|  |  |
| --- | --- |
| EMF-Shelf Projet module 133  S:\EMF\ElevesCommun\2015-2016\Waeberla\Logos\Logo_EMF-Informatique_FR_RVB_50.jpghttp://www.emf.ch/sites/default/files/images/logus/id_logus_fr.png | |
|  | http://www.emf.ch/sites/default/files/images/logus/id_logus_fr.png  Nathan Clapasson et Alexandre Haymoz  Groupe numéro et Classe  Module 133 du 23.03.2023 au 05.05.2023 |
|  |  |

**Table des matières**

1 Introduction et contexte du projet 3

2 Analyse à faire complètement avec EA 4

2.1 Use case avec les explications des cas pour les applications 4

2.1.1 Use case globale 4

2.1.2 Use case AP1 5

2.1.3 Use case AP2 6

2.1.4 Use case APIGateway 7

2.1.5 Use case REST1 8

2.1.6 Use case REST 2 9

2.2 Sequence System global entre les applications 10

2.2.1 Réserver un livre 10

2.2.2 Bannir un utilisateur 11

3 Conception à faire complétement avec EA 12

3.1 Class Diagram complet avec les explications de chaque application 12

3.1.1 Class Diagram APIGateway 12

3.1.2 Class Diagram REST 1 13

3.1.3 Class Diagram REST 2 14

4 Hébergement 14

4.1 REST 14

4.2 APIGateway 14

5 Installation du projet complet avec les 5 applications 14

6 Outils, langages (versions, définitions, installations) 14

7 Tests de fonctionnement du projet 15

8 Auto-évaluations et conclusions 15

# Introduction et contexte du projet

Dans le cadre de ce module, nous devons réaliser un projet découpé en 5 parties : deux clients, une Gateway et deux serveurs REST. Nous avons donc décidé de créer un système de bibliothèque avec un client pour les utilisateurs, un client pour les admins, un REST pour gérer les livres et les réservations, un REST pour gérer les comptes utilisateurs et la Gateway qui fait le lien entre les clients et les serveurs.

Une image contenant diagramme

Description générée automatiquement

# Analyse à faire complètement avec EA

## Use case avec les explications des cas pour les applications

### Use case globale

Du côté client, nous avons trois rôles : visiteur qui peut se connecter, user qui peut faire des actions sur l’application client et admin qui peut faire des actions sur l’application administrateur.

Une image contenant diagramme

Description générée automatiquement

### Use case AP1

Dans l’application client, les visiteurs peuvent voir les livres, se connecter et créer un compte. Les utilisateurs connectés peuvent se déconnecter, réserver un livre et le rendre.

Une image contenant diagramme, schématique

Description générée automatiquement

### Use case AP2

Dans l’application administrateur, un visiteur peut uniquement se connecter. Un admin connecté peut gérer les livres et les réservations, bannir un utilisateur et se déconnecter.

Une image contenant diagramme

Description générée automatiquement

### Use case APIGateway

Les deux clients peuvent demander une connexion pour les utilisateurs et tuer la connexion. Le client utilisateur peut demander une création de compte, demander les livres et gérer les réservations de l’utilisateur. Le client admin peut gérer les livres et les réservations de tous les utilisateurs et bannir un utilisateur.

Une image contenant diagramme, schématique

Description générée automatiquement

### Use case REST1

L’API peut effectuer des CRUD (rechercher, créer, modifier et supprimer) pour les livres et les réservations sur la base de données.

Une image contenant diagramme

Description générée automatiquement

### Use case REST 2

L’API peut enregistrer un nouvel utilisateur, connecter un utilisateur et bannir un utilisateur.

Une image contenant diagramme

Description générée automatiquement

## Sequence System global entre les applications

### Réserver un livre

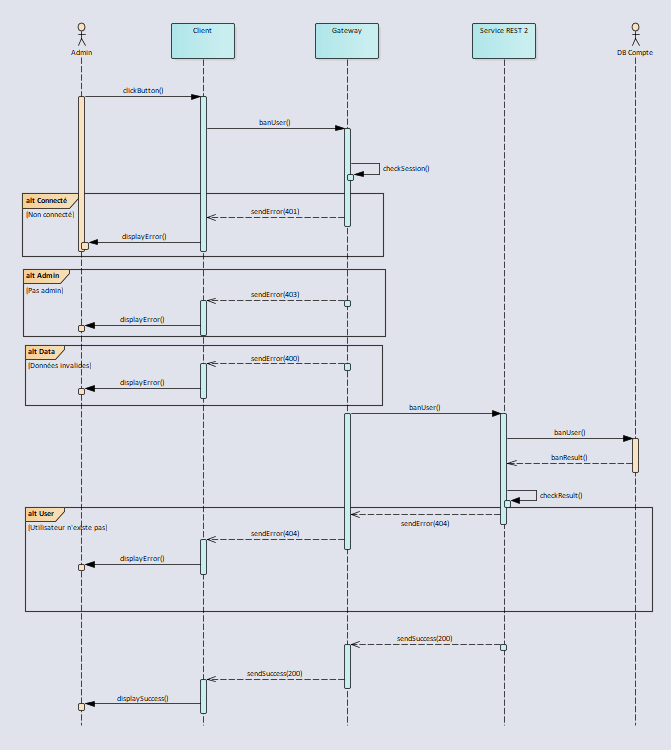
Lorsque l’utilisateur clique sur le bouton pour réserver un livre, le client va envoyer la requête au douanier qui va vérifier la session. Il va renvoyer des codes d’erreur si l’utilisateur n’est pas connecté (401) ou si des données sont manquantes (400). Il va par la suite envoyer la requête au REST 1 qui va vérifier si le livre est disponible. Si le livre n’est pas disponible, il va retourner une erreur 400. Pour finir, le service REST va créer la réservation et retourner un code 200 si tout va bien et 500 en cas de problème.

Une image contenant texte, ciel, carte

Description générée automatiquement

### Bannir un utilisateur

Lorsqu’un administrateur clique sur le bouton pour bannir un utilisateur, le client va envoyer la requête à la Gateway qui va vérifier la session. Il retourne un code d’erreur si l’admin n’est pas connecté (401), s’il n’est pas admin (403) ou si les données ne sont pas valides (400). Ensuite, la Gateway va envoyer la requête vers le service REST 2 qui va bannir l’utilisateur sur la base de données. Si l’utilisateur n’existe pas, un code d’erreur 404 est retourné.



# Conception à faire complétement avec EA

## Class Diagram complet avec les explications de chaque application

### Class Diagram APIGateway

Notre API est composée de trois parties chacune dédiée à une tâche à savoir la gestion des livres, la gestion des réservations et la gestion des utilisateurs. Chaque partie contient un bean associé, un service qui va accéder aux serveurs REST et un servlet qui va contrôler ce service.

Une image contenant texte, intérieur, mur, capture d’écran

Description générée automatiquement

### Class Diagram REST 1

Dans notre REST 1, nous avons deux beans pour les livres et les réservations et deux REST qui vont contrôler chacun leur Worker attribué. Les REST vont principalement implémenter les requêtes pouvant provenir de l’API à savoir les CRUD.

Une image contenant diagramme

Description générée automatiquement

Au niveau de la base de données, nous avons deux tables à savoir t\_livres et t\_reservation. Les livres contiennent une PK, un titre, un auteur, une description, une image et le nombre d’exemplaires disponibles. Les réservations sont composée d’une PK, de la FK du livre, de la FK de l’utilisateur et de la date de retour.

Une image contenant diagramme

Description générée automatiquement

### Class Diagram REST 2

Dans notre RESt2, nous avons un bean pour les utilisateurs et un REST qui va contrôler son Worker.

Une image contenant diagramme

Description générée automatiquement

Au niveau de la base de données, nous avons une table pour les utilisateurs contenant une PK, leur nom, leur mot de passe, s’il est administrateur et s’il est banni.

Une image contenant texte

Description générée automatiquement

# Hébergement

## REST

Les serveurs REST sont hébergés en ligne sur les comptes Tizoo de chacun :

Site de Nathan contenant le REST des livres et des réservations :

http://clapassonn.emf-informatique.ch/javaLivreREST/bibliotheque/livres

Site de Alexandre contenant le REST des utilisateurs :

https://haymozn.emf-informatique.ch/java\_compteREST/bibliotheque/utilisateurs

## APIGateway

Plusieurs moyens ont été testés afin d’héberger l’APIGateway. Etant donné que nous avons utilisé Java 18 et Tomcat 10.1.8, nous avons rencontré des difficultés car nous ne pouvons pas héberger notre API sur les comptes Tizoo ni sur le PC dédié. Pour faire fonctionner notre Gateway, nous avons testé d’installer docker sur le PC dédié et le mettre dedans mais nous avons eu un énorme problème avec les cors. Pour finir, nous avons décidé de la faire tourner en local.

## Client

Pour utiliser le client, nous utilisons le client en local à l’aide de l’extension Live Server de Visual Studio Code. Cette extension nous permet de faire tourner le site comme s’il était placé sur un serveur alors qu’il se trouve en local.

# Installation du projet complet avec les 5 applications

L’ensemble des applications est disponible sur le répertoire github :

https://github.com/AlexandreHaymoz/EMF-Shelf

# Outils, langages (versions, définitions, installations)

Dans le cadre de ce module, nous avons réalisé des projets en Java 8 et 18 avec les logiciels NetBeans et IntelliJ. Afin de faire des tests sur les parties REST et la Gateway, nous avons principalement utilisé Postman. Pour finir, nous avons utilisé de l’HTML et du javascript sur Visual Studio Code afin de faire nos clients.

# Tests de fonctionnement du projet

|  |  |  |  |
| --- | --- | --- | --- |
| Test | Résultat attendu | Résultat attendu | Validé ? |
| Se connecter avec un utilisateur valide et mauvais mot de passe | Erreur 401 + message d’erreur | Erreur 401 + message d’erreur | OK |
| Se connecter avec un utilisateur inexistant | Erreur 401 + message d’erreur | Erreur 401 + message d’erreur | OK |
| Se connecter avec un utilisateur existant et un bon mot de passe | Message indiquant qu’on est connecté | Message indiquant qu’on est connecté | OK |
| Se connecter alors qu’on est déjà connecté | Erreur 403 + message d’erreur | Erreur 403 + message d’erreur | OK |
| Créer un compte avec un nom existant déjà | Erreur 403 + message d’erreur | Erreur 403 + message d’erreur | OK |
| Créer un compte avec des identifiants valides | Message indiquant la réussite de l’enregistrement + retour à l’index | Message indiquant la réussite de l’enregistrement + retour à l’index | OK |
| Créer un compte lorsqu’on est connecté | Erreur 403 + message d’erreur | Erreur 403 + message d’erreur | OK |
| Lister les livres lors de l’arrivée sur le site | Affichage des livres avec un lien cliquable | Affichage des livres avec un lien cliquable | OK |
| Clic sur un lien vers la page d’une livre | Affichage des informations d’un livre lorsqu’on arrive sur sa page | Affichage des informations d’un livre lorsqu’on arrive sur sa page | OK |
| Créer un livre sans être connecté | Erreur 401 + message d’erreur | Erreur 401 + message d’erreur | OK |

# Auto-évaluations et conclusions

## Nathan

En conclusion, j’ai trouvé ce module très intéressant car il nous a appris beaucoup de choses notamment sur les serveurs REST mais j’ai ressenti beaucoup de frustration à cause des problèmes de compatibilité dues aux versions de Java et de Tomcat utilisés qui ont fait des conflits avec nos comptes Tizoo.