Document d'Architecture Technique (DAT)

MARMITISE

Nicolas Germani

SOMMAIRE

1/ Introduction

Marmitise est un site web réalisé dans le cadre du projet web donné aux étudiants d'IG3 de Polytech Montpellier durant l'année scolaire 2022-2023. C'est un site où les visiteurs pourront retrouver des recettes de cocktails et aussi trouver les bars qui proposent ces-derniers. De plus si le visiteur s'inscrit et se connecte, il pourra lui aussi proposer ses recettes de cocktails à la communauté.

Il y a aussi une version quand l'administrateur se connecte. Il pourra supprimer des utilisateurs, ajouter, modifier et supprimer des alcools, des softs, des ingrédients, des cocktails ainsi que des bars.

2/ Vue d'ensemble de l'architecture

a) Description générale de l'architecture

L'architecture de l'application repose sur une stack MERN (MongoDB, Express.js, ReactJS, Node.js), offrant une approche moderne pour le développement d'applications web. Voici une description des composants clés de l'architecture :

1/ Frontend (interface utilisateur)

- Développé en utilisant ReactJS, une bibliothèque JavaScript pour la construction d'interfaces utilisateur interactives.
- Utilise HTML, CSS et JavaScript/JSX pour la création de composants réutilisables.
- Communique avec l'API REST pour l'échange de données via des appels AJAX.

2/ API REST (backend)

- Développée avec Node.js et Express.js, fournissant une architecture légère et évolutive pour les API RESTful.
- Gère les requêtes HTTP (GET, POST, PUT, DELETE) pour les opérations CRUD (Create, Read, Update, Delete) sur les ressources.
- Utilise les modules npm tels que « express », <moogose »pour interagir avec la base de données MongoDB.

3/ Base de données MongoDB

- MongoDB, une base de données NoSQL orientée document, pour stocker les données de l'application.

- Les schémas de données sont définis à l'aide de Mongoose, une bibliothèque d'objets pour Node.js qui facilite la modélisation des données MongoDB.

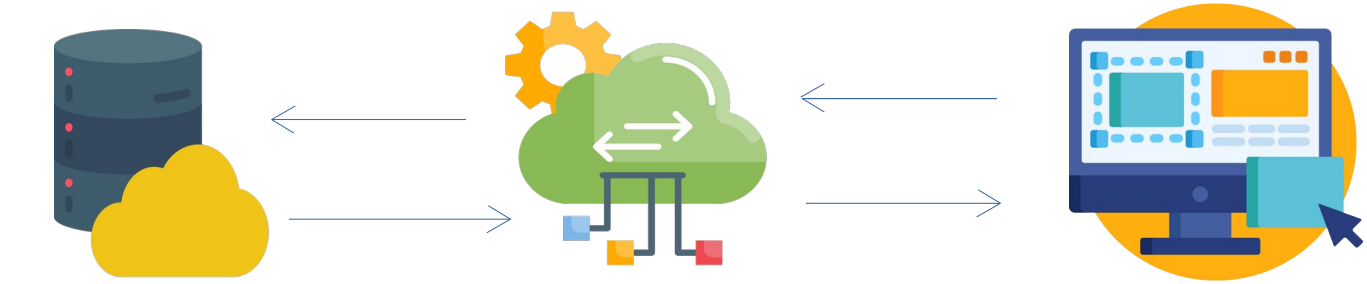
4/ Serveur Node.js

- Héberge l'API REST et interagit avec la base de données MongoDB.
- Gère les requêtes entrantes, les routages et les opérations de validation/authentification si nécessaire.

5/ Déploiement

- L'application est déployée sur une infrastructure cloud (par exemple, AWS, Azure, Google Cloud) ou sur des serveurs dédiés.

b) Schéma



Base de données MongoDB

API REST

Marmitise

3/ Technologies utilisées

a) Liste des technologies, frameworks et outils utilisés pour le développement du projet

- MongoDB : Base de données NoSQL orientée document pour le stockage des données.
- ReactJS : Bibliothèque JavaScript pour la construction d'interfaces utilisateur réactives et dynamiques.
- Node.js : Environnement d'exécution JavaScript côté serveur, utilisé pour le développement de l'API REST.
- Express.js : Framework minimaliste pour la création d'applications web avec Node.js.
- Mongoose : Bibliothèque d'objets pour Node.js, utilisée pour la modélisation et l'interaction avec la base de données MongoDB.
- HTML/CSS : Langages de balisage et de style pour la structure et la présentation de l'interface utilisateur.
- JavaScript/JSX : Langage de programmation utilisé pour la logique et l'interactivité de l'interface utilisateur.
- npm : Gestionnaire de paquets JavaScript utilisé pour installer et gérer les dépendances du projet.
- Postman : Outil de test d'API pour l'envoi de requêtes HTTP et la vérification des réponses.

b) Justification des choix technologiques

MongoDB : J’ai choisi MongoDB en raison de sa flexibilité en tant que base de données NoSQL orientée document. Cela permet de stocker des données de différentes structures sans avoir à définir des schémas rigides à l'avance. Cela facilite également la mise à l'échelle et l'adaptation aux changements de besoins.

ReactJS : J’ai opté pour ReactJS en raison de sa popularité, de sa grande communauté de développeurs et de sa richesse en fonctionnalités. ReactJS facilite la création d'interfaces utilisateur réactives, modulaires et performantes, grâce à son approche basée sur les composants. C’est aussi une bibliothèque que je connaissais et maîtrisais déjà.

Node.js : J’ai choisi Node.js pour le développement de l'API REST en raison de sa rapidité, de sa facilité de prise en main et de son écosystème riche en modules et en outils. De plus, Node.js permet de construire des applications web rapides et évolutives.

Express.js : J’ai utilisé Express.js en raison de sa simplicité et de sa légèreté. Il fournit une couche d'abstraction simple au-dessus de Node.js pour gérer les routes, les middlewares et les requêtes HTTP, ce qui facilite le développement rapide de l'API REST.

Mongoose : J’ai choisi Mongoose pour interagir avec MongoDB car il simplifie la modélisation des données, les opérations CRUD et la validation des schémas. Mongoose offre également des fonctionnalités supplémentaires telles que les hooks, les méthodes personnalisées et les requêtes avancées.

HTML/CSS/JavaScript/JSX : Ces technologies standard pour le développement web ont été utilisées pour créer une interface utilisateur interactive, bien structurée et esthétiquement agréable.

npm : npm est un gestionnaire de paquets très populaire dans l'écosystème JavaScript, offrant un accès facile à de nombreuses bibliothèques et modules utiles. Il m’a facilité la gestion des dépendances et la mise à jour des packages nécessaires au projet.

Postman : Postman est un outil très pratique pour tester et déboguer les API REST. Il m’a permis d'envoyer des requêtes HTTP aux endpoints de l'API, d'inspecter les réponses et de vérifier le bon fonctionnement des fonctionnalités.

Ces choix technologiques ont été faits en tenant compte de la popularité, de la maturité, de la documentation, de la communauté de soutien et de la compatibilité avec les exigences spécifiques du projet web.

4/ Architecture du système

Frontend (Interface utilisateur) :
Le frontend de l'application est responsable de l'affichage des données et de l'interaction avec l'utilisateur.
Le frontend communique avec l'API REST pour récupérer et envoyer des données via des appels AJAX.
Il utilise des composants réutilisables pour une structure modulaire de l'interface utilisateur.

API REST (backend) :
Le backend de l'application gère les requêtes HTTP entrantes et les routes correspondantes pour les opérations CRUD (Create, Read, Update, Delete) sur les ressources.
Les endpoints de l'API sont conçus pour interagir avec la base de données et répondre aux requêtes du frontend.
Il assure également la validation des données, l'authentification, l'autorisation et la sécurité de l’API.

Base de données MongoDB :
Les données sont organisées en collections et les opérations de lecture, d'écriture, de mise à jour et de suppression sont effectuées à l'aide de requêtes MongoDB.

Serveur Node.js :
Le serveur Node.js héberge l'API REST et interagit avec la base de données MongoDB.
Il reçoit les requêtes HTTP du frontend, les traite en fonction des routes définies et renvoie les réponses correspondantes.
Le serveur Node.js gère également les opérations d'authentification, d'autorisation

5/ Conception de la base de données

a) Schémas de base de données (tables, relations, contraintes)

Collection "ALCOOL":

- id: Identifiant unique de l'alcool
- nom: Nom de l'alcool
- degre : Pourcentage d’alcool
- précision: Précision sur l'alcool
- date_fabrication: Année de fabrication de l'alcool

Collection "SOFT":

- id: Identifiant unique du soft
- nomSoft: Nom du soft

Collection "INGREDIENT":

- id: Identifiant unique de l'ingrédient
- nomIngredient: Nom de l'ingrédient

Collection "COCKTAIL":

- id: Identifiant unique du cocktail
- nom: Nom du cocktail
- alcools: Tableau de tupple contenant une (ou plusieurs) référence à la collection "ALCOOL" (id) + une quantité (en centilitre)
- softs: Tableau de tupple contenant une (ou plusieurs) référence à la collection "SOFT" (id) + une quantité (en centilitre)
- ingredients: Tableau contenant (ou plusieurs) référence à la collection "INGREDIENT" (id)

Collection "BAR":

- id: Identifiant unique du bar
- Nom: Nom du bar
- localisation: Localisation du bar
- cocktails: Tableau contenant une (ou plusieurs) référence à la collection "COCKTAIL" (id)

Collection "USER":

- id: Identifiant unique de l'utilisateur
- email: Adresse e-mail de l'utilisateur
- password: Mot de passe de l'utilisateur
- admin: Rôle d'administrateur de l'utilisateur (par défaut, false)

Il y a plusieurs contraintes, sur la table :

- ALCOOL :
  - doit obligatoirement avoir un nom
  - doit obligatoirement avoir un degré

- SOFT :
  - doit obligatoirement avoir un nom

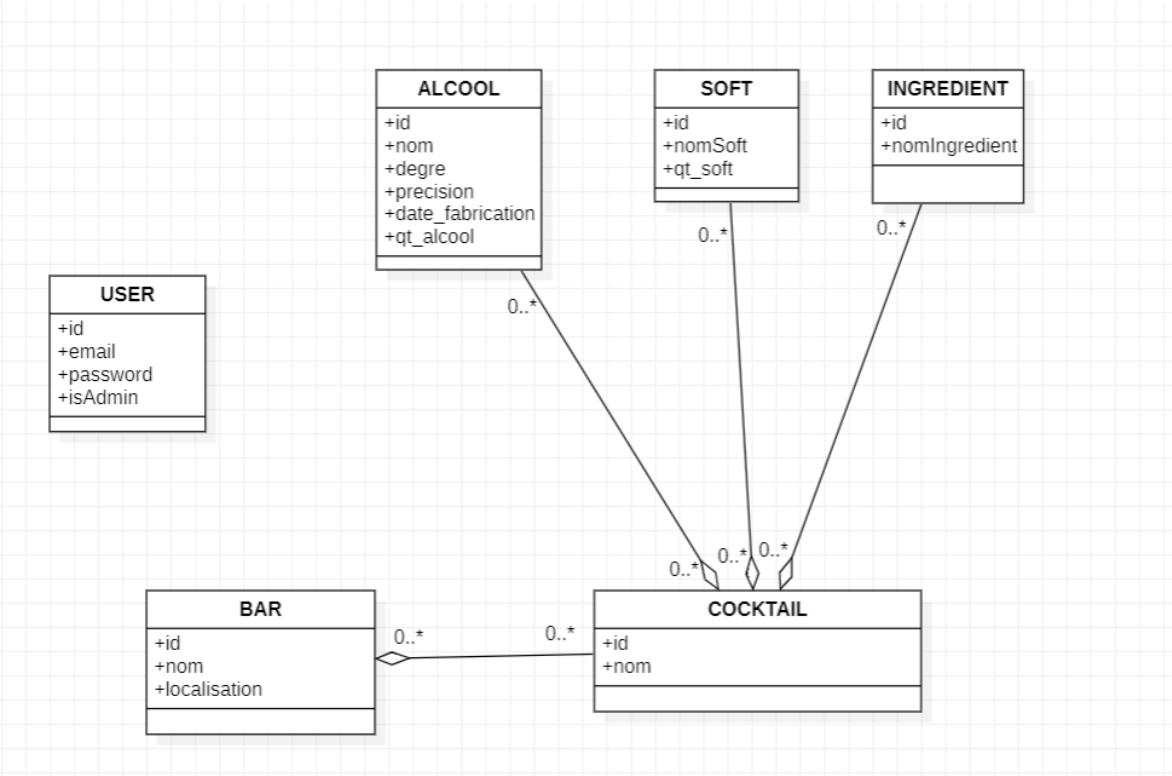
- INGREDIENT :
  - doit obligatoirement avoir un nom

- COCKTAIL
  - doit obligatoirement avoir un nom
  - doit contenir au moins 2 alcools ou au moins 2 softs ou au moins 1 alcool et 1 soft

- BAR
  - doit obligatoirement avoir un nom
  - doit obligatoirement avoir une localisation
  - doit obligatoirement proposer un alcool de la base de données

- USER
  - doit obligatoirement avoir un mail
  - doit obligatoirement avoir un mot de passe
  - lors de l'enregistrement, admin est automatiquement mis à « false »

b) Diagramme de la base de données



c) Explication des choix de bases de données (MongoDB) et de son utilisation spécifique dans le projet.

MongoDB a été choisi comme base de données pour ce projet en raison de ses caractéristiques clés :

- Flexibilité du schéma : MongoDB est une base de données NoSQL orientée document, ce qui signifie qu'elle n'impose pas de schéma rigide. Cela permet d'ajouter, de modifier ou de supprimer des champs sans avoir à modifier toute la structure de la base de données. Cela convient bien à un projet où les cocktails et les bars peuvent avoir des informations variables.

- Performance : MongoDB est connu pour sa rapidité et ses performances élevées. Il utilise des index et des requêtes optimisées pour des opérations de lecture et d'écriture rapides, ce qui est important pour une application web qui nécessite des temps de réponse courts.
  De plus, il s'agit d'un site proposant des recettes, une recette, une fois crée, n'est pas censée être modifiée ou si elle l'est, c'est très rare.

De plus, en utilisant MongoDB dans ce projet, j'ai pu bénéficier de la flexibilité du schéma, de bonnes performances et de la possibilité de faire évoluer facilement la base de données au fur et à mesure de l'avancement de l'application.
Une des première raison pourquoi j'ai choisi MongoDB est que c'est une technologie que je ne connaissais pas et ce projet était une opportunité d'essayer et apprendre son fonctionnement.

6/ Architecture des interfaces

L'application aura toujours la même interface, la même barre de navigation en haut et les éléments en-dessous. Il y a plusieurs vues :

- Home, c'est l'interface d'accueil avec un message de bienvenue.
- Cocktails, c'est l'interface qui présente l'ensemble des cocktails existant. Une petite barre de recherche est disponible pour trouver en direct un cocktail via son nom.
- Bars, c'est l'interface qui présente l'ensemble des bars existant. Une petite barre de recherche est disponible pour trouver en direct un bar via son nom.
- Submit your recipes, c'est l'interface qui permet de créer son propre cocktail. On peut sélectionner nos composants via 3 onglets dans le formulaire : Alcohol(s), Soft(s) et Ingredient(s).
- Contact, c'est l'interface pour les contacts avec un message donnant les coordonnées de l'administrateur.
- Login/Register, c'est l'interface qui permet de se connecter / créer un compte.

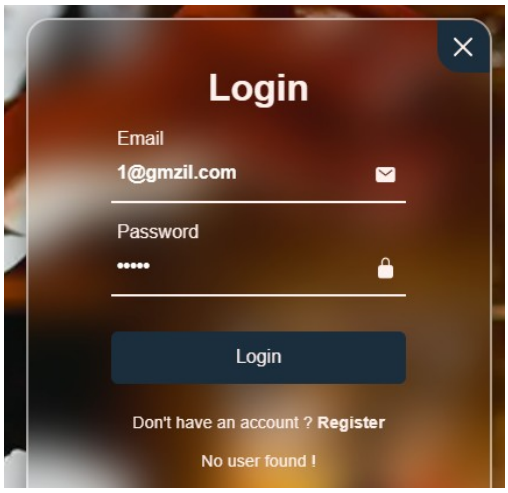
7/ Stratégie de test

Pour tester l'API, j'ai utilisé Postman pour faire tous les tests unitaires de chaque requêtes de l'API. Je n'ai cependant pas fait de test d'intégration et de performance. Il faut noter que quand j'étais en local, les appels à la base de données étaient rapides mais maintenant que le site est passé en prod, il faut souvent attendre jusqu'à 15 secondes (si ce n'est plus) pour voir les informations affichées à l'écran.

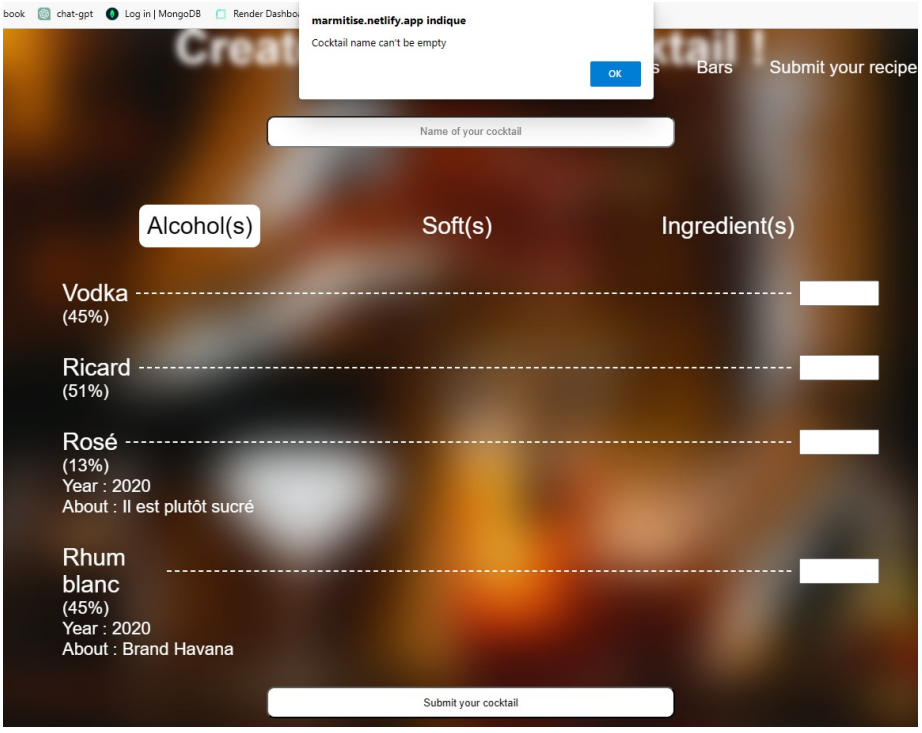
8/ Gestion des erreurs

Si il y a des erreurs, l'utilisateur sera forcément mis au courant.

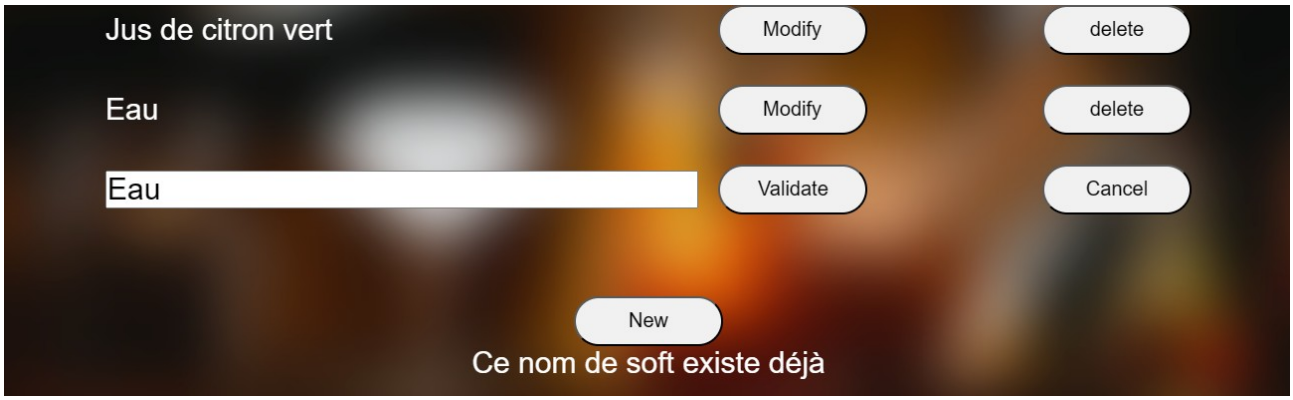
Par exemple, lorsqu'il veut se connecter mais qu'il rentre un mauvais login, alors un message apparaît en bas de la box. Idem lorsqu'il appuie sur le bouton register lorsqu'il veut se créer un compte mais se trompe.



De plus, lorsque l'utilisateur veut créer un cocktail mais qu'il ne rentre pas le nombre minimum d'information, un pop-up apparaît pour lui rappeler comment on créé un cocktail.



Enfin lorsque l’administrateur veut modifier un alcool, soft, ingrédient ou bar mais qu’il y a un problème, le message correspondant à l’erreur s’affichera en bas de la box.



9/ Sécurité

a) Description de la politique de gestion des utilisateurs et des accès

- Accès pour un utilisateur lambda :

  Il peut voir toutes les pages du site excepté la page « submit your recipe » où un texte nous indiquant de nous connecter pour soumettre sa recette apparaît.

- Accès pour un utilisateur qui s’est connecté :

  Il peut voir tout ce que voit l’utilisateur lambda mais lui, il a accès à la a page « submit your recipe » et peut créer et proposer son propre cocktail.

- Accès pour l’administrateur :

  Il peut voir tout ce qu’un utilisateur qui s’est connecté peut voir. Mais lui à la place du message d’accueil dans « Home », il a une interface lui permettant de faire l’intégralité des requêtes que propose l’API. Il ne peut cependant pas modifié l’adresse mail des utilisateurs.

b) Protection de l'API

Les routes sont sécurisées via des middlewares et j’ai rajouté dans l’API un cross-origin pour n’autoriser que les requêtes provenant du site.

c) Protection générale (OWASP top 10)

Je n’ai pas pu tester l’injection de code mais il y a des regexp qui limitent beaucoup les possibilités dans chaque input du site.

Je me suis assuré que les données sensibles soient correctement protégées lorsqu'elles sont stockées (mot de passe crypté dans le base de données), transmises (utilisation du protocole HTTPS) ou affichées.

J’ai utilisé des tokens qui sont stocké dans le localStorage pour les connexions.

10/ Intégration et déploiement

a) Intégration

J’ai fais l’intégration sur VisualStudioCode au vu de sa facilité d’utilisation et de toutes les extensions qu’il propose. A la fin de chaque étape de l’avancement, je pushais le code fait sur Github pour pouvoir travailler dessus de n’importe où quand j’avaus du temps.

b) Déploiement

J’ai déployé mon API sur Render car le déploiement y est facile, rapide et gratuit.

J’ai aussi choisi Render car il permet de faire du déploiement continu : Render offre des intégrations avec des outils populaires tels que Git, Docker, GitHub Actions, et bien d'autres. On peut facilement intégrer notre pipeline de déploiement continu (CI/CD) pour déployer automatiquement les mises à jour de notre API chaque fois qu’on effectue des modifications dans notre référentiel.

J’ai déployé mon front sur Netlify car comme pour Render le déploiement y est facile, rapide et gratuit.

11/ Conclusion

Marmitise a été un projet d’une trentaine-quarantaine d’heures où j’ai pu apprendre des nouvelles technologies et me rappeler d’autres. Honnêtement, j’en suis assez fière… Je pense que j’ai pu avec ce projet mettre la quasi-intégralité des connaissances que j’ai développé en programmation web.