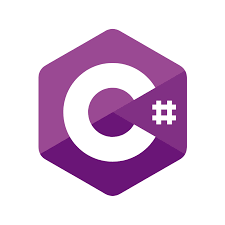


PROJET ARCHILOG

Création d’une librairie



Leo PEYRE

Aaron SAKSIK

Melly LUCAS

Année universitaire 2021-2022           Professeur : Vincent LECLERC

Table des matières

[Introduction 3](#_Toc88214856)

[Objectifs 3](#_Toc88214857)

[Critères 3](#_Toc88214858)

[Support de test 4](#_Toc88214859)

[La répartition des taches 4](#_Toc88214860)

[Description des tâches effectuées 5](#_Toc88214861)

[Le CRUD 5](#_Toc88214862)

[La pagination 5](#_Toc88214863)

[Les tris 5](#_Toc88214864)

[Les filtres 5](#_Toc88214865)

[La recherche 6](#_Toc88214866)

[Les tests unitaires 6](#_Toc88214867)

[La réflexion 6](#_Toc88214868)

[Difficultés rencontrées 7](#_Toc88214869)

[Conclusion 8](#_Toc88214870)

# Introduction

## Objectifs

L’objectif du projet est de créer et d’implémenter une librairie permettant de renvoyer une API REST générique qui fonctionnerai pour le CRUD, pagination, filtres, tri et recherche comme nous le verrons plus bas. Cette dernière doit être la plus complète possible ; en fonction des différents modèles proposés par l’application.

Le principe sera de réduire la charge de travail pour développeur API .NET car il aura juste à importer cette Librairie dans son projet est de faire hériter ces Controllers de "BaseController" et ces Models de "ModelBase"

## Critères

Les différents critères imposés étaient de respecter les différentes normes en termes de réponses puis de fournir une API permettant de générer :

* Le **CRUD**
* La **pagination** et les différents **Headers** qu’elle contient.
* Les différents **tris**.
* Les différents **filtres** que peut rencontrer une application web standard ;
* Une **recherche** générique ;
* Une **documentation** complète avec **SwaggerUI** ;
* Puis des **tests unitaires** permettant de vérifier que tout fonctionne.

## Support de test

* Afin de réaliser notre application, nous avons créé une **base de données** hébergée sur une instance **Azure,** par la suite créés différentsmodèles.
* Pour tester nos **requêtes** nous avons utilisé **Postman**.
* Nous avons également utilisé le **gestionnaire de version GitHub**, pour partager notre code et nous répartir les tâches avec différentes **branches attribuées à chacun**.
* L’utilisation du **déboguer** pour tester point par point les **différentes parties et cheminement de notre code.**

## La répartition des taches

Les différentes tâches contenues dans le projet, ont été dispatchées de manière équitable en fonction des qualités et de l’envie de chacun. En partant de ce principe :

Melly s’est occupé de la partie filtre avant la réflexion et des tests unitaires.

Léo s’est chargé de réaliser les tâches restantes c’est-à-dire les fonctions permettant le tri, la recherche, les headers pour la pagination.

Aaron quant à lui a structuré le "BaseController" afin de récupérer le travail du groupe est de l’implémenter dans différentes Methods, pour différentes Request et utilisé la réflexion.

# Description des tâches effectuées

## Le CRUD

Chaque méthode du crud (Get, Post, Put, Delete) est gérer côté librairie ce qui facilite la tâche du dev API.

## La pagination

Nous avons réalisé une méthode qui permet la pagination. En récupérant les values du queryString **range** nous pouvons créer un intervalle qui correspondra au nombre de produit que nous souhaitons afficher.

Nous avons également géré les **Response** **headers** correspondant aux values du tri.

## Les tris

Pour ce qui est des tris, nous avons réalisé2 fonctions de tri ; une permettant que les values soient triées de manière **ascendante** et l’autre **descendante**.

## Les filtres

Les Filtres vont permettre à l’utilisateur de réduire la Response de la Query afin d’affiner ses recherches. Les filtres sont construit de manière à ce que chaque champs soit filtrable indépendamment mais également à la suite sur une même Query. A l’avenir il serait intéressant de pouvoir créer une Feature qui permettrai au concepteur de l’API de limité les champs filtrable à sa guise.

## La recherche

Pour la recherche, une nouvelle route devra être exploité (/api/{model}/search/). Nous avons utilisé l’opérateur **contains** qui nous a permis de récupérer les valeurs dans le model qui contiennent les caractères mis en paramètre de cette dernière.

## Les tests unitaires

Les tests unitaires nous ont permis de tester toutes les méthodes. Pour cela, nous avons créé une nouvelle base de données stockée en mémoire avec de faux objets facile à utiliser. Autrement dit, nous avons moqué la base d’origine afin qu’elle s’applique sur les méthodes du BaseController. A partir des méthodes statiques d’assertions fournies par les tests, il nous a été possible de tester nos méthodes.

La réflexion

La réflexion nous a permis de réaliser une grande partie de ce projet car pour tous les sortes de filtres que nous avons créé, il nous fallait récupérer les QueryStrings pour appliquer tel ou tel methode et compléter la Request. Seulement une fois que nous avions la Key et la Value du filtre en question le Type de celles-ci était String, donc obligation de passer par la création de lambda, de property, la conversion. . .

# Difficultés rencontrées

Les différentes difficultés que nous avons pu rencontrer lors du développement étaient pour :

* Léo : L’aspect de **réflexion**, c’est un concept dont je comprends l’utilité, mais pas comment le faire individuellement. Mise à part cela rien en particulier.
* Melly : J’ai également eu des problèmes sur la compréhension de la réflexion et n’est pas réussi à l’appliquer aux filtres. Avec l’aide de mes camarades de projet, ce point s’est éclairci. La partie tests unitaires m’a posé un problème notamment pour moquer la base de données. A partir de l’aide de notre professeur, nous avons pu tester notre librairie.
* Aaron : C’est vrai que la réflexion était difficile pour tout le monde, car c’était la première fois où nous l’avons réellement utilisé et approfondi. Pour ma part j’ai eu un peu de mal au début du projet d’aboutir à une structure de code du côté de la librairie mais finalement tous c’est plutôt bien passé.

Dans un premier temps nous avons mis en place un **dépôt GitHub** pour pouvoir faire de **l’intégration continu** avec diverses branches en fonction des membres du groupe, mais nous avons pris conscience au bout d’un moment que cela n’était pas optimal comme façon de se répartir les tâches et que nous aurions dû mettre en place des **tickets** avec des instructions, puis en fonction de ses dernières, y attribuer un membre du groupe qui une fois fini le ticket terminé, **merge** sur la branche principale.

# Conclusion

Pour conclure, nous sommes tout particulièrement satisfait de ce projet.

Ce dernier nous a permis de mettre en pratique ce que nous avons vu en cours, nous sommes impatients de retrouver des défis équivalent durant notre formation pour pouvoir passer un nouveau cap et enrichir encore plus nos compétences donc pour cela nous vous en sommes très reconnaissant. Il y a eu une bonne entente sur ce projet autant productive qu’amicale ce que nous pensons être obligatoire pour réaliser un projet de bonne qualité en commun.

Nous sommes donc heureux d’avoir participé à ce projet et d’avoir acquis plusieurs compétences qui nous serons certainement obligatoire durant notre vie professionnelle.