Une image contenant Police, logo, Graphique, conception

Description générée automatiquement

DÉPARTEMENT DE MATHÉMATIQUES, D'INFORMATIQUE ET DE GÉNIE

**INF27523 – Technologies du commerce électronique**

**Hiver 2024**

**Travail pratique #2**

**Professeur: Yacine Yaddaden**

**Présenté par**

**Guillaume Terrière**

**Sorenza Morel**

[Introduction 3](#_Toc165240675)

[Technologies utilisées 3](#_Toc165240676)

[Architecture et composition du site web 3](#_Toc165240677)

[Structure du dossier 4](#_Toc165240678)

[Description détaillée 4](#_Toc165240679)

[OrderManagement 4](#_Toc165240680)

[Controller 4](#_Toc165240681)

[Model 4](#_Toc165240682)

[Data 5](#_Toc165240683)

[PayementProcessing 5](#_Toc165240684)

[Controller 5](#_Toc165240685)

[ProductCatalog 5](#_Toc165240686)

[Controller 5](#_Toc165240687)

[UserManagement 6](#_Toc165240688)

[Controller 6](#_Toc165240689)

[ShoppingCart 7](#_Toc165240690)

[Controller 7](#_Toc165240691)

[Appréciation sur ce projet 7](#_Toc165240692)

[Difficultés rencontrées 7](#_Toc165240693)

[Conclusion 7](#_Toc165240694)

# Introduction

Dans le cadre du cours de technologies du commerce électronique, ce rapport détaille la mise en œuvre d'une architecture en microservices pour l’API d’une boutique en ligne de revente de jeux vidéo, en utilisant ASP.NET Core. Ce projet s'inscrit dans le deuxième travail pratique du cours et vise à explorer divers aspects des API REST, de l'authentification par jeton, de l'utilisation de Swagger pour la documentation du back-end, ainsi que de l'intégration d'une passerelle avec Ocelot et le paiement électronique à l'aide de Stripe.

# Technologies utilisées

Pour la réalisation de ce projet, nous avons utilisé un ensemble de technologies spécifiques pour chaque aspect du développement. Tout d'abord, le langage de programmation principal était le C#, avec une approche orientée objet pour la conception des différents services et fonctionnalités. Pour la gestion des données, nous avons opté pour Entity Framework Core, en utilisant les packages `Microsoft.EntityFrameworkCore.SqlServer`, `Microsoft.EntityFrameworkCore.Tools` et `Microsoft.EntityFrameworkCore.Design`. En ce qui concerne l'authentification des utilisateurs, nous avons mis en œuvre une authentification par jeton en utilisant le packages `Microsoft.AspNetCore.Authentication.JwtBearer`. Pour la documentation du back-end, nous avons intégré Swagger à notre projet, en utilisant les packages `Swashbuckle.AspNetCore` et `Microsoft.AspNetCore.Mvc.NewtonsoftJson`. La passerelle d'API a été réalisée avec Ocelot, un framework qui nous a permis de gérer les requêtes et les rediriger vers les services appropriés. Nous avons utilisé les packages `Ocelot` et `MMLib.SwaggerForOcelot` pour la configuration et la documentation de cette passerelle. En ce qui concerne l'intégration du paiement électronique, nous avons utilisé la bibliothèque `Stripe.net` pour communiquer avec l'API de Stripe et gérer les transactions de paiement. Pour le développement, nous avons travaillé dans l'environnement de développement intégré Microsoft Visual Studio Community, qui offre un large éventail d'outils et de fonctionnalités pour le développement d'applications .NET. Pour tester les requêtes et les interactions avec les différentes API, nous avons utilisé l'outil POSTMAN. Nous avons conteneurisé tous les services avec Docker, et créer un script de lancement avec docker compose. Enfin, pour la gestion de version et le partage du code, nous avons utilisé Git, avec un dépôt hébergé sur GitHub.

# Structure du dossier

Le projet est organisé en différents services, chacun étant un projet ASP.NET Web API distinct. Chaque service contient les éléments suivants : des contrôleurs pour gérer les requêtes HTTP, des modèles de données pour représenter les entités métier et un contexte de données pour la persistance des données.

# Description détaillée

## ApiGateway

ApiGateway est le service passerelle entre tous les autres services. C’est là que Ocelot est configuré dans le dossier Routes, ainsi que Swagger avec tout les services.

## OrderManagement

OrderManagement est un service du système qui gère les commandes des utilisateurs. Il fournit des fonctionnalités telles que la création, la mise à jour, la suppression et la récupération des commandes. Les utilisateurs peuvent passer des commandes pour acheter des produits via cet API.

### Controller

Ce code définit un contrôleur ASP.NET Core pour la gestion des commandes dans le système OrderManagement. Le contrôleur fournit des points de terminaison pour effectuer des opérations CRUD (Create, Read, Update, Delete) sur les commandes. Les opérations incluent la récupération de toutes les commandes, la récupération d'une commande spécifique par son identifiant, la création d'une nouvelle commande, la mise à jour d'une commande existante et la suppression d'une commande. Chaque méthode du contrôleur correspond à une action HTTP (GET, POST, PUT, DELETE) avec des routes spécifiques pour accéder aux fonctionnalités associées. Les méthodes effectuent des opérations sur la base de données via le contexte de données OrderContext.

### Model

Le modèle `Order` dans le système OrderManagement représente une commande passée dans le système. Chaque instance de la classe `Order` contient des informations cruciales sur une commande spécifique. Cela comprend un identifiant unique `Id` pour chaque commande, permettant une identification sans ambiguïté dans le système. De plus, la propriété `ProductName` enregistre le nom du produit commandé, tandis que `Quantity` indique la quantité de ce produit commandée. Enfin, la propriété `TotalPrice` stocke le prix total de la commande, calculé en multipliant la quantité de produit par son prix unitaire. Ce modèle de données est essentiel pour la gestion des commandes et des transactions dans le système OrderManagement. Tout les autres dossiers Model fonctionne de la même façon.

### Data

La classe `OrderContext` dans le namespace OrderManagement.Data est une classe dérivée de DbContext, ce qui la rend responsable de la gestion de la connexion à la base de données et de la coordination des opérations de lecture et d'écriture des données liées aux commandes. Elle contient un DbSet<Order> nommé Orders, qui représente la table des commandes dans la base de données. Cette classe joue un rôle crucial dans l'interaction entre le système OrderManagement et la base de données, en permettant le stockage, la récupération, la mise à jour et la suppression des commandes dans le système. Tout les autres dossiers Data du projet fonctionne de la même façon.

## PayementProcessing

### Controller

Ce code représente un contrôleur ASP.NET Core pour gérer les paiements via l'API Stripe. Le contrôleur est responsable de la création et du traitement des transactions de paiement. Le contrôleur est annoté avec `[ApiController]` et `[Route("/api/payements")]`, ce qui indique qu'il fait partie de l'API et définit les endpoints pour les paiements. La méthode `CreateCharge` est annotée avec `[HttpPost("charge")]`, ce qui signifie qu'elle répond aux requêtes HTTP POST sur l'URL "/api/payements/charge". Elle prend en paramètre un objet de type `PaymentRequest`, qui contient les informations nécessaires pour effectuer le paiement, telles que le montant et le jeton de paiement. À l'intérieur de la méthode `CreateCharge`, un objet `ChargeCreateOptions` est créé en utilisant les informations fournies dans la demande. Ces options comprennent le montant du paiement, la devise, la source (le jeton de paiement Stripe) et une description optionnelle de la transaction. Ensuite, un objet `ChargeService` est utilisé pour créer la charge en appelant la méthode `Create` avec les options spécifiées. Si la charge est créée avec succès, elle est renvoyée en tant que réponse HTTP OK avec le détail de la charge. Sinon, si une exception de type `StripeException` est levée, une réponse HTTP BadRequest est renvoyée avec un message d'erreur provenant de l'API Stripe. Enfin, la classe `PaymentRequest` est définie pour structurer les données requises pour effectuer un paiement, notamment le montant et le jeton de paiement.

## ProductCatalog

### Controller

Ce code représente un contrôleur ASP.NET Core pour gérer les produits dans un catalogue. Le contrôleur définit les endpoints pour effectuer des opérations CRUD (Create, Read, Update, Delete) sur les produits. Le contrôleur est annoté avec `[ApiController]` et [Route("api/products")]`, ce qui indique qu'il fait partie de l'API et définit les routes pour les produits. Les méthodes HTTP GET, POST, PUT et DELETE sont utilisées pour récupérer, créer, mettre à jour et supprimer des produits respectivement. La méthode `GetProducts` répond aux requêtes HTTP GET sur l'URL "/api/products" en renvoyant la liste de tous les produits disponibles. La méthode `GetProduct` répond aux requêtes HTTP GET sur l'URL "/api/products/{id}" en renvoyant les détails d'un produit spécifique identifié par son ID. La méthode `CreateProduct` répond aux requêtes HTTP POST sur l'URL "/api/products" en ajoutant un nouveau produit au catalogue. Les détails du produit sont fournis dans le corps de la requête HTTP. La méthode `UpdateProduct` répond aux requêtes HTTP PUT sur l'URL "/api/products/{id}" en mettant à jour les détails d'un produit spécifique identifié par son ID. Les nouveaux détails du produit sont fournis dans le corps de la requête HTTP. La méthode `DeleteProduct` répond aux requêtes HTTP DELETE sur l'URL "/api/products/{id}" en supprimant un produit spécifique identifié par son ID. Chaque méthode utilise le contexte de base de données `ProductContext` pour interagir avec la base de données et effectuer les opérations CRUD nécessaires.

## UserManagement

### Controller

Ce code représente un contrôleur ASP.NET Core pour la gestion des utilisateurs dans un système de gestion des utilisateurs. Le contrôleur définit les endpoints pour effectuer différentes opérations liées aux utilisateurs, telles que l'authentification, la création, la récupération, la mise à jour et la suppression. Le contrôleur est annoté avec `[ApiController]` et `[Route("/api/users")]`, ce qui indique qu'il fait partie de l'API et définit les routes pour les utilisateurs. Le constructeur du contrôleur prend en paramètres un contexte de base de données `UserContext` et une interface de configuration `IConfiguration`. Ces dépendances sont injectées par le framework ASP.NET Core. La méthode `Authenticate` est annotée avec `[HttpPost("authenticate")]` et `[AllowAnonymous]`, ce qui signifie qu'elle répond aux requêtes HTTP POST sur l'URL "/api/users/authenticate" et qu'elle est accessible sans authentification. Cette méthode vérifie les informations d'identification fournies par l'utilisateur et génère un jeton JWT valide s'il est authentifié avec succès. Les méthodes `CreateUser`, `GetAllUsers`, `GetUser`, `UpdateUser` et `DeleteUser` gèrent respectivement la création, la récupération, la récupération d'un utilisateur spécifique par son ID, la mise à jour et la suppression des utilisateurs dans la base de données. La méthode `GenerateJwtToken` est une méthode privée utilisée pour générer un jeton JWT (JSON Web Token) lors de l'authentification réussie d'un utilisateur. Ce jeton contient des informations telles que l'ID de l'utilisateur, le nom d'utilisateur, l'e-mail (en supposant que le nom d'utilisateur est l'e-mail) et d'autres revendications nécessaires pour l'authentification et l'autorisation. Dans l'ensemble, ce contrôleur facilite la gestion complète des utilisateurs dans un système, y compris l'authentification sécurisée à l'aide de jetons JWT et les opérations CRUD de base sur les utilisateurs.

## ShoppingCart

### Controller

Ce code représente un contrôleur ASP.NET Core pour la gestion du panier d'achats dans un système de commerce électronique. Le contrôleur définit les endpoints pour effectuer différentes opérations liées au panier d'achats, telles que la création, la récupération, la mise à jour et la suppression des articles du panier. Le contrôleur est annoté avec `[ApiController]` et `[Route("api/cart")]`, ce qui indique qu'il fait partie de l'API et définit les routes pour le panier d'achats. Le constructeur du contrôleur prend en paramètre un contexte de base de données `ShoppingCartContext`, qui est injecté par le framework ASP.NET Core. Les méthodes HTTP GET, POST, PUT et DELETE sont utilisées pour implémenter les opérations CRUD (Create, Read, Update, Delete) sur les éléments du panier d'achats. Chaque méthode correspond à une opération spécifique et utilise les fonctionnalités asynchrones de C# et Entity Framework Core pour effectuer des opérations de base de données de manière efficace. Enfin, la méthode privée `CartItemExists` est utilisée pour vérifier si un article du panier d'achats existe déjà dans la base de données en fonction de son ID. En résumé, ce contrôleur facilite la gestion complète du panier d'achats dans un système de commerce électronique, permettant aux utilisateurs d'ajouter, de consulter, de modifier et de supprimer des articles de leur panier.

# Appréciation sur ce projet

Réaliser ce projet a été une expérience enrichissante. Malgré les défis rencontrés en cours de route, nous avons réussi à relever la plupart d’entre eux et à répondre aux exigences fixées pour ce projet. Nous avons pu mettre en place les fonctionnalités essentielles telles que la gestion des utilisateurs, l'intégration de l'API et la manipulation des données des utilisateurs. Bien que certaines étapes aient été plus exigeantes que d'autres, nous avons apprécié l'opportunité d'apprendre et de progresser tout au long de ce projet.

# Difficultés rencontrées

La difficulté rencontrée a été lors de l’implémentation de la base de données, en effet, l’un de nous deux développais sur Mac avec l’architecture de processeur M1, et le package Microsoft. EntityFrameworkCore.SqlServer n’était pas compatible, malgré de nombreux essais, la base de données n’est pas liée aux micro-services bien qu’elle soit lancée et fonctionnelle dans docker.

# Conclusion

Dans l'ensemble, la réalisation de ce projet a été une expérience stimulante et formatrice. Malgré les défis rencontrés en cours de route, notamment la mise en place d'une architecture en microservices et l'intégration de diverses technologies telles que l'authentification par jeton, la passerelle ocelot et le paiement électronique avec Stripe, nous sommes fiers d'avoir relevé ces défis avec succès, malgré le problème avec la base de données. Ce projet nous a permis d'approfondir nos compétences en développement d’api et en gestion de projet, tout en nous offrant une perspective concrète sur les technologies du commerce électronique. En conclusion, nous sommes satisfaits du résultat final et de ce que nous avons appris.