Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

2022/2023

Systèmes d'information 2

Rapport 01

TP1 – Comparaison Frameworks Backend ASP .NET et Spring MVC

Ecole	Haute école d'ingénierie et d'architecture de Fribourg (HEIA-FR)
Branche	Systèmes d'information 2
Etudiants	Barras Simon, Collaud Roman, Terreaux Nicolas
Classe	ISC-IL-3
Version/Date	V1 du 19.11.2022

Hes·so FRIBOURG/FREIBURG
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz

1 Introduction

Ce travail pratique a pour but nous faire découvrir et prendre en main deux frameworks backend à travers l'implémentation d'une simple application. Nous allons également comparer les deux solutions afin d'en sortir leurs points forts et faiblesses.

2 Comparaison

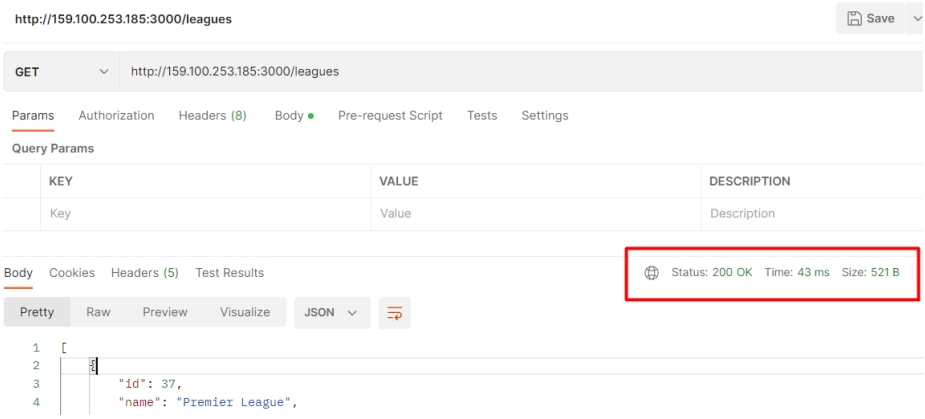
Les deux solutions sont des frameworks webs populaires au sein de l'industrie actuelle. Leur utilisation dépend essentiellement de la préférence des langages que les entreprises utilisent. Pour ceux qui ont une préférence pour les langages basé sur JVM comme Java out Kotlin partiront sans doute sur Spring Boot. Au contraire, les adeptes de C# vont certainement opter pour ASP.NET Core.

Nous avons choisi 3 critères de comparaison pour analyser les apects qui diffèrent entre les frameworks : la performance, la facilité d'implémentation des tests unitaires et le support pour une stratégie CI/CD.

2.1 Performance

Il est intéressant de savoir quel framework est le plus performant en termes de temps de requêtes.

Pour évaluer la performance des frameworks, nous avons requêté les APIs respectives à l'aide de Postman. Cet outil nous permet d'obtenir le temps de réponse par requête et les présenter dans un tableau pour une comparaison. La requête « /leagues » est particulièrement intéressante car elle va demander une jointure avec deux autres tables pour récupérer la liste des parieurs et des matchs de la ligue.



Nous avons fait le test ci-dessus avec un certain nombre de routes. Voici le tableau récapitulatif des requêtes avec les temps d'exécutions :

Requête Spring : http://159.100.253.185:3000 .NET : http://89.145.164.93:80	Temps	Temps
/leagues	43ms	20ms
/games	27ms	19ms
/players	33ms	15ms

/game/add
Body :
{
"date":"2022-11-02",
"location":"Ependes",
"homeTeam":"Test1",
"awayTeam":"Test2",
"leagueid": 1
}
	93ms	20ms

/players/add
Body :
{
"firstname": "TestFirstName",
"lastname": "TestLastName",
"birthdate": "2000-01-01",
"favoriteTeam": "TestFavorite"
}
	76ms	33ms

/league/add/LeagueTest	122ms	16ms

La différence est assez grande. Ceci s'explique en partie par la qualité de connexion du client mais .Net reste plus rapide.

De plus, nous avons effectué quelques recherches sur Internet et nous avons relevé que Spring est moins bon dans tous les domaines de la performance. Que ce soir de la rapidité des requêtes ou encore de l'utilisation de la RAM.

https://medium.com/@putuprema/spring-boot-vs-asp-net-core-a-showdown-1d38b89c6c2d

2.2 Support pour CI/CD

2.2.1 Spring

Pour le framework **SpringBoot**, le projet utilise maven comme outil de gestion de projet. Pour la mise en place de CI/CD, trois stages ont été mis en place. La compilation de l'application (mvn compile). Une phase de tests (mvn test). Enfin le stage de déploiement qui s'occuper de build une image docker, la push sur la registry de gitlab.

Un point faible de maven est la version utilisée. En effet, l'image que nous devons utiliser doit correspondre parfaitement avec celle que nous avons utilisé pour le développement de l'application. Nous avons eu quelques soucis afin de trouver une version compatible car il en existe énormément.

2.2.2 Asp.net

Le support pour le CI/CD et les tests est assez fournis mais solutions proposées sont toutes très compliquées et pas évidentes à mettre en place.

2.2.3 Synthèse

Les langages pre-devops ne sont pas toujours optimisé pour les techniques actuelles de CI/CD et il faut se débrouiller avec des gestionnaires de paquets. Heureusement que le langage actuel fournisse des solutions plus simples. Cependant, ils restent très largement utilisé en production est dès que nous avons une solution fonctionnelle, il est assez facile de reproduire les résultats et de les maintenir.

2.3 Facilité d'implémentation des tests unitaires

2.3.1 Spring

Spring fournit une annotation @SpringBootTest qui va permettre de créer un contexte d'application qui va contenir tous les objets nécessaires pour faciliter le testing.

Le framework fournit aussi d'autres annotations comme @WebMvcTest pour tester le bon fonctionnement des controlleurs ou encore @DataJpaTest qui permet de mettre en place un environnement avec une base de données intégrée pour tester la couche de persistance.

Dans notre projet, nous avons utilisé la base de données de productions qui se trouve sur l'instance d'Exoscale. Ce n'est pas forcément la bonne pratique à faire, mais au moins, le lien entre le backend et la base de données est aussi testé.

Pour se faire, nous avons utilisé **MockMvc.** Cet outil fournit un support pour les tests de Spring MVC. Il encapsule tous les beans de l'application web et les rend disponibles pour les tests. Il nous permet donc d'utiliser les routes que nous avons préalablement. C'est un peu comme si on utilisait Postman depuis les tests unitaires.

En conclusion, MockMvc nous permet d'effectuer des tests extrêmement facilement. Le seul souci, nous n'avons pas trouvé de manière pour effectuer des tests compatibles avec la pipeline CI/CD sans passer par la base de données de production ce qui pourrait poser problème lors d'une utilisation réel.

2.3.2 Asp.net

Les tests pour .NET sont compliqué à mettre en place. De plus, comme nous sommes débutants, nous n'avons pas forcément bien implémenter l'application pour qu'elle soit facilement testable. En revanche, une fois qu'ils sont mis en place ils sont faciles à être utilisé et améliorer.

2.4 Bilan

Les deux frameworks ont leur propre ensemble de fonctionnalités. Les deux sont des technologies matures, fiables et ont une grosse communauté derrière. En fin de compte, la meilleure solution dépend vraiment du besoin pour le projet que l'on veut développer.

3 Synthèse

Nous avons décidé de créer un frontend indépendant commun pour les deux backends.

Nous donnons la possibilité à l'utilisateur de changer le backend utilisé depuis l'interface de manière dynamique à l'aide d'un switch dans la barre de navigation.

3.1 Problèmes rencontrés (Frontend Vue.js)

Le frontend est un projet Vue3 + Vite avec TypeScript et utilise Pinia comme technologie de Store.

Après avoir commencé le développement avec composition API, les problèmes rencontrés notamment avec TypeScript et le Store m'ont poussé à utiliser finalement Option API.

Le déploiement du frontend a été un vrai challenge et m'a demandé un effort conséquent. En effet c'est la première fois que je mettais en place une stratégie DevOps complète avec le déploiement sur le serveur exoscale. La pipeline CI/CD comprend deux stages. Dans un premier temps, l'application est conteneurisée à l'aide du Dockerfile qui build le projet et va push l'image sur la registry de la forge.

La deuxième étape consiste à se connecter via ssh au serveur exoscale et lancer le docker-compose qui va entraîner la création et le lancement du container sur notre instance.

Ne connaissant que très peu la partie déploiement, j'ai rencontré pas mal de problèmes avec la configuration du serveur web. Je n'avais pas configuré de reverseProxy pour rediriger correctement les requêtes vers les APIs des backends et cela provoquait des erreurs CORS.

3.2 Spring

L'implémentation de Spring est plutôt simple. En effet, nous arrivons assez rapidement à avoir un backend fonctionnelle avec ORM. Le point le plus négatif pour d'après nous est la gestion des dépendances et le déploiement. Le déploiement est la partie qui nous a pris le plus de temps à cause de Maven. Nous avons eu des erreurs que nous n'avons pas pu reproduire lors de nos compilations avec maven. Par exemple, lorsque notre application était sur Exoscale, elle démarrait en boucle ce qui posait passablement de problème. Le simple fait d'avoir trouvé une bonne image a réglé le problème.

3.3 Asp.net

La partie .net à pauser énormément de problème et comme plein de code est généré il est très difficile de debuguer. Nous n'avons pas du tout apprécié cette expérience avec ce framework mais nous reconnaissons qu'il est très efficace et que nous pouvons faire des projets de grande envergure très performants.

4 Conclusion des frameworks

Pour conclure, nous pouvons dire que les deux frameworks ont leurs propres caractéristiques mais dans l'ensemble sont assez similaires. Même si asp.net est un peu plus rapide, Spring est tout de même un bon framework. La prise en main d'asp.net est plus difficile pour nous car nous venons du monde de Java et par conséquent, nous connaissons mieux cet univers.

5 Conclusion du travail

Nous n'avons pris aucun plaisir à effectuer ce travail. Cela s'explique par la charge de travail demandée qui est déraisonnablement démesuré par rapport au temps allouer en classe et par rapport à d'autres cours ayant le même nombre de crédits. Aussi, nous trouvons dommage que la note ne pèse pas plus que ça dans la moyenne cela demande un grand effort et nous ne sentons pas récompensé. De plus nous la partie du développement des framework ne nous n'a pas intéressé car nous avions déjà vu sa en première année et nous n'avions pas le temps de rentrer dans les détails. En revanche, nous avons quand même trouvé intéressant de réaliser un déploiement complet sur Exoscale et nous a permis aussi de mettre en pratique toutes nos connaissances théoriques sur le DevOps (CI/CD, Dockerization, Automatisation du déploiement). Pour finir, nous vous demandons de revoir complètement votre TP car, comme mentionner, la charge de travail est impossible, surtout si on rajoute une classe inversée et une évaluation…