un pattern plus global, nommé *Domain-Driven design pattern*.

Nous appliquerons le pattern Repository, dans sa version simple (1 seul repository).

2. Création d'une branche Git

Nous allons d'abord créer une branche Git sur laquelle appliquer nos modifications afin de ne pas modifier le code qui fonctionne.



3. Mise en place du pattern Repository

Les principes et la mise en place de ce pattern sont décrits ici :

<https://learn.microsoft.com/fr-fr/dotnet/standard/microservices-architecture/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-design>
<https://learn.microsoft.com/fr-fr/dotnet/standard/microservices-architecture/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-implementation-entity-framework-core>

Nous allons l'appliquer et un peu le simplifier car nous n'aurons qu'un seul dépôt et donc qu'une seule interface *Repository*.

3.1. Interface

Dans le dossier `Models`, ajouter un dossier nommé `Repository`. Y créer une nouvelle interface nommée `IDataRepository`.

Code de l'interface :

```
public interface IDataRepository<TEntity>
{
    ActionResult<IEnumerable<TEntity>> GetAll();
    ActionResult<TEntity> GetById(int id);
    ActionResult<TEntity> GetByString(string str);
    void Add(TEntity entity);
    void Update(TEntity entityToUpdate, TEntity entity);
    void Delete(TEntity entity);
}
```

Nous injecterons (injection de dépendance) plus tard cette interface dans notre contrôleur d'API. L'API communiquera avec le contexte de données (et donc la base) à l'aide de cette interface.

3.2. Classe concrète

Ensuite, créer une classe concrète qui implémente l'interface `IDataRepository`. Ajouter un nouveau dossier `Models` nommé `DataManager`. Créer ensuite une nouvelle classe `UtilisateurManager` :

```
public class UtilisateurManager : IDataRepository<Utilisateur>
{
    readonly FilmRatingsDbContext? filmsDbContext;

    public UtilisateurManager() { }

    public UtilisateurManager(FilmRatingsDbContext context)
    {
        filmsDbContext = context;
    }

    public ActionResult<IEnumerable<Utilisateur>> GetAll()
    {
        return filmsDbContext.Utilisateurs.ToList();
    }

    public ActionResult<Utilisateur> GetById(int id)
    {
        return filmsDbContext.Utilisateurs.FirstOrDefault(u => u.UtilisateurId == id);
    }

    public ActionResult<Utilisateur> GetByString(string mail)
    {
        return filmsDbContext.Utilisateurs.FirstOrDefault(u => u.Mail.ToUpper() == mail.ToUpper());
    }

    public void Add(Utilisateur entity)
    {
        filmsDbContext.Utilisateurs.Add(entity);
        filmsDbContext.SaveChanges();
    }

    public void Update(Utilisateur utilisateur, Utilisateur entity)
    {
        filmsDbContext.Entry(utilisateur).State = EntityState.Modified;
        utilisateur.UtilisateurId = entity.UtilisateurId;
        utilisateur.Nom = entity.Nom;
        utilisateur.Prenom = entity.Prenom;
        utilisateur.Mail = entity.Mail;
        utilisateur.Rue = entity.Rue;
        utilisateur.CodePostal = entity.CodePostal;
        utilisateur.Ville = entity.Ville;
        utilisateur.Pays = entity.Pays;
        utilisateur.Latitude = entity.Latitude;
        utilisateur.Longitude = entity.Longitude;
        utilisateur.Pwd = entity.Pwd;
        utilisateur.Mobile = entity.Mobile;
        utilisateur.NotesUtilisateur = entity.NotesUtilisateur;
        filmsDbContext.SaveChanges();
    }

    public void Delete(Utilisateur utilisateur)
    {
        filmsDbContext.Utilisateurs.Remove(utilisateur);
        filmsDbContext.SaveChanges();
    }
}
```

La classe `UtilisateurManager` gère toutes les opérations de base de données liées à l'`Utilisateur`. Le but de cette classe est de séparer la logique des opérations de données réelles du contrôleur de l'API.

Cette classe utilise les méthodes suivantes pour prendre en charge les opérations CRUD :

- `GetAll()` - Obtient tous les utilisateurs de la base de données.
- `GetById()` - Obtient un utilisateur spécifique de la base de données en transmettant un ID.
- `GetByEmail()` - Obtient un utilisateur spécifique de la base de données en transmettant un string (ici, un email).
- `Add()` - Crée un nouvel utilisateur dans la base de données.
- `Update()` - Met à jour un utilisateur spécifique dans la base de données.
- `Delete()` - Supprime un utilisateur spécifique de la base de données en fonction de l'ID.

Maintenant que notre Data Manager est configuré, il reste à modifier le contrôleur API et les points de terminaison pour la gestion des opérations CRUD.

3.3. Modification du contrôleur

Pour le moment, nous allons laisser les `async`, mais supprimer les `await` dans les actions. Nous n'aurons donc pas vraiment de méthodes asynchrones.

On n'instancie plus un objet `FilmRatingsDbContext` puisque l'on passe maintenant par la classe concrète de Repository. Modifier les parties suivantes en gras :

```
[Route("api/[controller]")]
[ApiController]
public class UtilisateursController : ControllerBase
{
    private readonly UtilisateurManager utilisateurManager;
    //private readonly FilmRatingsDbContext _context;

    public UtilisateursController(UtilisateurManager userManager)
    {
        utilisateurManager = userManager;
    }

    // GET: api/Utilisateurs
    [HttpGet]
    public async Task<ActionResult<IEnumerable<Utilisateur>>> GetUtilisateurs()
    {
        return utilisateurManager.GetAll();
    }

    // GET: api/Utilisateurs/{id}
    [HttpGet("{id}")]
    [ActionName("GetById")]
    [ProducesResponseType(StatusCodes.Status200OK)]
    [ProducesResponseType(StatusCodes.Status404NotFound)]
    public async Task<ActionResult<Utilisateur>> GetUtilisateurById(int id)
    {
        var utilisateur = utilisateurManager.GetById(id);
        //var utilisateur = await _context.Utilisateurs.FindAsync(id);

        if (utilisateur == null)
        {
            return NotFound();
        }

        return utilisateur;
    }

    // GET: api/Utilisateurs/toto@titi.fr
    [HttpGet]
    [Route("[action]/{email}")]
    [ActionName("GetByEmail")]
    [ProducesResponseType(StatusCodes.Status200OK)]
    [ProducesResponseType(StatusCodes.Status404NotFound)]
    public async Task<ActionResult<Utilisateur>> GetUtilisateurByEmail(string email)
```

```

{
    var utilisateur = utilisateurManager.GetByString(email);
    //var utilisateur = await _context.Utilisateurs.FirstOrDefaultAsync(e => e.Mail.ToUpper() == email.ToUpper());

    if (utilisateur == null)
    {
        return NotFound();
    }

    return utilisateur;
}

// PUT: api/Utilisateurs/5
// To protect from overposting attacks, see https://go.microsoft.com/fwlink/?linkid=2123754
[HttpPut("{id}")]
[ProducesResponseType(StatusCodes.Status204NoContent)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
public async Task<IActionResult> PutUtilisateur(int id, Utilisateur utilisateur)
{
    if (id != utilisateur.UtilisateurId)
    {
        return BadRequest();
    }

    var userToUpdate = utilisateurManager.GetById(id);

    if (userToUpdate == null)
    {
        return NotFound();
    }
    else
    {
        utilisateurManager.Update(userToUpdate.Value, utilisateur);
        return NoContent();
    }
}

// POST: api/Utilisateurs
// To protect from overposting attacks, see https://go.microsoft.com/fwlink/?linkid=2123754
[HttpPost]
[ProducesResponseType(StatusCodes.Status201Created)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
public async Task<ActionResult<Utilisateur>> PostUtilisateur(Utilisateur utilisateur)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

    utilisateurManager.Add(utilisateur);

    return CreatedAtAction("GetById", new { id = utilisateur.UtilisateurId }, utilisateur); // GetById : nom de l'action
}

// DELETE: api/Utilisateurs/5
[HttpDelete("{id}")]
[ProducesResponseType(StatusCodes.Status204NoContent)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
public async Task<IActionResult> DeleteUtilisateur(int id)
{
    var utilisateur = utilisateurManager.GetById(id);
    if (utilisateur == null)
    {
        return NotFound();
    }
}

```

```

    }

    utilisateurManager.Delete(utilisateur.Value);
    return NoContent();
}

//private bool UtilisateurExists(int id)
//{
//    return _context.Utilisateurs.Any(e => e.UtilisateurId == id);
//}
}

```

On voit que nos méthodes ne sont pas réellement asynchrones (warnings) :

Liste d'erreurs			
Solution complète	0 Erreurs	16 Avertissements	0 Messages
Code	Description	Projet	Fichier
CS1998	Cette méthode async n'a pas d'opérateur 'await' et elle s'exécutera de façon synchrone. Utilisez l'opérateur 'await' pour attendre les appels d'API non bloquants ou 'await Task.Run(...)' pour effectuer un travail utilisant le processeur sur un thread d'arrière-plan.	APIseries	UtilisateursController.cs
CS1998	Cette méthode async n'a pas d'opérateur 'await' et elle s'exécutera de façon synchrone. Utilisez l'opérateur 'await' pour attendre les appels d'API non bloquants ou 'await Task.Run(...)' pour effectuer un travail utilisant le processeur sur un thread d'arrière-plan.	APIseries	UtilisateursController.cs
CS1998	Cette méthode async n'a pas d'opérateur 'await' et elle s'exécutera de façon synchrone. Utilisez l'opérateur 'await' pour attendre les appels d'API non bloquants ou 'await Task.Run(...)' pour effectuer un travail utilisant le processeur sur un thread d'arrière-plan.	APIseries	UtilisateursController.cs
CS1998	Cette méthode async n'a pas d'opérateur 'await' et elle s'exécutera de façon synchrone. Utilisez l'opérateur 'await' pour attendre les appels d'API non bloquants ou 'await Task.Run(...)' pour effectuer un travail utilisant le processeur sur un thread d'arrière-plan.	APIseries	UtilisateursController.cs
CS1998	Cette méthode async n'a pas d'opérateur 'await' et elle s'exécutera de façon synchrone. Utilisez l'opérateur 'await' pour attendre les appels d'API non bloquants ou 'await Task.Run(...)' pour effectuer un travail utilisant le processeur sur un thread d'arrière-plan.	APIseries	UtilisateursController.cs

Exécuter le WS.

Vous obtiendrez l'erreur classique de dépendance d'injection.

An unhandled exception occurred while processing the request.

InvalidOperationException: Unable to resolve service for type 'APIseries.Models.DataManager.UtilisateurManager' while attempting to activate 'APIseries.Controllers.UtilisateursController'.

Microsoft.Extensions.DependencyInjection.ActivatorUtilities.GetService(IServiceProvider sp, Type type, Type requiredBy, bool isDefaultParameterRequired)

Stack Query Cookies Headers Routing

InvalidOperationException: Unable to resolve service for type 'APIseries.Models.DataManager.UtilisateurManager' while attempting to activate 'APIseries.Controllers.UtilisateursController'.

Microsoft.Extensions.DependencyInjection.ActivatorUtilities.GetService(IServiceProvider sp, Type type, Type requiredBy, bool isDefaultParameterRequired)
lambda_method9(Closure , IServiceProvider , object[])
Microsoft.AspNetCore.Mvc.Controllers.ControllerActivatorProvider+<>c__DisplayClass7_0.<CreateActivator>b_0(ControllerContext controllerContext)
Microsoft.AspNetCore.Mvc.Controllers.ControllerFactoryProvider+<>c__DisplayClass6_0.<CreateControllerFactory>g__CreateController|0(ControllerContext controllerContext)
Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.Next(ref State next, ref Scope scope, ref object state, ref bool isCompleted)
Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.InvokeInnerFilterAsync()
Microsoft.AspNetCore.Mvc.Infrastructure.ResourceInvoker.<InvokeFilterPipelineAsync>g_Awaited|20_0(ResourceInvoker invoker, Task lastTask, State next, Scope scope, object state, bool isCompleted)
Microsoft.AspNetCore.Mvc.Infrastructure.ResourceInvoker.<InvokeAsync>g_Awaited|17_0(ResourceInvoker invoker, Task task, IDisposable scope)
Microsoft.AspNetCore.Mvc.Infrastructure.ResourceInvoker.<InvokeAsync>g_Awaited|17_0(ResourceInvoker invoker, Task task, IDisposable scope)
Microsoft.AspNetCore.Routing.EndpointMiddleware.<Invoke>g_AwaitRequestTask|6_0(Endpoint endpoint, Task requestTask, ILogger logger)
Microsoft.AspNetCore.Authorization.AuthorizationMiddleware.Invoke(HttpContext context)
Swashbuckle.AspNetCore.SwaggerUI.SwaggerUIMiddleware.Invoke(HttpContext httpContext)
Swashbuckle.AspNetCore.Swagger.SwaggerMiddleware.Invoke(HttpContext httpContext, ISwaggerProvider swaggerProvider)
Microsoft.AspNetCore.Diagnostics.DeveloperExceptionPageMiddleware.Invoke(HttpContext context)

Show raw exception details

3.4. Injection de dépendance

Dans notre contrôleur, nous avons utilisé un objet de type `UtilisateurManager`.

Cependant, le code précédent n'est pas très propre car nous avons lié notre contrôleur à une implémentation spécifique de `IDataRepository`.

Pour remédier à cela, nous allons utiliser l'interface. Le code du constructeur du contrôleur devient :

```
private readonly IDataRepository<Utilisateur> dataRepository;
```

```
public UtilisateursController(IDataRepository<Utilisateur> dataRepo)
```

```
{
    dataRepository = dataRepo;
}
```

Il faut également remplacer partout `utilisateurManager` par `dataRepository`.

Exécuter l'API. Vous obtenez toujours l'erreur d'injection de dépendance.

Aucune interface `IDataRepository` n'est en effet envoyée au constructeur paramétré du contrôleur. C'est le mécanisme d'injection de dépendance qui va en avoir la charge.

La configuration du repository à l'aide de l'injection de dépendance se fait dans la méthode `Main` du fichier `Program.cs` :

```
public static void Main(string[] args)
{
    var builder = WebApplication.CreateBuilder(args);

    ...
    builder.Services.AddScoped<IDataRepository<Utilisateur>, UtilisateurManager>();

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

Ici, nous avons lié l'interface à sa classe concrète.

Méthode `AddScoped` :

<https://learn.microsoft.com/fr-fr/dotnet/api/microsoft.extensions.dependencyinjection.servicecollectionextensions.addscoped>
<https://learn.microsoft.com/fr-fr/aspnet/core/fundamentals/dependency-injection>

On peut aussi utiliser `AddTransient` ou `AddSingleton` pour lier une interface à sa classe concrète, mais EF requiert `AddScoped`. Différence entre les 3 mécanismes :

<https://www.developpez.com/actu/154398/Apprendre-l-injection-de-dependances-avec-ASP-NET-Core-un-billet-d-Hinault-Romaric/>
<https://stackoverflow.com/questions/38138100/what-is-the-difference-between-services-addtransient-service-addscoped-and-serv>

Exécuter l'API.

4. Tests unitaires

Les tests unitaires ne sont plus fonctionnels.

Ajouter le code suivant :

```
private UtilisateursController controller;

private FilmRatingsDbContext context;

private IDataRepository<Utilisateur> dataRepository;

public UtilisateursControllerTests()
{
    var builder = new DbContextOptionsBuilder<FilmRatingsDbContext>().UseNpgsql(...);
    context = new FilmRatingsDbContext(builder.Options);
    dataRepository = new UtilisateurManager(context);
    controller = new UtilisateursController(dataRepository);
}
```

Les appels au contrôleur deviennent : `UtilisateurController controller = new UtilisateurController(dataRepository);`

On remarque que pour les tests, on utilise toujours le contexte qui permet de récupérer les données provenant directement de la base et on compare ces données avec les données issues de l'appel aux actions du WS. Ainsi, on teste à la fois l'API et le Repository. Par contre, on reste dépendant de la couche de données. On pourrait modifier les tests pour comparer les données de l'API avec celles du repository.

5. Asynchronisme

Il est nécessaire de remettre en place l'asynchronisme.

Exemple pour le `Get (email)` :

- Modifier l'interface `IDataRepository` :


```
ActionResult< TEntity > GetByStringAsync(string str); -> Task< ActionResult< TEntity > > GetByStringAsync(string str);
```

- Modifier la méthode `GetString` de `UtilisateurManager` :

```
public async Task<ActionResult<Utilisateur>> GetStringAsync(string mail)
{
    return await filmsDbContext.Utilisateurs.FirstOrDefaultAsync(u => u.Mail.ToUpper() == mail.ToUpper());
}
```
 - Modifier la méthode `GetUtilisateurByEmail` du contrôleur :

```
public async Task<ActionResult<Utilisateur>> GetUtilisateurByEmail(string email)
{
    var utilisateur = await dataRepository.GetStringAsync(email);

    if (utilisateur == null)
    {
        return NotFound();
    }
    return utilisateur;
}
```
- Vous devriez avoir un avertissement lié à l'asynchronisme de moins maintenant...

Pour les méthodes `Add`, `Update` et `Delete` de `IDateRepository` et `UtilisateurManager`, il faut remplacer `void` par `Task`. Exemple :

`Task AddAsync(TEntity entity);`

```
public async Task AddAsync(Utilisateur entity)
{
    await filmsDbContext.Utilisateurs.AddAsync(entity);
    await filmsDbContext.SaveChangesAsync();
}
```

6. Fusion des branches

Exécuter l'application. La tester.

Vérifier l'exécution des tests unitaires.

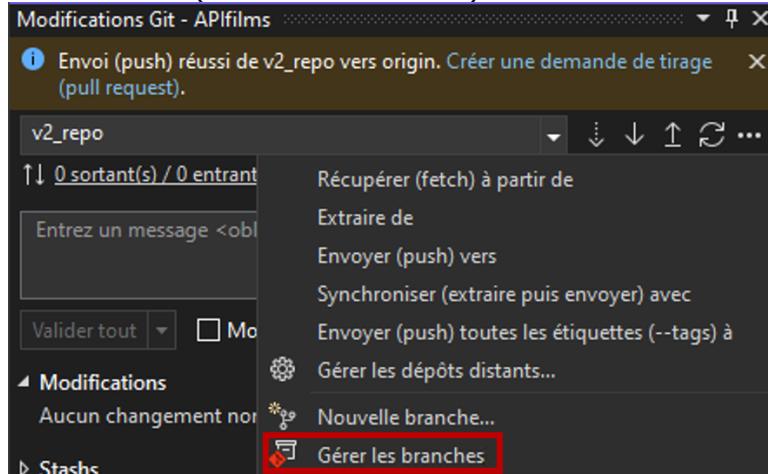
Valider les modifications et synchroniser les branches.

The screenshot shows a GitHub repository page for 'VincentCOUTURIER-USMB / APIfilms'. The repository has 2 branches and 0 tags. A message at the top indicates 'v2_repo had recent pushes 1 minute ago'. Below this, it says 'This branch is 1 commit ahead of master.' The commit history for the 'vcout' branch shows 8 commits:

Commit	Message	Time
.github/workflows	publish	2 weeks ago
APIfilms	Repository	1 minute ago
APIfilmsTests	Repository	1 minute ago
.gitattributes	Ajouter .gitattributes et .gitignore	2 weeks ago
.gitignore	Ajouter .gitattributes et .gitignore	2 weeks ago
APIfilms.lutconfig	API avec tests (sans PUT)	last week
APIfilms.sln	API avec tests (sans PUT)	last week
CodeMap1.dgml	API complete	2 weeks ago

At the bottom, there's a note to 'Add a README with an overview of your project.' and a 'Add a README' button.

Vous pouvez comparer les 2 branches (locales ou distantes).



The screenshot shows the 'Dépôt Git - APIfilms' interface. On the left, the repository structure is displayed with branches: master, v2_repo, and remotes/origin/master. A context menu is open over the 'v2_repo' branch, listing options like 'Extrire', 'Validation du conseil d'extraction (--detach)', 'Nouvelle branche locale de...', 'Fusionner 'origin/master' dans 'v2_repo'', 'Rebaser 'v2_repo' sur 'origin/master'', 'Réinitialiser', 'Cherry-Pick', 'Supprimer', 'Voir l'historique', 'Comparer 'origin/master' avec 'v2_repo'' (which is highlighted), 'Récupérer (fetch)', 'Tirer (pull)', 'Envoyer (push)', and 'Synchroniser (extraire puis envoyer)'. On the right, the commit history for the master branch is shown, with the last few commits being merges from 'origin/master'.

The screenshot shows the 'Comparaison d...avec v2_repo' window. It displays two versions of the 'UtilisateursController.cs' file side-by-side. The left pane shows the file from the 'origin/master' branch, and the right pane shows it from the 'v2_repo' branch. The 'Comparer' tab is active, showing the base branch as 'origin/master' and the compare branch as 'v2_repo'. The 'Modifications' section on the right lists the changes made in the 'v2_repo' branch, including additions and deletions of code blocks. The status bar at the bottom indicates 17 modification(s), -29, and +25.

Il y a ensuite 2 possibilités :

- Vous pouvez directement fusionner la branche v2_repo dans master si vous êtes le développeur principal (ou le seul développeur).
- OU vous pouvez créer une requête de tirage pour demander au développeur principal de fusionner les 2 branches.

Cliquer sur « Créer une requête de tirage » sur origin/v2_repo :

The screenshot shows a Git commit history for the 'v2_repo' branch. A context menu is open over the 'v2_repo' branch, with the 'Créer une requête de tirage' option highlighted.

Auteur	Date	ID
vcout	19/02/202...	a9a8cef5
master	08/02/202...	f05312f6
vcout	08/02/202...	979e81db
vcout	08/02/202...	d07934d7
vcout	08/02/202...	f8ff4850
vcout	08/02/202...	2e4138b9
vcout	07/02/202...	769099f5
vcout	07/02/202...	af2a426f

The screenshot shows the GitHub pull request creation interface. The 'base' dropdown is set to 'master' and the 'compare' dropdown is set to 'v2_repo'. The 'Repository' text area contains the message 'J'ai appliqué le pattern Repository pour améliorer le code.'

Requête de tirage à valider (ou non) par le développeur principal :

The screenshot shows a GitHub repository page for 'VincentCOUTURIER-USMB / APIfilms'. A pull request from 'v2_repo' to 'master' has been merged. The commit message is: 'J'ai appliqué le pattern Repository pour améliorer le code.' The pull request summary indicates 'Require approval from specific reviewers before merging' and 'This branch has no conflicts with the base branch'. The 'Merge pull request' button is visible. Below the pull request, there is a comment section with 'Write' and 'Preview' tabs, and a 'Comment' button.

Vous pouvez alors :

- Demander des précisions en ajoutant un commentaire qui sera reçu par le développeur ayant soumis la requête de tirage.
- OU fermer la requête de tirage sans prendre en compte la demande de merge (« Close pull request »)
- OU valider la requête de tirage (« Merge pull request ») et ainsi valider la fusion. Vous pourrez ensuite supprimer la branche.

Valider la requête de tirage :

The screenshot shows a GitHub repository page for 'VincentCOUTURIER-USMB / APIfilms'. A pull request has been merged into the 'master' branch from the 'v2_repo' branch. The merge commit message is: 'J'ai appliqué le pattern Repository pour améliorer le code.' The commit hash is 'a9a8cef'. A comment from the user 'VincentCOUTURIER-USMB' says: 'Pull request successfully merged and closed. You're all set—the v2_repo branch can be safely deleted.' There is a 'Delete branch' button next to this comment. Below the comments, there is a comment form with a 'Comment' button.

Puis supprimer la branche (« Delete branch »).

Résultat :

The screenshot shows the same GitHub repository page after the 'v2_repo' branch has been deleted. The repository now has 1 branch and 0 tags. The commit history shows the merge commit and the 9 commits from the 'v2_repo' branch. A message at the bottom encourages adding a README file.

File	Action	Time
.github/workflows	publish	2 weeks ago
APIfilms	Repository	22 minutes ago
APIfilmsTests	Repository	22 minutes ago
.gitattributes	Ajouter .gitattributes et .gitignore	2 weeks ago
.gitignore	Ajouter .gitattributes et .gitignore	2 weeks ago
APIfilms.lutconfig	API avec tests (sans PUT)	last week
APIfilms.sln	API avec tests (sans PUT)	last week
CodeMap1.dgml	API complete	2 weeks ago