Conception technique

Système de réservation

Projet 12

Koshman Bohdan

Table des matières.

Introduction

Ce document constitue les spécifications techniques du projet Système de réservation.
Un système qui permet de réserver une visite dans un lieu public et de contrôler le nombre de visiteurs simultanés.

La première partie constitue la description du domaine fonctionnel et des différentes entités utilisées dans le projet.

La seconde partie détaille les relations entre les différents composants internes ou externes du système. En effet dans une base de données relationnelle, les données sont regroupées par concept dans des tables et les concepts sont liés les uns aux autres par des relations.

La dernière partie décrit le déploiement du système.

Architecture Technique.

Composants généraux.

Pour faciliter la maintenance, la robustesse, la sécurité, la scalabilité et le développement, on utilise une architecture à base de microservices. Pour réduire les coûts d'investissement de départ et facilité la montée en puissance du système, on utilise **AWS** pour déployer **SysRes**.

La gateway.

Pour publier, gérer, surveiller et sécuriser facilement les **API** du système d'information on utilise le service **Zuul Gateway**. En liaison avec microservice **Authentification** pour effectuer l'identification des utilisateurs.

Le load balancer.

Avec l'accroissement du nombre de clients sur internet, certaines **API s** seront dupliquées pour répartir la charge. Un se charge de répartir la charge sur les différentes instances d'une **API**.

Microservice Authentification.

Authentification-api

C’est un contrôleur RESTful qui communique directement avec la base de données User. Il implémente toutes les opérations de gestion des utilisateurs qu’ils soient clients ou employé. C’est la partie backend qui fait l’intermédiaire entre le frontend et la base de données. Les différentes opérations sont accessibles suivant l’authentification et les droits de l’utilisateur. Seules les personnes habilitées pourront créer des comptes pour les Employés. Le Client pourra créer et modifier son compte mais pas le supprimer. La pile logicielle est la suivante :

- Docker
- Application : **J2EE** (**JDK** 14) / **Apache Maven** (3.6.3)/ **SpringBoot** (2.3.1.RELEASE)

Base de données

Pour des soucis de sécurité, de confidentialité et suivant les obligations de la CNIL il est préférable de séparer les données des utilisateurs des autres données. Cette base de données contiendra en plus des identifiants et des coordonnées des Clients. La pile logicielle est la suivante :

- **Amazon RDS : PostgreSQL** (12)

Microservice Establishment.

Establishment-api

C’est un contrôleur RESTful qui communique directement avec la base de données Establishment. Il implémente toutes les opérations de gestion des lieux publics et des commentaires. C’est la partie backend qui fait l’intermédiaire entre le frontend et la base de données. Les différentes opérations sont accessibles suivant l’authentification et les droits de l’utilisateur.

La pile logicielle est la suivante :

- Application : **Docker** / **J2EE** (**JDK** 14) / **Apache Maven** (3.6.3)/ **SpringBoot** (2.3.1.RELEASE)

Base de données

Pour des soucis de sécurité, de confidentialité et suivant les obligations de la CNIL il est préférable de séparer les données des utilisateurs des autres données. Cette base de données contiendra des lieux publics et des commentaires de clients. La pile logicielle est la suivante :

- **Amazon RDS : PostgreSQL** (12)

Microservice Reservation.

Reservation-api

C’est un contrôleur RESTful qui communique directement avec la base de données Reservation. Il implémente toutes les opérations de gestion des réservations. C’est la partie backend qui fait l’intermédiaire entre le frontend et la base de données. Les différentes opérations sont accessibles suivant l’authentification et les droits de l’utilisateur.

La pile logicielle est la suivante :

- Application : **Docker** /**J2EE** (**JDK** 14) / **Apache Maven** (3.6.3)/ **SpringBoot** (2.3.1.RELEASE)

Base de données

Pour des soucis de sécurité, de confidentialité et suivant les obligations de la CNIL il est préférable de séparer les données. Cette base de données contiendra des lieux publics et des réservations de clients. La pile logicielle est la suivante :

- **Amazon RDS : PostgreSQL** (12)

Microservice Angular.

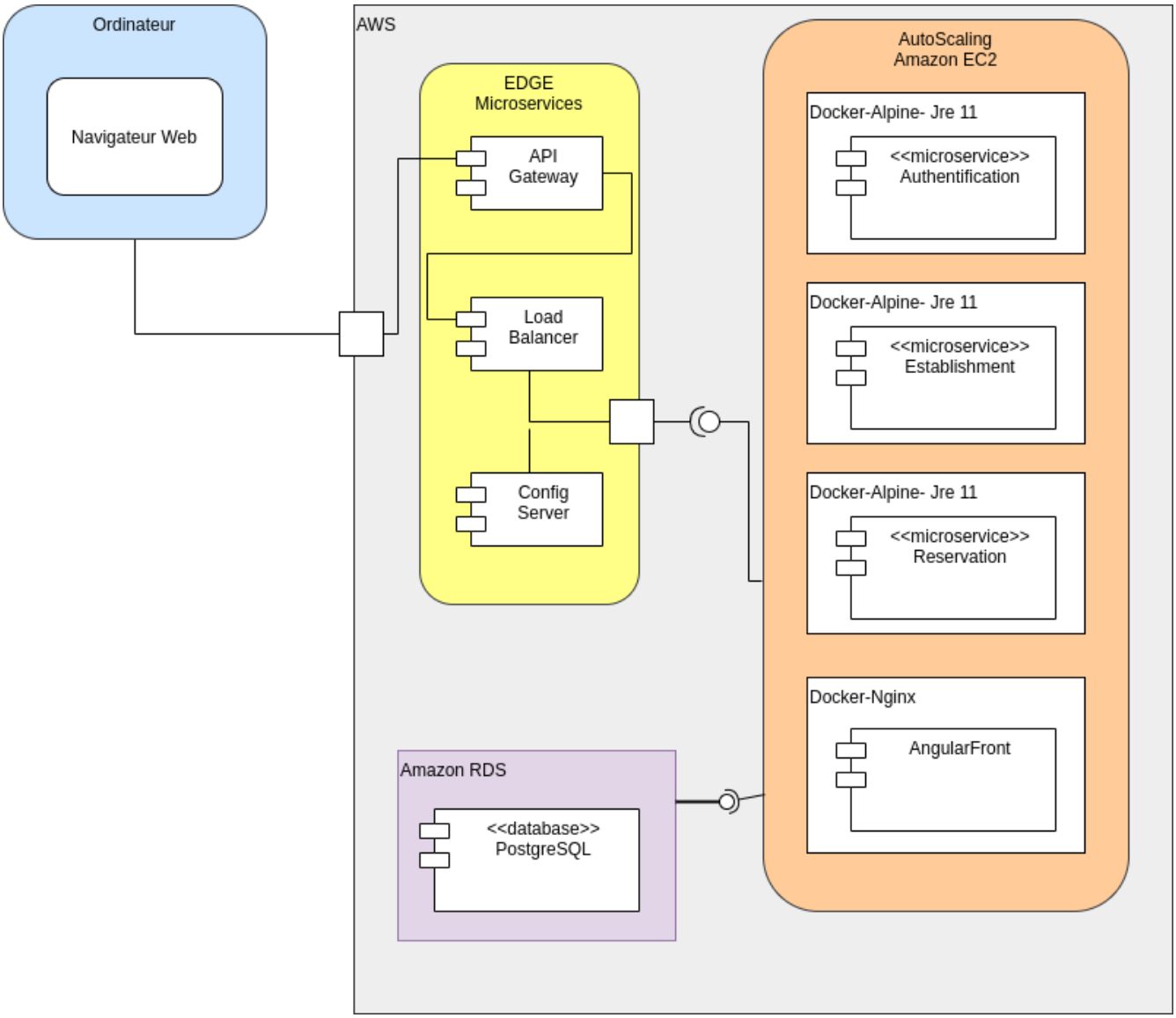
Frontend

C ‘est la partie frontend. Il interagit avec backend (RESTful) pour créer modifier un utilisateur, une réservation ou un lieu.

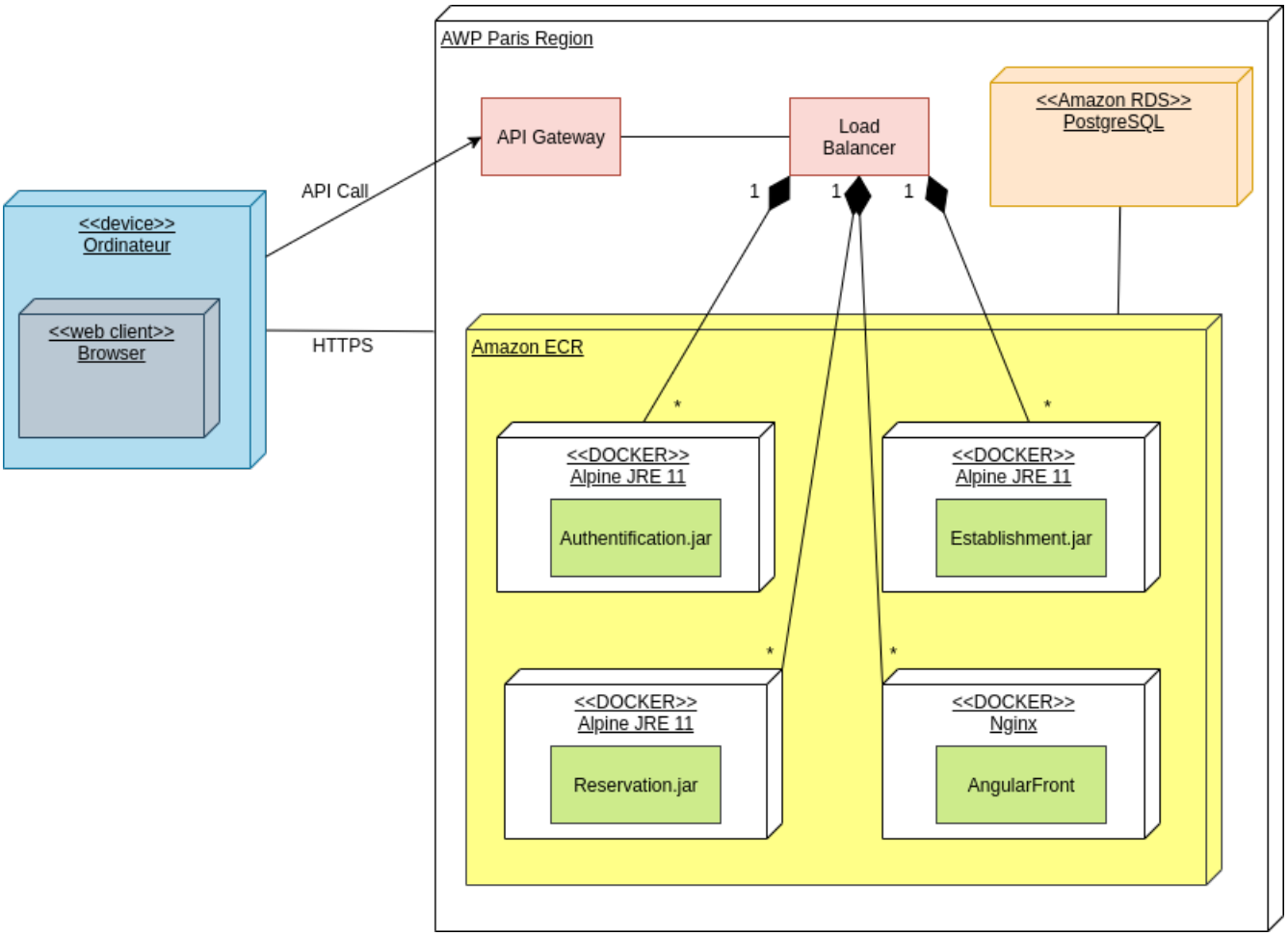
La pile logicielle est la suivante :

- Application : **Docker** /**Angular 10** / **Nginx** / **TypeScript**

Diagramme des composants.



Architecture de déploiement.



L’application est scalable pour répondre à la demande et des load-balancers répartissent la charge sur les différentes instances du même microservice.

AWS permet de spécifier des planificateurs pour le fonctionnement des microservices afin de moduler les instances en fonction de la demande.

On peut aussi répartir les microservices sur plusieurs zones pour répartir le trafic internet et s’étendre à l’international.

Serveur de Base de données.

Pour déployer les bases de données on utilise Amazon RDS **PostgreSQL**.

Comme indiqué plus haut dans ce document, par mesure de sécurité on utilisera une base pour la gestion des utilisateurs.

La base des establishments pourra contenir en outre les informations sur les commentaires.

La base des reservation pourra contenir en outre les informations sur le nombre de réservations enregistrées.

Microservices.

Chaque microservice sera déployé sur une instance **EC2**. On utilisera Docker Linux Alpin (64bits) qui comporte les outils Java :

- **Docker**
- **Java** version (14)
- **OpenJDK Runtime Environment**
- OpenJDK 64-Bit VM

Architecture logicielle.

Principes généraux.

Le versionnage du projet est effectué avec **Git Flow** et le code source est conservé sur repository privé **GitHub**. Le code est implémenté en **Java2EE**, les dépendances et le packaging sont effectués par **Apache Maven**. On utilise le Framework open source **Spring Boot** pour construire et définir l'infrastructure de l'application.

Les modules.

On utilise **Apache Maven** pour le packaging de l'application. Chaque microservice est dans un conteneur **Docker** et constitue un module de l'application. On obtient les modules suivants :

- eureka
- zuul
- reservation
- establishment
- authentification
- angularFront

Structure des sources.

Pour alléger les diagrammes, les sous-répertoires test ne seront pas représentés. La structure des répertoires **test/java** est la même que celle des répertoires **main/java** de chaque module. Un fichier **README.md** décrira le contenu de chaque module. Le fichier **application.properties** dans les dossiers **test/ressources** contiennent toutes les propriétés nécessaires à la phase de test. Chaque module aura son fichier de configuration pom.xml pour **Apache Maven** et son fichier d'intégration **.travis.yml** pour **GitHub**. La structuration des répertoires du projet suit la logique suivante de façon à respecter la philosophie **Maven**. On aura un répertoire par module et les fichiers statics pour les applications seront centralisés au même endroit. Le fichier pom.xml principal contiendra toutes les déclarations des versions pour les dépendances.

```
SysRes
├─ <eureka>
├─ <zuul>
├─ <establishment>
├─ <authentification>
├─ <reservation>
├─ <angularFront>
├─ static
│   ├─ css
│   ├─ images
│   └─ js
│
├─ pom.xml
├─ travis.yml
├─ docker-compose.yml
└─ README.md
```

Microservice Authentification.

Les couches.

Les microservices backend sont des contrôleurs **RESTful** avec lesquels les autres API communiquent. Ils sont découpés suivant les couches suivantes :

- La couche **model** qui contient les modèles des objets métiers (Entity).
- La couche **DTO** (**Data Transfer Object**) qui contient la définition des modèles transféré entre les différentes API.
- La couche **controller** qui expose des **endpoints** aux autres **API**, consomme et produit des **DTO** sous forme de **JSON** (**JavaScript Object Notation**).
- La couche **business** qui fait la liaison entre le **controller** et la base de données en manipulant les **DTOs** et les **Entitys**.

Structure des sources

```
├─ <authentification>
│    ├─ src
│    │   ├─ main
│    │   │   ├─ java
│    │   │   │   └─ com.sys.authentification
│    │   │   │       ├─ configuration
│    │   │   │       ├─ controller
│    │   │   │       ├─ dto
│    │   │   │       ├─ exception
│    │   │   │       ├─ model
│    │   │   │       ├─ repository
│    │   │   │       ├─ service
│    │   │   │       └─ AuthentificationApplication.java
│    │   │   └─ resources
│    │   │       └─ bootstrap.properties
│    │   └─ test
│    │       ├─ java
│    │       └─ ressources
│    │           └─ application.properties
│    │
│    ├─ .gitignore
│    ├─ Dockerfile
│    ├─ pom.xml
│    └─ README.md
```

Microservice Establishment.

Les couches.

Les microservices backend sont des contrôleurs **RESTful** avec lesquels les autres API communiquent. Ils sont découpés suivant les couches suivantes :

- La couche **model** qui contient les modèles des objets métiers (Entity).
- La couche **DTO** (**Data Transfer Object**) qui contient la définition des modèles transféré entre les différentes API.
- La couche **controller** qui expose des **endpoints** aux autres **API**, consomme et produit des **DTO** sous forme de **JSON** (**JavaScript Object Notation**).
- La couche **business** qui fait la liaison entre le **controller** et la base de données en manipulant les **DTOs** et les **Entitys**.

Structure des sources

```
├─ <establishment>
│    ├─ src
│    │   ├─ main
│    │   │   ├─ java
│    │   │   │   └─ com.sys.establishment
│    │   │   │       ├─ configuration
│    │   │   │       ├─ controller
│    │   │   │       ├─ dto
│    │   │   │       ├─ exception
│    │   │   │       ├─ model
│    │   │   │       ├─ repository
│    │   │   │       ├─ service
│    │   │   │       └─ EstablishmentApplication.java
│    │   │   └─ resources
│    │   │       └─ bootstrap.properties
│    │   └─ test
│    │       ├─ java
│    │       └─ ressources
│    │           └─ application.properties
│    │
│    ├─ .gitignore
│    ├─ Dockerfile
│    ├─ pom.xml
│    └─ README.md
```

Microservice Reservation.

Les couches.

Les microservices backend sont des contrôleurs **RESTful** avec lesquels les autres API communiquent. Ils sont découpés suivant les couches suivantes :

- La couche **model** qui contient les modèles des objets métiers (Entity).
- La couche **DTO** (**Data Transfer Object**) qui contient la définition des modèles transféré entre les différentes API.
- La couche **controller** qui expose des **endpoints** aux autres **API**, consomme et produit des **DTO** sous forme de **JSON** (**JavaScript Object Notation**).
- La couche **business** qui fait la liaison entre le **controller** et la base de données en manipulant les **DTOs** et les **Entitys**.

Structure des sources

```
├─ <reservation>
│    ├─ src
│    │   ├─ main
│    │   │   ├─ java
│    │   │   │   └─ com.sys.reservation
│    │   │   │       ├─ configuration
│    │   │   │       ├─ controller
│    │   │   │       ├─ dto
│    │   │   │       ├─ exception
│    │   │   │       ├─ model
│    │   │   │       ├─ repository
│    │   │   │       ├─ service
│    │   │   │       └─ ReservationApplication.java
│    │   │   └─ resources
│    │   │       └─ bootstrap.properties
│    │   └─ test
│    │       ├─ java
│    │       └─ ressources
│    │           └─ application.properties
│    │
│    ├─ .gitignore
│    ├─ Dockerfile
│    ├─ pom.xml
│    └─ README.md
```

Microservice AngularFront.

Les couches.

Les module frontend forment la partie visible par les utilisateurs qui présente une interface graphique. Ils communiquent avec le backend en utilisant des requêtes HTTPS.

Les couches suivantes :

- La couche **vue** qui constitue l'interface entre l'utilisateur et l'application et permet de saisir et afficher les données des **DTO**.
- La couche **DTO** (**Data Transfer Object**) qui contient la définition des modèles transféré entre les différentes API.
- La couche **controller** qui expose des **endpoints** et permet d'exécuter les opérations déclenchées par la couche **vue**.

Structure des sources

```
├─ <angularFront>
│    ├─ src
│    │   ├─ app
│    │   │   ├─ admin
│    │   │   │   ├─ dashboard
│    │   │   │   ├─ info
│    │   │   │   ├─ shared
│    │   │   │   └─ admin.module.ts
│    │   │   ├─ employee
│    │   │   │   └─ employee.module.ts
│    │   │   ├─ shared
│    │   │   ├─ services
│    │   │   ├─ my-reservation
│    │   │   ├─ app.component.ts
│    │   │   ├─ app.module.ts
│    │   │   └─ app-routing.module.ts
│    │   ├─ assets
│    │   │   └─ img
│    │   └─ environment
│    │
│    ├─ .gitignore
│    ├─  Docker
│    ├─  nginx.conf
│    ├─  angular.json
```

Environnement de développement.

Le choix de l’environnement de développement reste libre à la charge des développeurs. Toutefois, **IntelliJ IDEA** version 2020.1 intègre de nombreux plugins pour faciliter la production de code en J2EE avec :

- **Git Flow Integration** 0.7.2 de Opher Vishnia pour l'utilisation de **Git Flow**.
- **SonarLint** 4.7.0.17141 de SonarSource pour la vérification du code.
- **AWS Toolkit** pour une intégration de **AWS CLI**.
- **Maven**, **Markdown**, **Properties**, **Resource Bundle Editor**, **Git**.

Glossaire.

AMI	(**Amazon Machine Image**) logiciel d'exploitation Amazone de type Linux.
AWS	(**Amazon Web Services**) service internet d'Amazon.
CNIL	(**Commission nationale de l'informatique et des libertés**).
DTO	(**Data Transfer Object**) type d'objet permettant de transférer des données.
EC2	(**Elastic Cloud Compute**) serveur de base permettant d'intégrer de nombreux systèmes.
JDK	(**Java Developer Kit**) outils de développement du langage **Java**.
JRE	(**Java Runtime Environment**) outils pour exécuter un exécutable **Java**.
load-balancing	Action de répartir la charge entre plusieurs instances d'une même application.