Titre : *Intégration d'une API REST avec une architecture en microservices pour la boutique en ligne pour la vente de produits en utilisant ASP.NET Core.*

Cours	Technologies du commerce électronique (INF27523)
Session	Hiver 2025
Contact	yacine_yaddaden@uqar.ca

Table des matières

1. Objectif du travail

L'objectif principal de ce deuxième travail pratique est de concevoir et développer la partie **back-end** de la plateforme de e-commerce développée dans le premier travail pratique, en appliquant les principes de l'**architecture en** *microservices*. Ce projet permettra aux étudiants de mettre en œuvre les concepts suivants :

- ☑ Création d'**API REST** à l'aide de **Microsoft ASP.NET Web API**
- ☑ Décomposition de l'application en plusieurs services indépendants
- ☑ Mise en place d'une *authentification* par *jeton* **JWT**
- ☑ Intégration d'un système de *paiement électronique* avec **Stripe**
- ☑ Utilisation de l'outil **Swagger** pour la documentation
- ☑ Création d'une *passerelle* API (API Gateway) avec **Ocelot**

Ce travail vient compléter le développement de la boutique en ligne en séparant le *front-end* du *back-end* selon les bonnes pratiques du développement web moderne.

2. Technologies à utiliser

Pour mener à bien ce projet, les technologies suivantes seront utilisées :

- ☑ **Environnement de développement et outils :**
  - → **Microsoft Visual Studio 2022 Community** (programmation).
  - → Git pour la gestion de version.
  - → GitHub pour héberger le dépôt du projet.
- ☑ **Langage et Frameworks :**
  - → **C#** pour le développement des services
  - → **ASP.NET Web API** pour la création des API REST
  - → **Entity Framework Core** pour la persistance des données
  - → **Ocelot** pour la passerelle d'API (API Gateway)
  - → **Stripe.net** pour l'intégration du paiement électronique.
- ☑ **Bibliothèques et packages requis :**
  - → Gestion des données :
    - **Microsoft.EntityFrameworkCore.SqlServer**
    - **Microsoft.EntityFrameworkCore.Tools**

- **Microsoft.EntityFrameworkCore.Design**

→ Authentification :
- **Microsoft.AspNetCore.Identity.EntityFrameworkCore**
- **Microsoft.AspNetCore.Authentication.JwtBearer**

→ Documentation API :
- **Swashbuckle.AspNetCore**
- **Microsoft.AspNetCore.Mvc.NewtonsoftJson**

→ Passerelle d'API :
- **Ocelot**
- **MMLib.SwaggerForOcelot**

☑ **Logiciel :**

→ **Postman** pour tester les API REST.

Il est recommandé de faire de la *gestion de version* avec l'outil **Git**. Il est recommandé également de créer un dépôt sur GitHub.

3. Description détaillée

Afin de réaliser le projet, il est nécessaire de réaliser ce qui suit :

3.1. Architecture en *microservices*

Le projet doit être divisé en **plusieurs services indépendants**, chacun dédié à une fonctionnalité spécifique de la plateforme e-commerce. Par exemple :

☑ **Service Produits** → gestion des articles et catégories
☑ **Service Utilisateurs** → gestion de l'inscription, connexion et profils
☑ **Service Commandes** → traitement et suivi des commandes
☑ **Service Panier** → gestion des articles ajoutés par les clients
☑ **Service Paiement** → gestion des transactions Stripe
☑ **Service Authentification** → génération de jetons JWT

Chaque service sera un **projet ASP.NET Web API distinct** avec son propre port et base de données.

3.2. Création d'une passerelle API (API Gateway)

Une **passerelle d'API** devra être mise en place à l'aide d'**Ocelot**, pour centraliser les requêtes des clients et les rediriger vers les services concernés.

☑ Configurer correctement le fichier **ocelot.json**
☑ S'assurer de l'interconnexion entre la passerelle et les *microservices*
☑ Ajouter **Swagger** unifié à travers **MMLib.SwaggerForOcelot**

3.3. Authentification par jeton (JWT)

Le service d'authentification doit :

☑ Gérer l'enregistrement et la connexion des utilisateurs
☑ Générer un *jeton* **JWT** sécurisé après l'authentification
☑ Permettre à **Swagger** de tester les appels protégés avec ce jeton

3.4. Documentation avec Swagger

Chaque service doit être documenté avec **Swagger**. Il est aussi demandé de :

☑ Configurer **Swagger UI** dans chaque projet
☑ Mettre en place une interface unifiée au niveau de la passerelle :
→ En utilisant **MMLib.SwaggerForOcelot**

3.5. Intégration du paiement électronique (Stripe)

Un service dédié au paiement doit être développé pour :

☑ Envoyer une requête **REST à l'API Stripe** pour initier le paiement
☑ Vérifier que la transaction a été correctement validée
☑ Utiliser l'**environnement de test (sandbox)** de **Stripe**
☑ Gérer les **clés API** (*publique* et *secrète*) avec sécurité

3.6. Persistance des données

Chaque *microservice* doit utiliser **sa propre base de données**.

☑ Utiliser **Entity Framework Core**
☑ Adapter les *entités* du premier travail pour chaque service concerné
☑ Configurer les connexions à la base dans chaque projet

3.7. Récupération des données externes

Comme dans le premier travail, il est recommandé d'utiliser des API REST publiques pour enrichir la plateforme de données factices :

☑ **Fake Store API** → https://fakestoreapi.com/
☑ **Dummy JSON** → https://dummyjson.com/

Ces données peuvent être utilisées pour remplir les produits, les utilisateurs ou les commandes à des fins de démonstration.

3.8. Déploiement sur Microsoft Azure

Une fois le

4. Structure du projet

Lors de la création de la solution dans Visual Studio, il est recommandé de suivre les bonnes pratiques suivantes :

- ☑ Créer une seule **solution** contenant plusieurs **projets** (*microservices*)
- ☑ Organiser les fichiers dans chaque projet :
  - → **Models/** pour les entités
  - → **Controllers/** pour les contrôleurs
- ☑ Stocker la configuration d'Ocelot dans un fichier **ocelot.json**
- ☑ Utiliser **Git** pour la gestion de version
- ☑ Héberger le projet sur **GitHub**

5. Modalité d'évaluation

Le travail pratique comptera pour **25%** de la note du cours. Ces points sont répartis de la manière suivante :

Partie	Points
Création du back-end	**21,5%**
→ Architecture en *microservices* (*arborescence*)	3.0%
→ Implémentation des différents services	5.0%
→ Configuration de la passerelle API (Ocelot)	2.0%
→ Authentification par jeton (JWT)	3,0%
→ Documentation Swagger + interface unifiée	2.0%
→ Paiement électronique avec Stripe	2.0%
→ Les *entités* et la *persistance* des données	2.5%
→ Communication entre les services	2,0%
→ Déploiement → Microsoft Azure (Bonus I)	2,0%
Qualité du code source	1,0%
Qualité du rapport final	1,5%
La vidéo de présentation du projet	1,0%
Poster pour le FI3E (Bonus II)	5,0%
Total	**25%**

6. Date de remise

- → La date limite de remise est le **26 avril 2025 avant 23h00**
- → Fichiers à remettre :
  - ✓ Un fichier compressé (.zip) contenant le code source,
  - ✓ Le rapport (un modèle **LaTex** est fourni),
  - ✓ Une courte vidéo de démonstration faite avec **OBS Studio**.

Tous les fichiers doivent être remis sur Moodle avant la date limite.

7. Assistance

Si vous avez besoin d'assistance, vous pouvez contacter l'auxiliaire d'enseignement, **Madame Dorra Lamouchi**, directement par courriel à l'adresse Dorra_Lamouchi@uqar.ca afin d'organiser une rencontre sur ZOOM.

Des questions peuvent également être posées au professeur à la fin des séances de cours.

8. Points importants

Note I :

- → Le travail est en équipe de deux (voir la liste des équipes),
- → Le professeur peut poser des questions liées au travail pratique,
- → Le non-respect de l'énoncé entraînera une perte de points,
- → Le plagiat sera sanctionné selon la politique de l'université,
- → Le retard dans la remise du projet entraînera des pénalités.

Note II : *Dans le cas où il y a des aspects qui ne sont pas clairs, n'hésitez pas à m'en faire part afin que je puisse apporter des éclaircissements et éventuellement mettre à jour l'énoncé du travail pratique.*

9. Documentation

Il est recommandé d'utiliser la documentation officielle des différentes bibliothèques indiquées plus haut. Il est également possible de consulter d'autres documentations sur internet.