Université Gustave Eiffel

Master Informatique

Rapport Technique

REST et Web Services

Jimmy Teillard - Irwin Madet - Lorris Creantor

Alternants

Chargé de Cours: Mahdi Zargayouna

M2

Décembre 2022

Contents

1 Introduction

Ce rapport technique à pour but de compiler nos différents choix d'implémentation ainsi que l'architecture du projet. Il sera dédié une partie quant aux difficultés rencontrés, ainsi que de possible pistes d'auto-améliorations.

Il contient aussi un manuel utilisateur des différents modules qui composent le projet.

Ce projet a pour but d'appliquer les bases de la programmation de Services Web SOAP et RMI.

2 Manuel Utilisateur

2.1 Compilation

Pour compiler ce projet et l'exécuter, il est nécessaire d'avoir Java 17 (ou plus) installé sur votre machine, ainsi qu'un serveur Tomcat 8 disponible. Les commandes explicitées ci-après supposeront que vous avez Gradle installé sur votre machine aussi ; cela dit, ce n'est pas une nécessité, et vous pouvez utiliser le wrapper script fourni à la racine du projet (gradlew et gradlew.bat).

Pour compiler, il suffit de lancer la commande suivante à la racine du projet :

```
gradle clean build
```

Ensuite, il est possible de lancer les différents modules avec votre IDE préféré.

2.2 Modules

Chaque module est représenté par un dossier, et est défini par un fichier Gradle à la racine du projet. Les modules sont les suivants :

- common : qui contient les interfaces RMI qui représentent les utilisateurs ainsi que les vélos, et toutes les interfaces qui permettent de les manipuler aisément.
- users-server : dépend de common, et qui contient les implémentations des interfaces RMI utilisateur, ainsi qu'un main pour lancer le serveur RMI.
- bikes-server : dépend de common et de users-server, et qui contient les implémentations des interfaces RMI des vélos, ainsi qu'un main pour lancer le serveur RMI. Il dépend de users-server car les vélos sont définis par leur "owner" et leur "orderer", qui sont des utilisateurs.
- webserver : dépend de common, users-server, et bikes-server, est un serveur REST Spring qui sert d'API inverse aux serveur RMI, pour communiquer au frontend.
- web-client : une application frontend Vue.js qui permet de créer des utilisateurs et se connecter, ainsi que de louer des vélos, pour ensuite les rendre et y laisser des avis ainsi que l'état desdits vélos.
- bankservice : est un Web Service SOAP qui sert de banque temporaire pour les utilisateurs.
- webservice : dépend de common, users-server, et bikes-server, est un Web Service SOAP qui permet d'acheter les vélos qui ont été loués plus de deux fois.

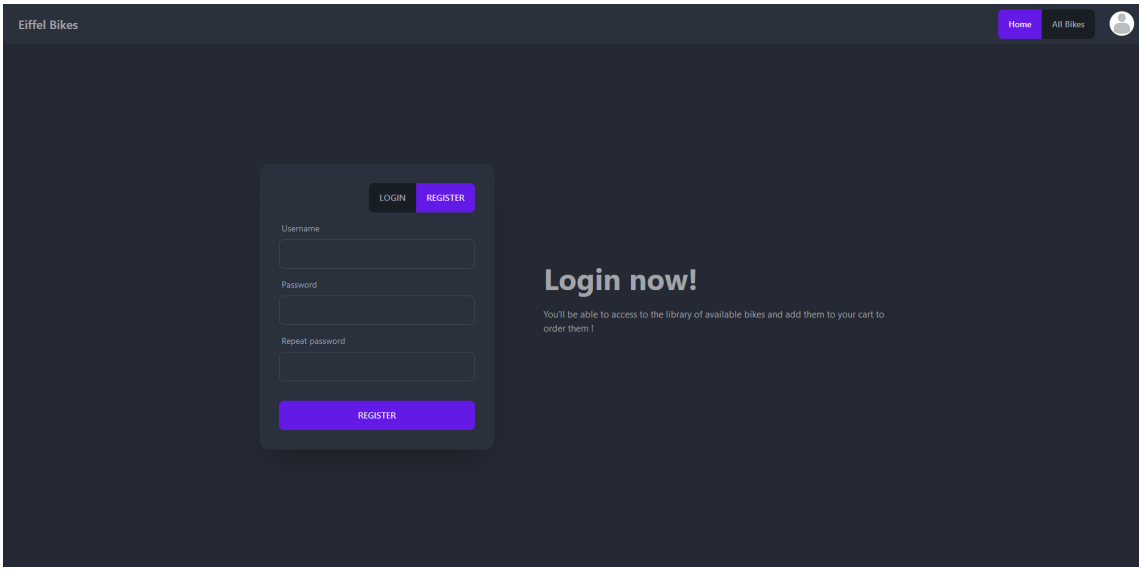
2.3 Exécution

Il est conseillé de lancer les différents services dans cet ordre :

- Le user-server
- Le bike-server
- Le bankservice
- Le webservice
- Le webserver Spring
- Le web-client JS

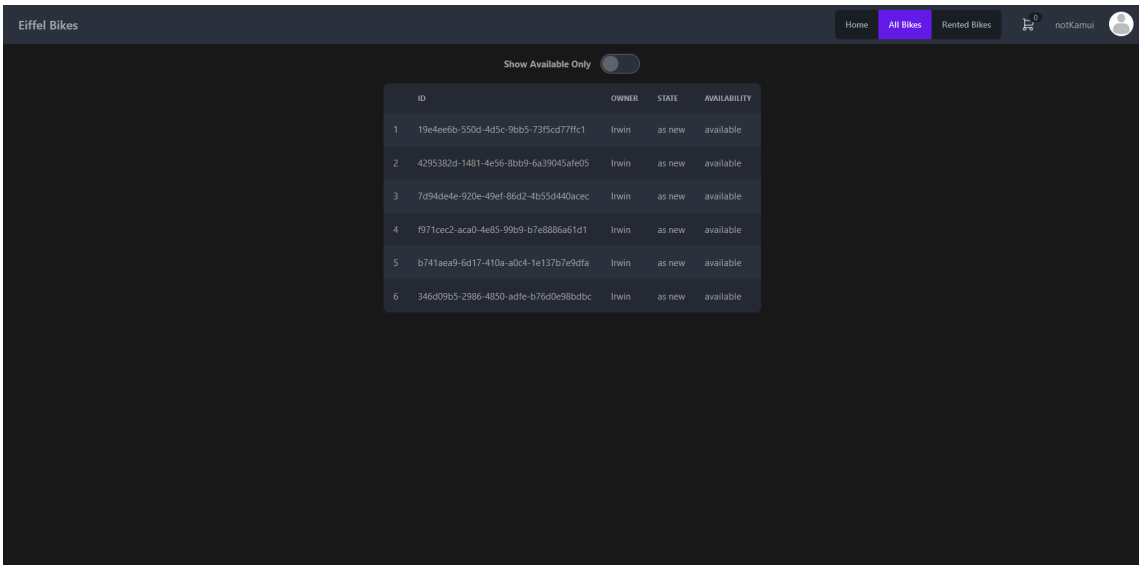
2.4 Web Client

La première page visible pour un utilisateur lambda est la page de création de compte/connexion



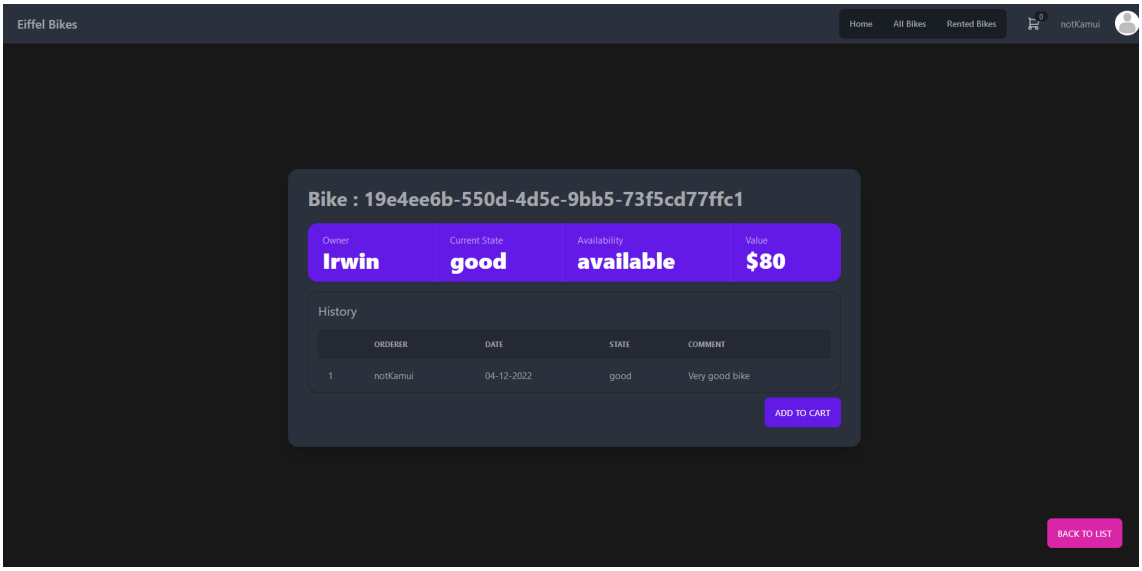
Il suffit d'y renseigner un nom d'utilisateur qui n'est pas déjà utilisé, et un mot de passe. Si le formulaire n'est pas correcte, une notification le prononcera.

Une fois connecté, il est possible d'aller consulter la liste de vélos disponibles à louer



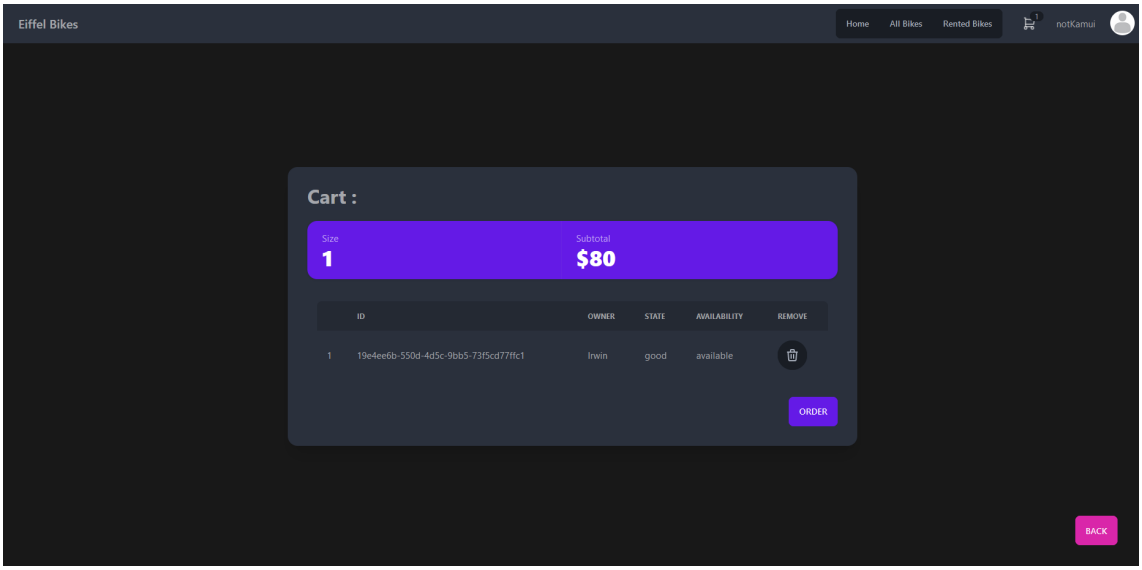
Sur cette liste on peut voir les différents vélos disponibles et même cacher ceux qui sont déjà loués. Chaque vélo à un possesseur, un état actuel, ainsi qu'une valeur représentant sa disponibilité.

En cliquant sur l'une des lignes, on peut voir une vue plus détaillée du vélo



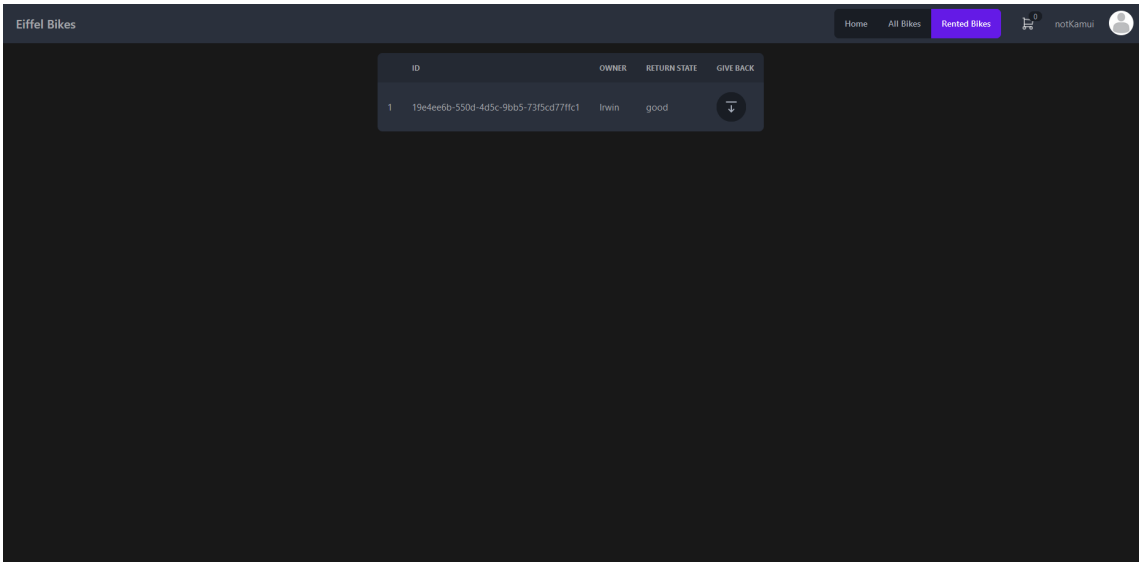
Sur cette vue, on peut voir, en plus des informations précédentes, la valeur du vélo, ainsi que son historique de commandes, avec les utilisateurs qui les ont faites, la date, et les commentaires ajoutées.

On peut cliquer sur le bouton en bas pour l'ajouter au panier, que l'on peut consulter en cliquant sur le bouton en haut à droite de la page, en forme de chariot.

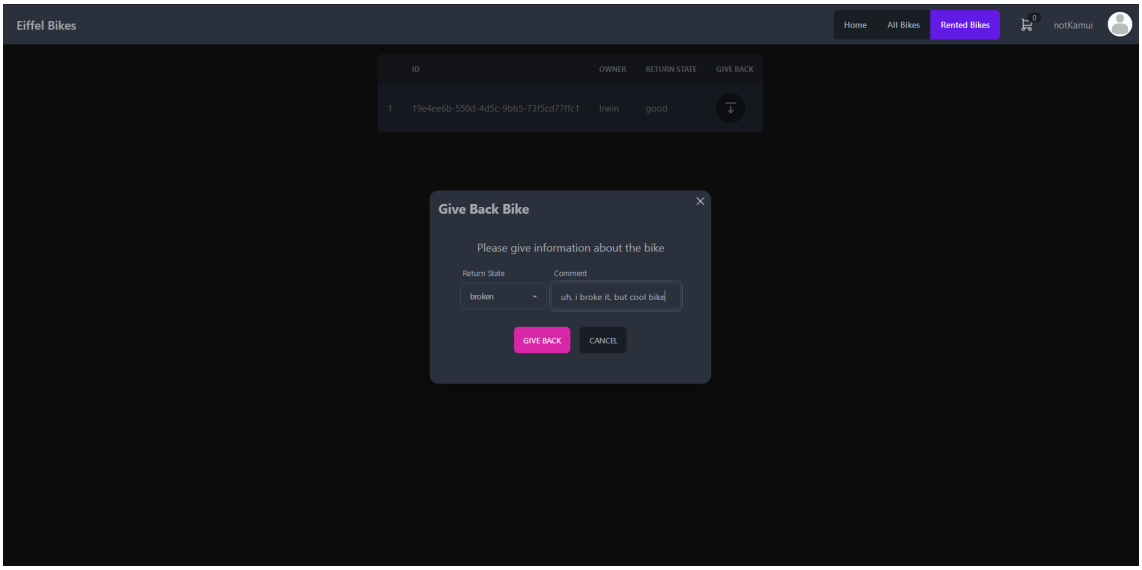


Sur l'écran du panier, on peut retirer les vélos qui ne nous intéressent plus, ou bien les commander.

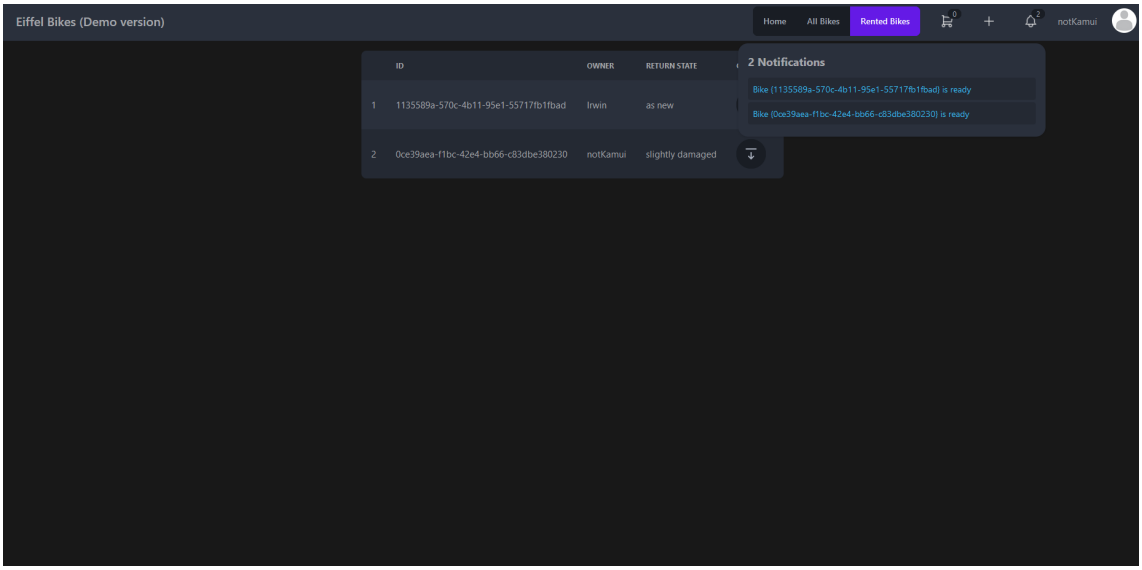
Une fois les vélos commandés, on peut les consulter sur l'écran "Rented Bikes" (Commander un vélo ne coûte rien)



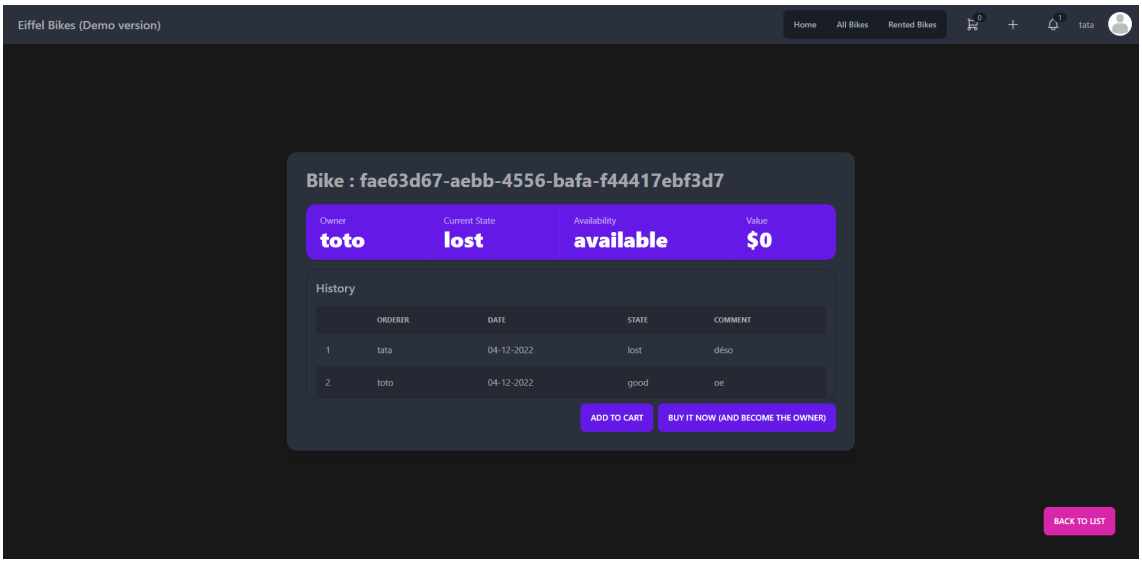
Finalement, on peut rendre un vélo en cliquant sur la petite icône de la ligne du vélo concerné.



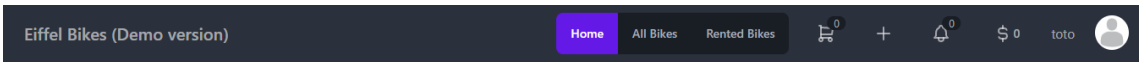
Il est noter que si un utilisateur tente de commander un vélo qui est actuellement déjà loué (marqué "unavailable"), alors il est placé en liste d'attente, et il ne sera pas disponible dans "Rented Bikes" immédiatement. Lorsque le vélo est rendu, et que l'utilisateur est le prochain sur la liste d'attente, alors le vélo lui est remit, et il reçoit une notification



Un utilisateur peut aussi acheter un vélo pour en devenir le propriétaire, s'il possède assez d'argent, et que le vélo a été loué au moins une fois. (oui, un vélo perdu est gratuit, parce que les bulles spéculatives, c'est stylé).



A noter que l'utilisateur peut choisir sa devise et voir l'argent qu'il a sur son compte en temps réel.



3 Choix d'implémentations et Difficultés

3.1 Services RMI

D'après l'énoncé, il nous était necessaire de lancer deux JVM différentes pour le service qui gère les utilisateurs, et le service qui gère les vélos en eux mêmes. Pour cela, on a choisi de créer des modules Gradle, et d'avoir un module intermédiaire "common", sur lequel les deux autres dépendent, et qui contient les interfaces qui seront implémentées par les deux autres modules qui servent alors de serveurs RMI.

Un utilisateur est représenté par un id, et un username. Un vélo est représenté par un id, un possesseur (owner), un potentiel locataire (orderer) s'il le vélo est loué, et une liste qui représente l'historique de commandes. Ces deux structures sont des interfaces Remote, car il est nécessaire qu'elles soient mutables à distance (elles ne sont pas read-only)

Par opposition, les structures qui représentent les objets read-only, tels que ceux qui représentent les entrées de l'historique de commandes, sont des POJO serializables.

Pour manipuler les utilisateurs et les vélos plus aisément, nous avons opté pour la créations d'interfaces Storage : UserStorage et BikeStorage.

En effet, les implémentations de ces interfaces, stockent les structures qui gèrent et manipulent les utilisateurs et vélos en masse. Elles sont thread-safe. Par exemple, UserStorage permet de se login, recupérer des session tokens, etc ; et le BikeStorage permet de louer des vélos, changer leur état, rendre les vélos, etc.

En ce qui concerne la commande de vélos déjà en cours de location, nous avons choisi d'implémenter une queue, ainsi qu'un pattern listener pour attendre une notification lorsqu'un vélo se libère.

3.2 Web Services

Les deux services à créer sont BankService et GustaveBikeService, respectivement dans les modules bankservice et webservice. Cela à été pour nous le point de difficulté principal de ce projet. En effet, nous étions partis dans l'optique de faire notre projet entièrement avec IntelliJ, mais ce fut une erreur fatale. SOAP étant une technologie obsolète, le support de IntelliJ est non loin d'être inexistant.

Nous avons donc du gérer notre projet dans deux IDE différents (Eclipse), avec grande difficulté et confusion. Le service de banque, très simple en fonctionnalité, à pu être créé, et est fonctionnel, et consommable. Cependant, en ce qui concerne le GustaveBikeService, nous n'arrivions pas à consommer l'API RMI depuis ce dernier, recevant des exceptions telles que des NoClassDefFoundExceptions. Au final, le problème était dû au fait que les Web Service semblent ne pas savoir ce que sont les projets multimodulaires... Honnêtement, c'est une assez grosse déception, d'un point de vu de développeur, en terme de passage à l'échelle.

3.3 REST Server et Client frontend

Le serveur Spring est simplement un miroir de l'API RMI, et permet d'interagir avec celle-ci avec une API REST. Cela permet au client JS de consommer cette dernière et donc de laisser les utilisateurs louer des vélos et laisser des avis facilement.

4 Rétrospections et Conclusion

Pour être honnête, ce projet à été douloureux et décevant ; en effet, la majorité de sa difficulté et des frustrations rencontrées ont été dues au sujet très peu explicite, ainsi qu'à l'utilisation de technologies obsolètes. D'autre part, c'est un point qui s'est

vocalisé à plusieurs reprises au sein de la promotion, et même des précédentes, mais cette matière n'a absolument aucun droit de s'appeler "REST". Depuis le début de nos études supérieures, c'est strictement et unanimement la première fois que nous voyons un projet comme une corvée plutôt qu'une opportunité d'apprentissage.