IMT Mines Alès
École Mines-Télécom

Juin 2024

Rapport de TP :
Web Services Sécurisés

IMT Mines Ales

Kee-Soon ARMAND
Ines BERRADI
Fanny PETITJEAN

I. Introduction

Le TP se concentre sur la mise en place de plusieurs Web Services et d'une délégation d'authentification. Dans ce rapport, nous allons présenter l'ensemble de la solution mise en place. Nous mettrons en avant les difficultés rencontrées, la façon dont nous les avons surmontées, ainsi que les améliorations possibles. L'objectif principal était de permettre la réservation de vols et d'hôtels via une interface utilisateur moderne et sécurisée, en utilisant des protocoles d'authentification via OpenID Connect pour garantir la sécurité des API.

Au niveau des choix effectués, nous avons décidé de partir sur la création de l'application SOAP. Pour le langage, nous avons choisi JAVA. Ensuite, nous avons décidé d'appeler l'API Google People API.

Nous allons commencer par évoquer la mise en place du déploiement et des tests. Ensuite, nous mettrons en avant la mise en place de l'application SOAP et de l'application REST. Enfin, nous expliquerons la mise en place de l'application Javascript.

II. Explications de déploiements

L'ensemble de nos applications sont sur Docker, Il suffit de faire un "docker-compose up –build" pour lancer les applications. Sur localhost:5500 nous avons la SPA, ensuite le port 8081 nous avons Keycloak, sur le port 8080 l'application REST et le port 8082 pour le SOAP. Nous avons aussi le client du SOAP sur le port 3000. Nous avons aussi une application basic-app que nous avons utilisé pour essayé une deuxième fois keycloak mais nous n'avons pas réussi à le mettre en place.
En allant sur http://localhost:5500/web-app vous aurez une page du Front qui va vous permettre d'accéder à l'authentification Google. Nous avons aussi un formulaire http://localhost:5500/web-app/form.html

Pour Keycloak, pour s'authentifier sur http://locahost:8081/ , le mot de passe et l'utilisateur sont "admin".

III. Application SOAP

Nous avons entrepris de développer une application serveur SOAP en Java nommée `javasoapserver`. Cependant, nous avons rencontré des difficultés dès le début avec la commande Maven de base pour créer un projet Java Web, principalement en raison de problèmes avec les caractères spéciaux tels que les tirets et les guillemets. En effet, après avoir testé à de nombreuses reprises et de façon différentes la commande, elle ne fonctionnait pas. Après avoir identifié cette source de problème, nous avons pu progresser en mettant en place les services et modèles suivants : `HotelRoom`, `HotelReservationService`, et `HotelReservationServiceImpl`. Ces classes sont basiques et ne contiennent pas vraiment de code logique lié aux hôtels. Elles ont été créé pour les modifier par la suite

Ensuite, nous avons ajouté plusieurs dépendances nécessaires pour mettre en œuvre les web services et leurs méthodes associées. Cependant, nous avons constaté que la version de notre OpenJDK était trop récente par rapport aux versions requises par ces dépendances. Cela a provoqué des problèmes supplémentaires que nous avons résolus en ajustant la version de notre OpenJDK à la version 11.

Nous avons ensuite développé une fonction main pour créer une URL complète et déployé le Web Service sur un serveur d'application WildFly en utilisant `javax.xml.ws.Endpoint` pour exposer le service en tant qu'endpoint autonome. Cette URL est cruciale car elle permet d'effectuer des recherches de réservations via notre application. Nous avons donc testé le bon fonctionnement du serveur grâce au WSDL. Nous n'avons pas rencontré de difficultés à ce niveau.

Par la suite, nous avons créé un client nommé `javasoapclient` pour héberger le client SOAP. Nous avons utilisé `wsimport` pour générer les classes utilitaires nécessaires au client SOAP, préférant intégrer le code source dans le projet client pour une configuration plus flexible de l'URL de connexion au Web Service. Nous n'avons pas effectué la partie du côté client qui utilise les classes pour accéder au web service.

IV. Application REST

Nous avons développé notre application REST en utilisant Java avec Spring Boot. Pendant cette phase, nous avons conçu des modèles pour les vols, les compagnies, les dates et les places, ainsi que des services pour gérer les vols et les réservations d'hôtels. Ensuite, nous avons créé un contrôleur pour gérer les opérations relatives aux vols.

Le processus de création de chaque modèle et service était laborieux mais nécessaire pour assurer le bon fonctionnement de l'application. Une fois le développement terminé et l'application fonctionnelle, nous avons entrepris d'utiliser les API de Google Cloud. Nous avons configuré un projet Google Cloud avec un client OAuth 2.0 et créé un utilisateur de test avec une adresse e-mail jetable. L'URL de rappel spécifiée pour l'authentification OAuth 2.0 était la suivante : `http://localhost:8080/callback`. Nous avons également enregistré les informations de client_id et client_secret dans un fichier JSON.

Après la création des identifiants OAuth et des clients sur la plateforme Google Cloud, nous avons ajouté des classes à notre application REST pour gérer l'appel pour obtenir le Token et écouter sur l'adresse de callback. Nous avons essayé de faire cela avec les dépendances de Google. Cependant, avec Spring Boot, nous avons rencontré des problèmes. Nous n'avons donc pas utilisé des servlets, mais des classes de sécurité et de configuration avec Spring Security. Nous avons donc changé les dépendances.

Nous avons rencontré de nombreux problèmes à cette étape. Nous n'arrivions pas à faire un code fonctionnel, rencontrant toujours des problèmes de dépendances incompatibles avec Spring Boot. Une autre problématique concernait les client_id et client_secret. Nous avions bien récupéré un JSON avec les client_id, client_secret et les URL de l'API, cependant, aucune fonction ne permettait de les utiliser directement avec Spring Boot. Par la suite, nous avons enregistré toutes ces informations dans le fichier `application.properties`, mais le code permettant de les utiliser nous retournait une erreur d'inaccessibilité de ces informations. Le programme n'arrivait pas à atteindre et trouver les informations dans le fichier. Nous avons donc décidé de les intégrer directement dans le code qui lance la requête à l'API. Après cette modification, l'accès à la requête était possible. Nous arrivions bien à nous connecter. Cependant, lors de la réception du callback, le code rencontrait une erreur et ne pouvait pas effectuer l'échange du token. Au final, après avoir modifié tout le code, nous avons réussi à faire la connexion. Nous sommes donc passé pour la récupération du token. La récupération du token s'est passé sans problème. Nous avons tout simplement appelé notre api pour récupérer le profil de l'utilisateur. Nous affichons pour l'instant simplement le json sur la page, par la suite,

nous allons rajouter les informations du profil directement pour la réservation des vols et des chambres.

Par la suite, nous avons débuté à créer un docker pour l'API REST. Dans ce docker, nous allons aussi mettre en place keycloak. La création du Docker était moyennement facile, nous avons dû nous rappeler comment le mettre en place. Après avoir revu comment faire, c'est aller assez vite.

Nous sommes donc passé à la création et la mise en place de Docker pour nos applications. Nous avons mis quelques temps à comprendre quelles informations insérer dans le docker-compose.yml et le DockerFile. En effet, la création du conteneur et de KeyCloak nous a pris quelques temps. Nous avons appris que Spring Boot crée un fichier .jar contenant l'ensemble de l'application. Nous avons utilisé ce fichier pour créer le conteneur. Ensuite, nous avons rencontré quelques problèmes avec les ports. En effet, nous avons essayé de changer les ports de notre application, cependant cela ne fonctionnait pas. Nous avons fini par mettre le KeyCloak sur le port 8081 et notre application REST sur le port 8080. Nous avons ajouté un utilisateur et un mot de passe pour l'administrateur.

Nous allons donc créer par la suite toutes les informations sur KeyCloak. Nous avons créé un nouveau Realm avec le nom "test". Ensuite, nous avons ajouté un user de test et deux clients. Notre 1er client se nomme "Client-front", nous lui avons ajouté des url de redirections et des web origins, il est pas authentifier. Ensuite, on a le 2ème client nommé "Client-back" qui est authentifié. Suite à ces créations, nous avons longuement cherché l'adapter utilisé dans Spring Boot pour sécuriser. Nous avons trouvé un adapter directement dans Spring Boot Security. Cependant, nous n'arrivons pas à sécuriser la route, elle nous donnait toujours une erreur 404. Nous avons essayé de faire un nouveau projet java pour tester keycloak.

V. Application Javascript

Nous avons créé une application Web en Javascript. Nous avons ajouté les caractéristiques suivantes :

Caractéristiques	Informations complémentaires
Nom	
Prénom	
Adresse Email	Format avec @
Ville	Ville prédéfinie
Pays	Pays prédéfini
Nombre Adulte	Max 10 personnes
Nombre Adolescent	Max 10 Personnes
Nombre Enfant	Max 10 Personnes
Formule petit-déjeuner	
Type de chambre	Choix prédéfini
Date arrivée	
Date départ	

L'application nous permettra d'afficher les vols et les hôtels pour les clients. Cette application se connecte à l'application REST pour faire toutes les requêtes.