ECF Le Quai Antique

1) Une architecture N-tiers Mariadb – Spring Boot - Angular

Il existe un grand nombre de moyens de produire une application web, dépendant des caractéristiques du client. Sans possibilité de parler au client et de discuter avec lui des différentes possibilités (puisqu'il est fictif), le choix s'est reposé sur les quelques informations glanées dans la consigne et l'annexe.

Nous savons donc :

- que le site implémente des foncitonnalités d'un site dynamique, tels que la création de session pour un utilisateur, dont le contenu offert et postés via le site interagit avec une base de données. Des données, telles que les réservations, sont autant pertinentes à connaître pour les clients du restaurant que pour les employés du restaurant, qui utiliseront ces données pour préparer la réservation d'un client. Cela veut dire qu'à terme, les données utilisées et postées par ce site pourront également pour une autre infrastructure exploitant ces données. Cela implique donc que le Backend de l'application soit utilisées pour d'autres sites que le site web publique, et donc qu'un Back sous forme de RestAPI accompagné d'un front soit plus pertinent qu'un site basé sur du server-side-rendering.
- que le site est celui d'un restaurant, et se trouve donc être l'une des façades d'un commerce , mais non sa principale façade. Additionnellement, cela veut aussi dire qu'il sera répertorié par Google dans Maps lorsqu'un utilisateur cherchera un restaurant dans la région, ou tapera son nom dans la barre de recherche. Cela amoindri donc la nécessité d'être bien répertorié dans les résultats de recherche Google, par rapport par exemple à un site de e-commerce. Cela ouvre donc la possibilité d'avoir un front en Single Page Application.
- que le commerçant ouvre un troisième restaurant, qu'il en possède donc déjà deux, et qu'il pourra potentiellement en ouvrir d'autres. Cela demande donc d'avoir une application avec une bonne scalabilité.

Sachant cela, et étant donné nos compétences, nous avons choisit une architecture N-tiers, avec les spécifications suivantes :

Serveur :

- **AWS :** Propose une solution gratuite d'hébergement cloud de PVS sous condition, ce qui permet de déployer une applilcation Spring Boot contrairement à un hébergeur comme Netlify. De plus, c'est un hébergeur très utilisé et donc une compétence intéressante à mettre dans le CV.

- **Ubuntu 20 :** Solution Lightweight permettant de mettre en ligne une application peu couteuse en ressource gratutitement sur AWS, donc très intéressant.

Base de données :

- **Mysql :** Étant donné que certaines données étaient liées par des relations pouvant être complexe (utilisateur est lié à une table allergene lié à une table plat pour permettre de détecter les plats auxquels l'utilisateur pourrait être allergique), une base de données relationnelle nous a paru pertinente.
L'application ne va pas stocker une quantité de données très importante, ne demande pas nécessairement d'être extrêmement performante, inutile donc d'aller chercher des solutions

propriétaires comme Oracle. On choisit donc une base de données relationnelle Open Source, et pour des questions de préférences personnelles, notre choix s'est porté sur Mariadb

API :

- Java : Le fait de créer une API pour un projet avec une bonne scalabilité sans gros soucis de performance et fait par un seul développeur avec des exigences de temps très contraignantes demande à notre sens un langage avec certaines caractéristiques. D'abord, il doit être fortement typé, de façon à ce qu'un autre développeur qui aurait été engagé pour étendre l'application qui ne saurait être mise en production en 70 heures de travail par un développeur puisse savoir exactement le type des paramètres et du retour des fonctions qu'il va utiliser. Ensuite, pour un projet scalable et potentiellement repris plus tard par d'autres développeurs, un langage dont l'utilisation est soumise à une philosophie très rigide quant à la façon dont le code doit être organisé semble être une bonne idée. Enfin, le langage doit avoir un framework web reconnu comme très efficace pour créer une API. C'est donc la raison pour laquelle, à notre sens, le meilleur choix pour ce projet est le C#. Mais étant donné que le marché nantais est très porté Java, et donc que nous avons une bien plus grande expérience en Java qu'en C#, ainsi qu'un accès à IntelliJ Ultimate, nous avons décidé de choisir le Java.

- Spring Boot avec Maven : Spring est un peu le choix par défaut du développeur web utilisant Java pour le Back. Possède un système d'annotations et d'injections de dépendances rendant très rapide la création de RestController. Maven est choix personnel lié à mon apprentissage de technologies adaptées au marché nantais.

Dépendances de l'API :

- JPA : ORM permettant de faciliter la transition de données en base à des objets Java

- Mapstruct : Outil permettant de faciliter la sérialisation d'objets Java pour transformer des objets d'une certaine classe en objet d'une autre classe similaire. Utile pour renvoyer récupérer des DTO des entités Java.

- Mariadb.jdbc : dépendance permettant de gérer l'accès à une base de données Mariadb via JDBC (qui est une dépendance de JPA)

- Spring boot starter security : dépendance permettant de sécuriser les sessions utilisateurs.

- **JsonWebToken :** outil permettant de gérer les sessions via la création et la vérification de JWT.

- Lombok : outil de métaprogrammation permettant d'éviter d'avoir à créer du code redondant, comme les getters, les setters, les constructeurs, ou les classes builder.

- Slf4j : outil de log.

Front-End :

- SASS : Simplement du CSS plus pratique d'utilisation.

- **Angular :** Framework JS connu pour sa scalabilité, sa structure rigide orientée objet avec injection de dépendances et son utilisation native de typescript. Choisi pour les même raisons que Java. Possède en plus un CLI très efficace.

- **ngBootstrap :** Permet la création facile de carousel, et un design rapide des interfaces avec lesquels les utilisateurs du site ne visitent pas comme le panel d'adminstration.

- **RXJS :** Permet de simplifier le flow des données dans l'application via le système de souscription.

2) Récit d'utilisation du site

En tant que visiteur, j'accède à la page d'accueil lorsque que je me connecte sur le site. La première que je peux voir, en dessous de la barre de navigation, est un carousel des photos des plats. A coté, un slogan, une brêve description du restaurant, et un bouton me permettant de réserver. Si je descend, je peux observer les horaires du restaurant. Les horaires du jour sont également disponibles dans le footer de la page.

Curieux de voir la carte avant de réserver, je peux consulter, grâce à la barre de navigation, la carte des plats et menus du restaurant. Sont alors disponibles un carousel des plats, que je peux trier par type (entrée, plat et dessert), ainsi qu'en dessous, une liste sélectionnable des menus. Lorsque je sélectionne un menu, je peux voir apparaître une liste de formules avec chacune une liste de plats. En dessous de tout cela, je peux voir un carousel présentant tous les plats du menu actuellement sélectionné.

Ensuite, je peux accéder à un onglet me permettant de réserver une ou plusieurs places dans le restaurant.
En choisissant une date ainsi que le moment de la journée où je souhaite réserver (midi ou soir), j'ai alors accès au nombre de chaises disponibles dans les places restantes. Une place est une entité pouvant représenter une table ou toute entité possédant des chaises disposé à la réservation (par exemple, une table de douze pourrait très bien être constitué de deux entités de 6 chaises). En précisant mon identité, le nombre de convives m'accompagnant et les allergies de mon groupe, je peux ensuite cliquer sur réserver, ce qui m'ouvrira une plage d'horaires disponibles. En cliquant sur l'une d'elle, j'envoie une demande de réservation, qui sera prise en compte par le serveur. Il réservera automatiquement les places permettant d'optimiser le nombre de chaises restantes par rapport au nombre de convives de ma réservation ainsi que le nombre de places occupées. Par exemple, si je fais une réservation pour sept, qu'il y a deux place de quatre chaise disponibles, une place de huit et une place de douze, il privilégiera la réservation de la place de huit chaises.
Tout cela se fait sans contrôle d'identité (il était précisé dans le cahier des charges qu'un utilisateur pouvait faire cela sans créer de compte, et le patron était injoignable, donc la fonctionalité a été implémentée telle quelle).

Enfin, je peux également créer un compte. Après avoir entrer une adresse mail (potentiellement la mienne, mais cela n'est pas vérifié, car il serait absurde d'implémenter ce niveau de sécurité alors qu'il a été demandé d'implémenter une fonctionalité qui permet de faire des réservations sans avoir vérifier son identité), mon compte est alors créé, et je suis automatiquement connecté en tant que client. La connexion est sécurisée, car l'administrateur peut également se connecter via ce formulaire et je ne dois pas pouvoir définir mon rôle en tant qu'administrateur.

Si j'avais déjà un compte, je peux me connecter. Pour cela, je rentre mon adresse mail et mon mot de passe. S'il est bon, je deviens client. Sinon, je dois corriger le formulaire pour qu'il corresponde à un client bien inséré en base de données.

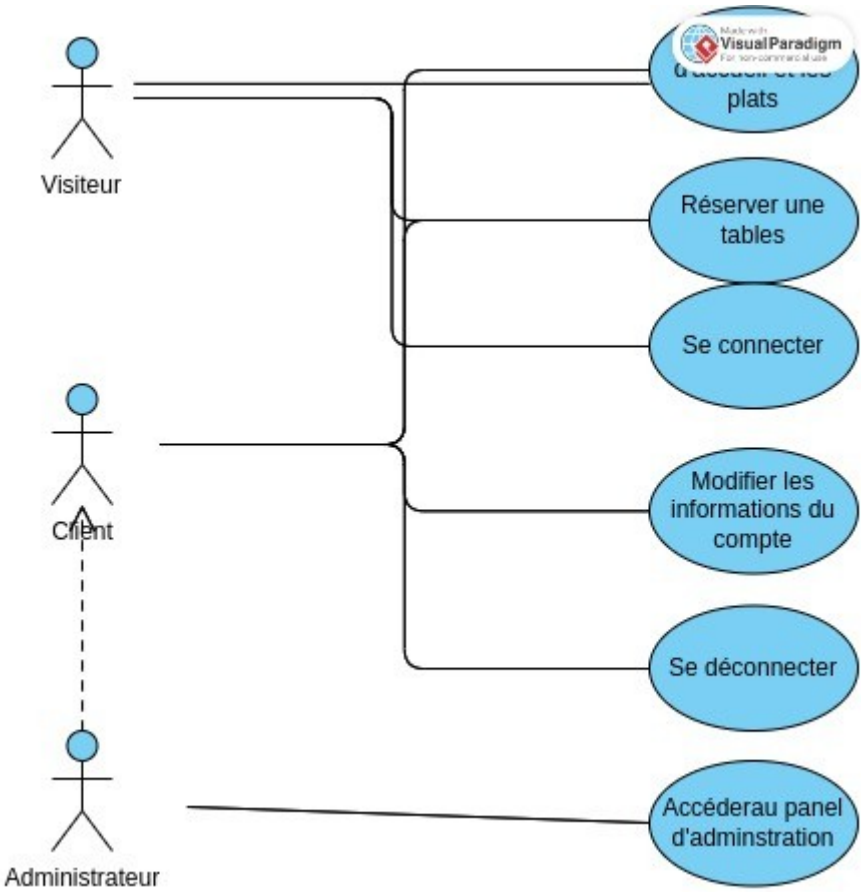
En tant que client, j'ai accès à une page « mon compte » via la barre de navigation, me permettant de remplir, via un formulaire, des informations à propos de mes allergies et du nombre de convives par défaut de mes réservations.

Maintenant, lorsque je consulterait la carte, si un des plats ou une des formules contient un allergene auquel je suis allergique, cela sera précisé par un message d'avertissement.

Enfin, je peux, via la barre de navigation, me déconnecter.

En tant qu'administrateur, j'ai accès au panel d'adminstration. Ainsi, je peux consulter les informations sur le restaurant (description, horaires, nombre de convives maximum autorisés), les menus, les formules et les plats, et les changer, ce qui permet de changer le contenu du site. Il pourrait être pertinent de changer d'autres informations, comme les places disponibles dans le restaurant, ou permettre la création de nouveaux menus, plats, formules et places, mais cela n'a pas encore été implémenté.

Résumé en diagramme d'utilisation :



Fonctionnement de la réservation :

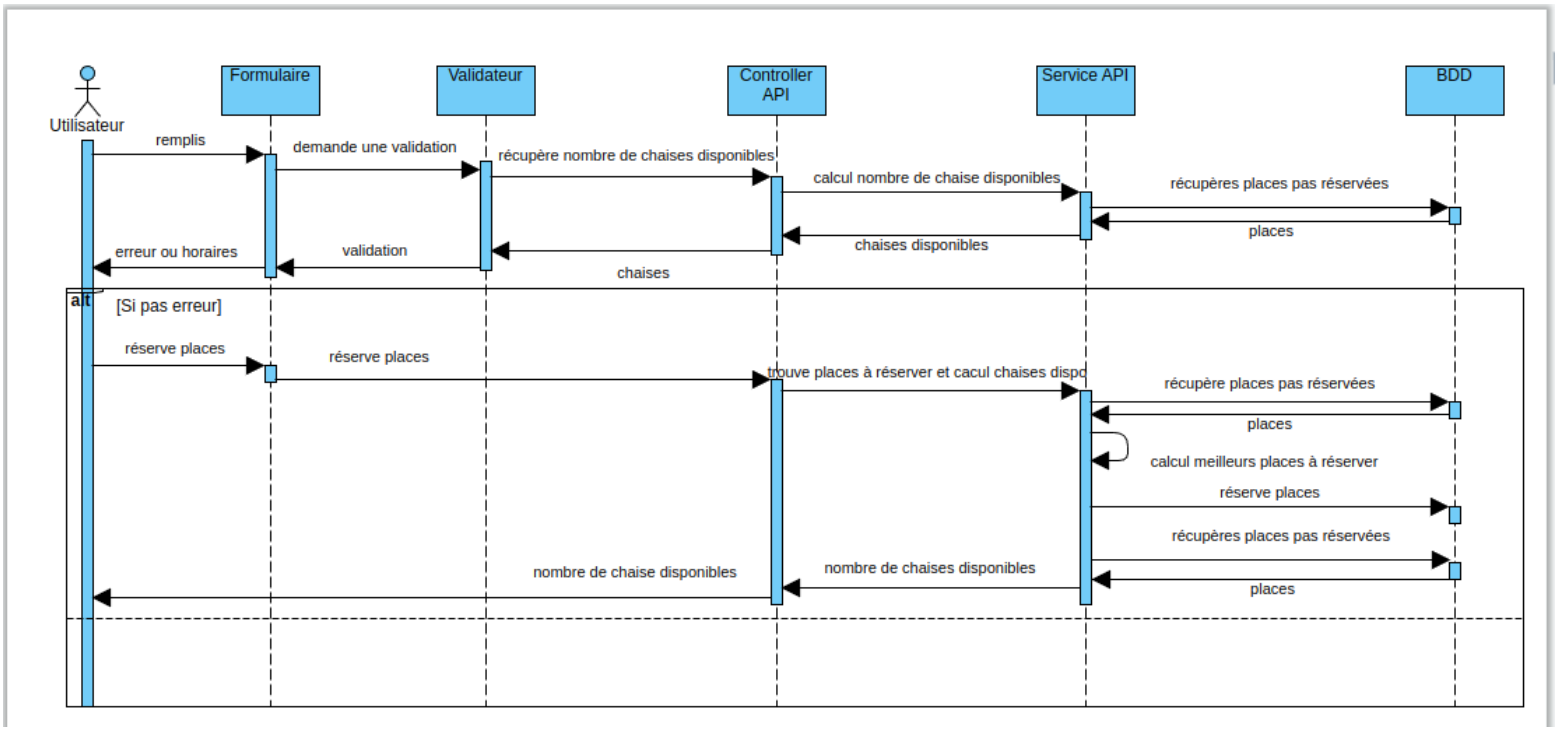


Diagramme UML :