UQAC
Université du Québec
à Chicoutimi

Projet de session :

6GEI311 – Architecture des logiciels

Logiciel de comparaison de prix d'hôtels

Administrateur

Mr. Oussama Jebbar

Présenté par

KAGONE, Yasmine Oulia(KAGY18530307)

ANDRIANANDRAINA, Antsanirina Jessy (ANDA23620100)

09 décembre 2024

Table des matières

Résumé

Dans le cadre de ce projet de conception, nous avons réalisé un logiciel de comparaison de prix d'hôtels. Ce rapport documente la démarche adoptée pour la conception, l'implémentation et l'évaluation de l'architecture du projet.

Le projet vise à aider les utilisateurs à comparer les prix d'hôtels en fonction de leurs critères de recherche, offrant ainsi une solution intuitive et extensible. La réalisation de ce projet suit une méthodologie structurée, comprenant la conception architecturale et le prototypage.

Évaluation de l’architecture

L’évaluation de l’architecture vise à vérifier si le système respecte les attributs de qualité définis, à savoir la modificabilité, la déployabilité et l’utilisabilité, tout en mettant en lumière les choix techniques et les compromis effectués lors de la conception. Cette évaluation va se baser sur l’analyse du code, la structure du logicielle et l’implémentation.

Modificabilité

L’architecture du système démontre une capacité à évoluer pour intégrer de nouvelles fonctionnalités ou plateformes grâce aux choix suivants:

Chaque fonctionnalité principale (comme les réservations et la gestion des utilisateurs) est isolée dans des contrôleurs et modèles dédiés (ex : UserController, BookingController).

Les classes pour les différents produits (User, Hotel, Car, etc.) sont bien séparées, respectant le principe de séparation des préoccupations.

Commenté [YK1]: c'est l'héritage on doit mettre en évidence

Le système repose sur une architecture basée sur des interfaces et des abstractions, comme l’utilisation de IUserRepository pour définir des contrats de services.

Les dépendances, telles que MongoUserRepository et BookingRepository, sont injectées via un mécanisme central dans dependancy.py. Cela facilite la substitution de modules pour implémenter de nouvelles fonctionnalités.

Les dépendances sont injectées lors de l’instanciation des controllers comme suit :

```
user_controller = UserController(user_repository)

booking_controller = BookingController(booking_repository)
```

Pour ajouter un nouveau produit (par exemple des prix de billets d’avion), nous pourrions rajouter la classe, ses différents repository et ses controllers affiliés.

Étant donné qu’on n'a pas pu trouver des API avec de plus grandes capacités de requêtes, nous nous sommes limités aux fichiers JSON. Ils ont été utilisés de manière à simuler une API externe. Les différents endpoints donnant accès aux ressources utilisées dans ces fichiers JSON peuvent être et sont destinées à etre remplacé par de réels APIs externes pour un choix plus diversifié et un contenu plus consistant.

Déployabilité

Le système est conçu pour être accessible via un navigateur sur un ordinateur, répondant aux exigences de déployabilité.

L'application utilise Flask pour gérer le backend et MongoDB pour stocker les données. Ces choix permettent une grande flexibilité et une compatibilité multiplateforme. L'intégration de CORS garantit l'accessibilité depuis des applications clientes situées sur d'autres domaines.

L'interface est construite sur Next.js, optimisée pour le SSR (Server-Side Rendering) pour assurer une compatibilité responsive.

Utilisabilité

L'interface utilisateur vise à offrir une expérience fluide et intuitive, avec des fonctionnalités bien intégrées pour la recherche, la navigation et les réservations.

L'API fournit des points d'entrée pour les recherches (routes /hotels et /cities). Cependant, la recherche est limitée à des données préchargées via hotels.json. Bien que fonctionnelle, cela restreint les capacités dynamiques de notre application.

D'un autre côté, les composants comme Billing et HomeView encapsulent des fonctionnalités spécifiques tout en permettant une extension facile.

Initialement, l'équipe a exploré l'utilisation d'API externes pour les données d'hôtels. Cependant, en raison de la limitation des clés d'API et des requêtes, cela n'a malheureusement pas pu être implémenté. Cette décision de basculer sur les fichiers JSON, bien qu'adaptée à court terme, pourrait devenir une limitation pour l'évolution future.

Documentation de l'architecture

La partie suivante est une documentation complète de l'architecture du logiciel de comparaison de prix entre différentes plateformes de réservation d'hôtels. L'application repose sur une séparation claire des responsabilités et utilise une architecture en couches.

Le comparateur d'hôtels est un système qui permet aux utilisateurs de rechercher des hôtels, comparer leurs prix et réserver des séjours. L'application est composée de deux principaux sous-systèmes : un frontend basé sur Next.js et un backend basé sur Flask. Les données sont stockées dans une base de données MongoDB.

Structure de Modules

La structure de modules décrit la décomposition logique de l'application en différents modules fonctionnels. Elle repose sur une architecture en couches pour assurer la modularité et la maintenabilité.

Diagramme de classe

Le diagramme de classes ci-dessous illustre les modules principaux et leurs responsabilités respectives au sein de l'architecture backend. Il montre comment les entités sont organisées en modèles, contrôleurs et repositories.

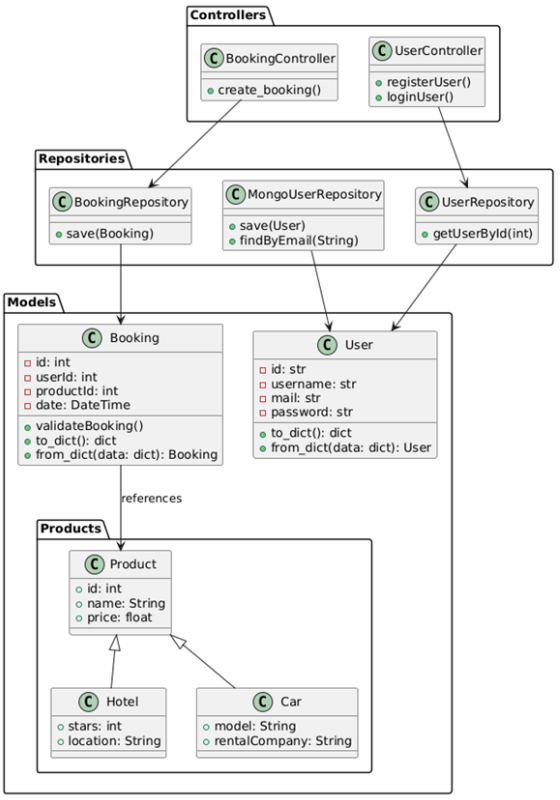

Figure 1: Diagramme de classe

Cette organisation permet de séparer les responsabilités entre les contrôleurs, les modèles, et les repositories, ce qui rend le système modulaire, maintenable et extensible.

Les controllers, comme BookingController et UserController, sont le point d'entrée du backend. Ils reçoivent les requêtes HTTP envoyées par le frontend (par exemple, développé avec Next.js) et orchestrent les actions nécessaires. Ces contrôleurs ne contiennent pas la logique métier directement mais s'appuient sur des repositories pour gérer les interactions avec la base de données et sur des services pour exécuter des opérations plus complexes.

Les repositories, comme BookingRepository et MongoUserRepository, agissent comme une interface entre la logique métier et la base de données (MongoDB, dans ce cas). Ils encapsulent les détails de persistance, permettant ainsi de changer facilement de base de données (par exemple, passer de MongoDB à SQL) sans affecter la logique métier. Ce découplage est rendu possible grâce au principe d'injection de dépendances, qui garantit que les classes collaborent via des interfaces bien définies au lieu de dépendre directement de leurs implémentations.

Les models, tels que Booking, User, et Product, représentent les entités principales du système. Ils définissent les attributs et les méthodes associées à chaque entité, comme la conversion des objets en dictionnaires (to_dict). La classe Product est une classe parent, avec des sous-classes comme Hotel et Car, ce qui illustre l'héritage et la modularité de l'architecture. Cela facilite l'ajout futur de nouveaux types de produits, comme des croisières, sans modification majeure du code existant.

Cette architecture tire parti des atouts de MongoDB, une base de données NoSQL flexible. Grâce à un fichier .env géré par la bibliothèque dotenv, les informations sensibles, comme l'URI de connexion à la base, sont externalisées pour renforcer la sécurité et faciliter la gestion des environnements de développement et de production.

Enfin, cette structure est optimisée pour les appels réseau, avec une gestion claire des requêtes et réponses HTTP entre le frontend et le backend. Elle prend également en compte les défis courants, comme les erreurs CORS, qui surviennent lorsque des requêtes sont effectuées entre des domaines ou ports différents. Globalement, cette architecture est conçue pour être robuste, évolutive et facile à maintenir, tout en restant flexible pour s'adapter aux besoins futurs.

Structure de Composantes et Connecteurs

Cette structure met en évidence les interactions entre les différents composants logiciels (frontend, backend, et base de données) et leurs connecteurs (API, repositories, etc.). Elle est documentée à travers une vue dynamique, qui illustre les interactions à travers des scénarios clés.

Nous avons mis en place quelques scénarios clés pour illustrer le comportement du système et les interactions entre les composants.

- **Scénario 1 : Recherche d'un hôtel**

Lorsque l'utilisateur saisit des critères de recherche via l'interface utilisateur (UI), la requête est transmise à l'API Gateway. Cette passerelle relaie la requête au composant HotelService du backend, qui interroge des APIs externes pour récupérer les données des hôtels. Une fois les résultats traités, ils sont renvoyés au frontend via l'API Gateway, où ils sont affichés à l'utilisateur.

- **Scénario 2 : Création d'une réservation**

Lorsqu'un utilisateur sélectionne un hôtel et soumet une demande de réservation, la requête est transmise au composant BookingController via l'API Gateway. Ce contrôleur valide les données à l'aide du modèle Booking et les enregistre dans la base de données via le BookingRepository. Une confirmation est ensuite renvoyée à l'utilisateur.

- **Scénario 3 : Connexion ou inscription d'un utilisateur**

Lorsqu'un utilisateur tente de se connecter ou de s'inscrire, la requête est envoyée au UserController. Pour une inscription, un nouvel utilisateur est créé et enregistré dans MongoDB via le MongoUserRepository. Pour une connexion, les informations d'identification sont vérifiées, et une réponse (succès ou échec) est renvoyée.

Diagramme des composants

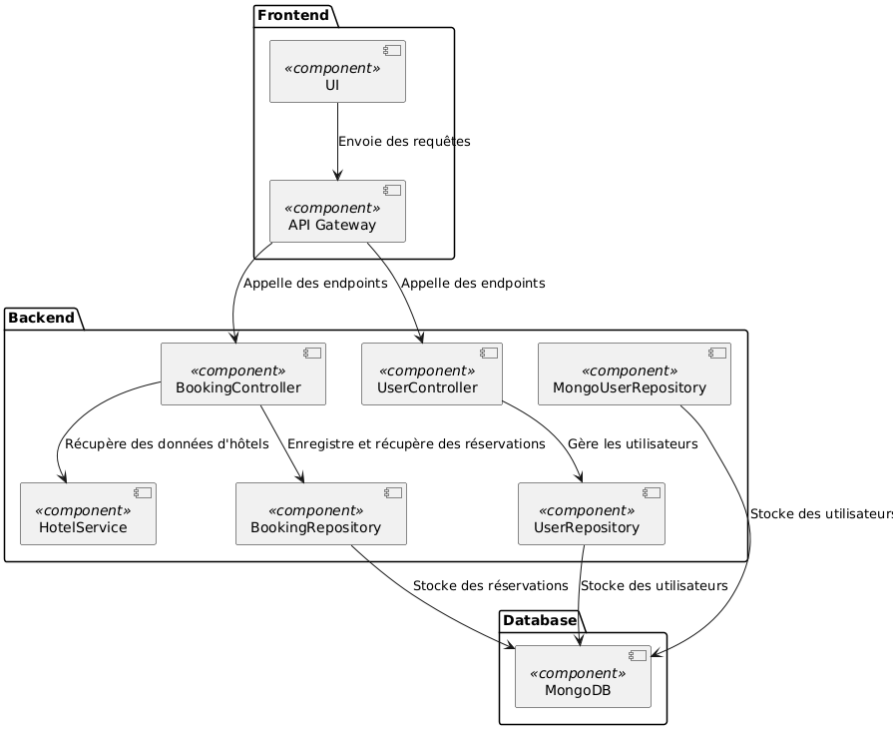

Figure 2: Diagramme des composantes

Ce diagramme de composants met en évidence l'interaction entre le frontend et le backend, ainsi que le rôle crucial de l'API Gateway en tant qu'intermédiaire entre ces deux couches. L'API centralise les appels aux endpoints backend, ce qui simplifie la gestion des requêtes et renforce la sécurité et la modularité du système. Le composant HotelService se distingue par sa capacité à récupérer des données d'hôtels via des APIs externes, enrichissant ainsi les fonctionnalités liées à la réservation. Par ailleurs, MongoDB agit comme une base de données NoSQL flexible, permettant de stocker des données, comme les utilisateurs et les réservations, tout en s'intégrant de manière fluide avec les repositories pour garantir un accès centralisé et cohérent aux données.

Le système bénéficie également d'une communication bien orchestrée entre les contrôleurs et les repositories, permettant une gestion fluide des entités principales. Par exemple, UserController et BookingController s'appuient respectivement sur MongoUserRepository et

BookingRepository pour garantir une abstraction complète des opérations de persistance, renforçant ainsi le découplage entre les couches logiques et la base de données. Enfin, cette architecture est pensée pour être évolutive et adaptable, grâce à une structure modulaire qui facilite l'ajout de nouvelles fonctionnalités, comme d'autres services ou types de données, sans perturber les composants existants.

Références

- Devdojo. (2018, 9 décembre). HTML, CSS, and Javascript in 30 minutes. Saisi le 9 décembre 2024 de https://www.youtube.com/watch?v=_GTMOmRrqkU
- Next.js. (s.d.). Documentation officielle de Next.js. Saisi le 9 décembre 2024 de https://nextjs.org/docs?utm_source=create-next-app&utm_medium=appdir-template-tw&utm_campaign=create-next-app
- Node.js. (s.d.). Documentation officielle de Node.js. Saisi le 9 décembre 2024 de https://nodejs.org/fr
- MongoDB. (s.d.). Documentation officielle de MongoDB. Saisi le 9 décembre 2024 de https://www.mongodb.com/docs/
- Miguel Grinberg. (2018, 12 juin). Flask Mega-Tutorial. Saisi le 9 décembre 2024 de https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world
- Dotenv. (s.d.). dotenv: Load environment variables. Saisi le 9 décembre 2024 de https://github.com/motdotla/dotenv
- MongoDB Atlas. (s.d.). Guide d'utilisation de MongoDB Atlas. Saisi le 9 décembre 2024 de https://www.mongodb.com/atlas
- Flask-CORS. (s.d.). Flask-CORS Documentation. Saisi le 9 décembre 2024 de https://flask-cors.readthedocs.io/
- Tailwind CSS. (s.d.). Documentation officielle de Tailwind CSS. Saisi le 9 décembre 2024 de https://tailwindcss.com/docs
- Academind. (2020, 8 juillet). What is Next.js and why it's awesome. Saisi le 9 décembre 2024 de https://www.youtube.com/watch?v=11a4oAMKP7c