|  |
| --- |
| Université du Québec à Rimouski |
| Travail Pratique II |
| INF27507 - Technologies du commerce électronique  Professeur – Yacine Yaddaden, Ph. D. |

|  |
| --- |
| ⭢ Goulet, Bastien  ⭢ Schwedt, Kyllian  ⭢ Tchadji Youbi, Nelly Laetitia |

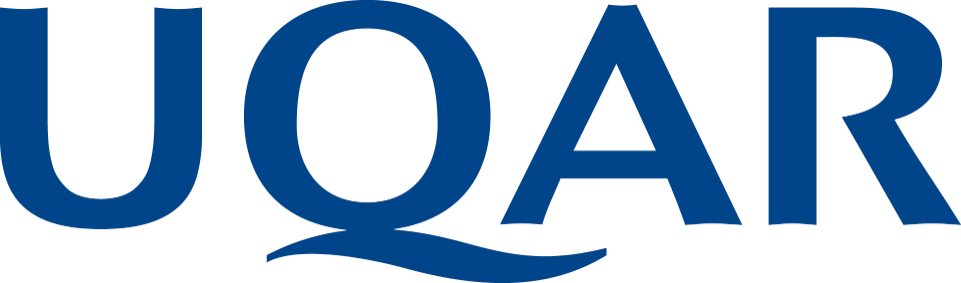


Table des matières

[Introduction 2](#_Toc100852640)

[Technologies utilisées 2](#_Toc100852641)

[Aperçu et description du site développé 3](#_Toc100852642)

[Problèmes et Difficultés rencontrées 4](#_Toc100852643)

[Conclusion 4](#_Toc100852644)

[Références 4](#_Toc100852645)

# Introduction

Durant le précédent travail, nous avons dû effectuer la mise en place d’une boutique de vêtements en ligne grâce au *Framework* web ASP.Net Core MVC. Cependant, pour être en mesure d’atteindre une part de marché plus importante et une meilleure visibilité, il est important de multiplier nos canaux de communication. Ces canaux peuvent comporter une application mobile ou encore une application de bureau.

Un problème se présente avec l’ajout de canaux dans notre écosystème de vente. Comment peut-on faire fonctionner ces applications en parallèle et unifier les données qui sont cruciales au fonctionnement de notre entreprise ? Une solution naturelle se présente : les APIs REST.

# Technologies utilisées

Dans ce travail, il nous fallait mettre en place divers API (*Application Programming Interface*) REST dans un projet à part afin de rendre les actions principales de notre application disponibles à divers canaux de manière standardisée et sécuritaire.

Pour y parvenir, nous avons créé un nouveau projet ASP.Net Core Web API comportant une dépendance avec notre précédent projet MVC. Cette dépendance était nécessaire afin d’accéder au service de base de données. En effet, il était avantageux de réutiliser les opérations à la base de données déjà mises en place dans le précédent travail afin d’éviter de dupliquer du code inutilement.

Un autre objectif de ce travail était d’intégrer un système d’authentification dans les Apis. Afin de respecter la contrainte REST de l’absence d’état, il était nécessaire d’utiliser une méthode d’authentification appliquée lors de chaque requête. Une bonne solution à ce problème était les JWT (*Json Web Token*), une longue chaîne de caractères chiffrée comportant les informations utiles à la validation d’un utilisateur comme son nom d’utilisateur, ses rôles, la durée de vie du jeton et un *hash* permettant de valider l’intégrité du jeton. Le jeton devait être générée lors de l’appel à l’API d’authentification, dans le contrôleur *AuthController* et retourné au client, qui pouvait l’ajouter à l’en-tête *Authorization* de chacune de ses requêtes subséquentes. Ainsi la librairie *JwtBearer* peut valider le jeton lors de chaque requête et retourner une erreur, au besoin, si l’utilisateur n’est pas autorisé à accéder à la ressource demandée.

Pour maintenir les données d’authentification des utilisateurs ainsi que leurs rôles dans l’application, la librairie *Microsoft.AspNetCore.identity* devait être ajoutée au projet. Avec l’ajout du *DbContext* approprié, héritant de la classe *IdentityDbContext*, celle-ci pouvait générer les tables nécessaires à la gestion des utilisateurs dans notre base de données. Nous pouvions ensuite utiliser divers services offerts par *Identity*, tels *UserManager*, *RoleManager*, ou encore *SignInManager* afin de créer de nouveaux utilisateurs, de nouveaux rôles ou encore valider les informations d’authentifications comme le nom d’utilisateur et le mot de passe.

Afin de tester nos Apis, nous devions configurer l’utilisation de l’outil *Swagger*, offrant une interface conviviale pour effectuer des requêtes auprès de nos Apis. Un avantage de cet outil est qu’il effectue la liste automatiquement des Apis présents dans le projet. Il suffit donc de remplir les champs demandés par un Api, et il peut être testé en un clic de souris.

Enfin, nous devions mettre en place une méthode de paiement alternative dans notre application ASP.Net Core MVC, présentée dans le premier travail. La méthode de paiement en question était *Stripe*. La librairie *Stripe* devait donc être ajoutée au projet et le code nécessaire à la génération du formulaire de paiement et à la création d’une transaction devait être implémenté.

# Aperçu et description du site développé

La seule nouveauté présente dans notre premier projet ASP.Net Core MVC est l’ajout de Stripe comme méthode de paiement alternative. Pour la constater, il suffit de procéder à un achat. Donc, ajoutez au moins un article dans votre panier et cliquez sur le bouton pour confirmer. Vous arrivez sur la page présentant les différentes méthodes de paiement. Outre le paiement par crédits dans le compte utilisateur, le paiement par Stripe est maintenant fonctionnel.

Cliquer sur le bouton de paiement affiche une fenêtre modale créée par Stripe demandant d’entrer ses informations de carte de crédit. Une confirmation lance le processus de validation de la carte de crédit. Une fois la carte approuvée, la méthode *StripePayment* du contrôleur *PaymentController* qui crée un nouveau client auprès de Stripe s’il n’existe pas déjà et qui initialise les objets requis afin d’effectuer une transaction auprès de Stripe. Bien entendu, la transaction est fictive puisque nous nous trouvons dans un environnement de test.

Au sein du projet *Api*, plusieurs contrôleurs peuvent être trouvés, chacun comportant des API. Le contrôleur *AuthController* rassemble les API *register* et *login*. Le premier reçoit diverses informations de la part d’un utilisateur et enregistre un nouvel utilisateur dans la base de données *Identiy*. Le second, pour sa part, reçoit le nom d’utilisateur et le mot de passe d’un utilisateur et génère un JWT si les informations correspondent à celles d’un usager existant.

Les contrôleurs *ClientController* et *SellerController* rassemblent des API qui effectuent les mêmes opérations que plusieurs méthodes d’actions principales de la première application. En effet, nous retrouvons des API pour la création ou la modification de produits dans le contrôleur des vendeurs. Nous retrouvons également des API permettant la gestion du panier, le visionnement de ses informations ainsi que l’achat simple dans le contrôleur pour les clients.

Bien entendu, afin d’appeler avec succès les API dans le *ClientController*, il faut être authentifié avec le rôle « Client » et avec le rôle « Seller » pour effectuer des requêtes au *SellerController*. Il est possible de tester les API facilement avec l’outil Swagger qui est ouvert dans le navigateur par défaut lors du lancement du projet *Api*.

# Problèmes et Difficultés rencontrées

Les seuls problèmes, dans ce deuxième travail, sont survenus de la configuration des librairies *Identity* et *JwtBearer*. En effet, afin de faire fonctionner l’authentification des utilisateurs et, plus particulièrement appliquer *JwtBearer* en tant que *middleware* dans ASP.Net Core afin qu’il valide le JWT dans le champ *Authorization* de l’en-tête des requêtes aux API. Cette étape a nécessité plusieurs tentatives et beaucoup de recherche afin de trouver les configurations nécessaires dans le fichier *Program.cs*. La configuration a été problématique puisque les librairies en cause évoluent très rapidement et chaque personne sur internet semble avoir sa manière propre de les configurer.

# Conclusion

En conclusion, ce travail a été un aperçu intéressant dans le monde des API et des méthodes modernes d’authentification. Les JWT sont désormais répandus dans l’univers de la programmation web, et il aura été avantageux de se pencher sur leur fonctionnement de façon autonome. Ainsi, nous ressortons de ce travail avec une compréhension utile de cette technologie.

# Références

[1] Yaddaden Y. INF27507 - Séance 09 - Initiation à l’ASP.NET – Introduction aux API REST - V1.0

[2] Yaddaden Y. INF27507 - Séance 07 - Introduction aux systèmes et services de paiement électroniques - V1.0

[3] Saseendran S. (2022, janvier). JWT Authentication And Authorization In .NET 6.0 With Identity Framework. Repéré à <https://www.c-sharpcorner.com/article/jwt-authentication-and-authorization-in-net-6-0-with-identity-framework/>

[4] Use JWT Bearer Authorization in Swagger OpenAPI. Repéré à <https://www.freecodespot.com/blog/use-jwt-bearer-authorization-in-swagger/>

[5] Kempeth. (2022, février). How do I get Identity UserManager in .NET 6.0. Repéré à <https://stackoverflow.com/questions/71294352/how-do-i-get-identity-usermanager-in-net-6-0>