ETNA : TIC-CLO5

Etape 2 : API & Architecture

Contexte

Le client dont nous essayons de répondre aux besoins possède deux hôtels. Bien que différents par le nombre et le type de chambres disponibles, ces deux hôtels mettent en place un système de prix et de réservation similaire. Disposant d'un site internet pour chacun de ses hôtels, le client voudrait néanmoins mettre en place un système de réservation unique pour tous les hôtels, et tous les canaux de ventes.
C'est ainsi que ce client nous a contacté afin de mettre en place une API de réservation hôtelière.

Besoins & contraintes

L'API demandée se doit de mettre à disposition plusieurs fonctionnalités :

1. Création d'un catalogue d'hôtels et de chambres au caractéristiques différentes
2. Permettre l'enregistrement et la gestion d'une réservation
3. Permettre la gestion d'utilisateurs et de rôle au sein de l'API

Catalogue :

Le catalogue doit donc donner la possibilité d'ajouter, visualiser, modifier, et supprimer des hôtels, ainsi que des chambres, associées à un hôtel.
Un hôtel ***doit*** contenir des caractéristiques précises (nombre de chambres, adresse, numéro de téléphone,…).
Une chambre est associée à un hôtel, et doit elle aussi contenir des caractéristiques (un nom, comme Chambre de luxe, Suite royal…, une capacité maximum, un prix,…).
Il est aussi important de notifier que la demande inclut explicitement la possibilité de modifier le catalogue, notamment par l'ajout d'un nouvel hôtel, contenant de nouvelles chambres, mais aussi par l'ajout de nouveau type de chambre, aux caractéristiques différentes de celles existantes.

Réservation :

L'API doit mettre en place un système de réservation, et donc, la réservation d'une chambre mais aussi la visualisation, la modification et la suppression de cette réservation.
Une réservation aura donc des caractéristiques importantes et variables, telles que le prix, la durée du séjour, ou encore le nombre de personnes inclues. Un autre aspect à prendre en compte est l'ajout de services spécifiques à la réservation. En effet, la demande mentionne des services ajoutés lors de la réservation, avec une politique de prix bien spécifique. Ces services sont par exemple l'ajout d'un petit-déjeuner, d'une place de parking…

Gestion des utilisateurs :

La gestion des utilisateurs n'est pas explicitement mentionnée, mais tout système de réservation se doit de mettre en place une politique de gestion des utilisateurs. Ainsi, il faudra restreindre les droits donnés à un utilisateur lambda de l'API (potentiellement un client d'un des hôtels qui souhaite visionner le catalogue sur un site), ainsi

qu’accorder des droits plus poussés à un manager ou un gérant de l’entreprise (par exemple la modification des prix, ou l’ajout d’une chambre dans un hôtel).

Contraintes techniques :

Certaines contraintes techniques ont étés spécifiées, quant à l’implémentation de l’API en elle même, ainsi qu’à la mise en place d’un workflow de développement clair et automatisé.

API :

- L’architecture doit être pensée en Microservices
- Des test doivent être mis en place afin d’assurer le fonctionnement de l’API
- Une documentation de l’API doit être fournie
- Des logs d’exécution doivent être renvoyés et exploitables

Infra :

- Le code source de l’API devra être containeurisé via Docker
- Les aspects CI/CD devront être orchestrés grâce à GitLab CI
- Pour l’automatisation et le déploiement, Ansible devra être utilisé, ainsi qu’une solution à choisir entre Docker Swarm et Kubernetes

Afin de mener à bien le développement, nous avons décidé de scinder les tâches en deux catégories, qui seront traitées en parallèle : la partie API, et la partie Infra.

Notre équipe est constituée de quatre personnes :

- Alexandra PIERIN
- Trithuan NGO
- Guillaume DELON
- Léo RESSAYRE

Alexandra et Léo s’occuperont de l’***implémentation de l’API***, tandis que Guillaume et Trithuan mettront en place la partie ***Infra.***

Développement de l’API

Choix techniques :

Tout d’abord, nous allons lister les technologies que nous avons choisi d’utiliser pour la mise en place de l’API.

Langage de développement :
Nous avons choisi d’utiliser le langage de développement ***Typescript*** afin de rester proche d’un langage bien maitrisé par les développeurs (à savoir Javascript), tout en tirant profit des avantages d’un langage typé, permettant de garder un code clair et structuré.

Environnement d'exécution :
Une fois le langage choisi, à savoir ***Typescript***, l'environnement d'exécution devient alors une évidence : NodeJs, environnement le plus utilisé et fonctionnel permettant le fonctionnement de Typescript.

Framework :
Afin de créer l'API, nous avons décidé d'utiliser un framework, afin d'avoir à disposition les outils nécessaires à l'implémentation d'un serveur web. Nous avons ainsi choisi ***Express***, puisque ce framework correspondait à nos attentes, sans pour autant mettre en place une structure fixe et potentiellement limitante (notamment afin de choisir l'architecture de projet, qui comme dit précédemment, doit être au format Microservice).
Base de données :
Le stockage des données se fera via une base de donnée relationnel

Pour le stockage, nous avons choisi d'utiliser une base de données MySql. En effet, l'organisation des hôtels, des chambres et des catégories de chambre nous a immédiatement tourné vers la stratégie relationnelle, et nous avons finalement opté pour MySql pour sa simplicité de mise en place.
Afin d'assurer la connexion entre notre API et la base de données, nous avons choisi d'utiliser l'ORM Sequelize. Ce choix est tout simplement motivé par le fait que nombre d'entre nous ont l'habitude de s'en servir.

Tests :
Afin de mettre en place les tests fonctionnels demandés, nous avons choisi d'utiliser le logiciel Postman, pour sa praticité et pour ses nombreuses fonctionnalités de regroupement de requêtes, de testing, ainsi que d'export.

Voici finalement un schéma global de l'organisation des technologies utilisées :

IMAGE

Pour la suite, nous allons vous présenter de manière général les fonctionnalités développées, puis suivra une présentation plus exhaustive et technique de ces fonctionnalités, via le listage de chaque route disponible sur l'API.
Afin, nous verrons l'organisation structurelle et technique de l'API. Ce sera l'occasion de valider l'architecture du projet.

Fonctionnalités

Afin de rendre l'API aussi fonctionnelle qu'intuitive, nous avons choisi de diviser le projet en 4 groupements métiers différents :

1. Le groupement Catalogue, permettant la gestion d'un catalogue d'hôtels, ainsi que de chambres liées à ces hôtels
2. Le groupement Configuration, permettant la gestion global des politiques appliquées aux hôtels. Ce groupement permettra donc de configurer les catégories de chambres existantes (par exemple Suite Royal, Suite Standard,…) avec leur prix, leur capacité maximale, etc. De plus, il mettra en place la gestion de la politique de prix de l'entreprise (prix des services additionnels, politique dépendant des jours de la semaine, …).

3. Le groupement Réservation, permettant la gestion des réservations, faites par rapport au catalogue, ainsi qu'à la configuration. Ce groupement implémente aussi une fonctionnalité de vérification de disponibilité d'une chambre par rapport aux réservations existantes.
4. Le groupement Utilisateur, permettant la gestion des accès à l'API.

Groupement Catalogue :

Le groupement Catalogue comme expliqué précédemment met en place les fonctionnalités de gestion du Catalogue. Il gère 2 types de données :

- Hotel :
  - id : l'id de l'hotel
  - name : le nom de l'hotel
  - address : l'adresse de l'hotel
  - phoneNumber : le numéro de téléphone de l'hotel
- Room :
  - id: l'id de la chambre
  - hotelId: l'id de l'hotel auquel appartient la chambre
  - categoryCode: le code de la catégorie de chambre à laquelle appartient la chambre (cf. Groupement Configuration)

Groupement Configuration :

Le groupement Configuration comme expliqué précédemment met en place les fonctionnalités de générales de gestion des politiques des hôtels. D'un côté, il mettra en place la création de catégorie de chambre (nous avons vue au point précédent qu'une chambre correspond en fait à une entité qui va représenter le fait qu'une chambre d'un certain type existe d'un un hôtel).

Voici la donnée qui est alors utilisée :

- Category :
  - id : l'id de la catégorie
  - code : le code de la catégorie, utile pour représenter une catégorie
  - name : le nom de la catégorie
  - capacity : le nombre de personne maximum accepté pour la catégorie
  - basePrice : le prix de base pour une nuit dans cette catégorie de chambre

D'un autre côté, ce groupement met en place la politique générale de prix. On stockera en fait en base de donnée une liste fixe de types de prix, qui sera ensuite utile lors de la réservation, afin de calculer le prix final. Ainsi, le groupement permettra de modifier la politique de prix, sans avoir à toucher au code source. Cela s'appliquera donc pour les prix de services spéciaux (type parking, ou petit-déjeuner), ainsi qu'aux autres règlementations (pourcentage en plus ou en moins selon le jour de la semaine, ou selon le nombre de personnes de la réservation).
Bien que permettant une certaine flexibilitié quant à la politique de prix, ce groupement ne permettra pas de

mettre en place de nouvelles politiques de prix (par exemple si le propriétaire veut ajouter un prix suite au désistement d'une réservation, cela va nécessiter un développement supplémentaire).

Voici la donnée qui est alors utilisée :

- PricePolicy :
  - id : l'id de la politique de prix
  - code : le code de la politique
  - type : le type de la politique (prix fixe, ou pourcentage)
  - price : le prix (utilisé lorsque le type est un prix fixe)
  - percentage : le taux à appliquer (utilisé lorsque le type est un pourcentage, et contenant un nombre flottant pouvant être négatif)

Groupement Réservation :

Le groupement Réservation comme expliqué précédemment permet la gestion des réservations, ainsi que des disponibilités des chambres. La raison de lier les disponibilités aux réservations et non pas au catalogue est que le complexité de cette fonctionnalité réside plus dans le fait de s'assurer qu'aucun réservation n'existe dans un certain laps de temps, plutôt que de s'assurer que la chambre existe, dans le bon hôtel. Ainsi, ce regroupement permettra une plus grande efficacité.
Voici la donnée qui sera alors utilisée :

- Reservation :
  - id : l'id de la réservation
  - userFullName : nom complet de la personne à l'origine de la réservation
  - userId : l'id de l'utilisateur à l'origine de la réservation
  - numberPerson : le nombre de personnes inclues dans la réservation
  - roomId : l'id de la chambre réservée (cf. Groupement Catalogue)
  - moduledPrice : le prix de la réservation après application des taux inerrants au jour de la semaine ainsi qu'au nombre de personne de la réservation. Calculé une première fois lors de la création de la réservation, et une seconde fois lors de la validation de la réservation
  - totalPrice : le prix final de la réservation, après prise en compte des services additionnels (parking, pack romance…)
  - parking : représente la présence ou non d'un place de parking dans la réservation
  - romancePack : représente la présence ou non d'un pack romance dans la réservation (disponibilité vérifiée une première fois lors de la création de la réservation, et une seconde fois lors de la validation de la réservation)
  - kidBed : représente la présence ou non d'un lit pour bébé dans la réservation
  - breakfast : représente la présence ou non d'un petit-déjeuner dans la réservation
  - checkInDate : la date de début de séjour (disponibilité vérifiée à la création et pré-réservée jusqu'à la validation ou la suppression de la réservation)

- checkOutDate : la date de fin de séjour (disponibilité vérifiée à la création et pré-réservée jusqu'à la validation ou la suppression de la réservation)

Groupement Utilisateur :

Le groupement Utilisateur comme expliqué précédemment permet la gestion des accès à l'API. Ainsi, il existera trois types d'utilisateurs :

1. Utilisateur non authentifié
2. Utilisateur authentifié standard
3. Utilisateur authentifié administrateur

Ce groupement permettra donc l'ajout d'un nouvel utilisateur, ainsi que la visualisation, la modification et la suppression des utilisateurs présents dans la base de donnée.
Voici la donnée qui sera utilisée :

- User :
  - id : l'id de l'utilisateur
  - first_name : le prénom de l'utilisateur
  - last_name : le nom de famille de l'utilisateur
  - username : le pseudo de l'utilisateur
  - email : l'adresse email de l'utilisateur
  - password : le mot de passe de l'utilisateur
  - role : le rôle, et donc le type d'accès au sein de l'API (STANDARD ou ADMIN)

Politique d'accès :

L'API sera sécurisée par un système d'authentification et de rôle, permettant d'identifier les utilisateurs, et de restreindre leurs accès si besoin.
Voici les droits accordés, selon le type d'utilisateur :

Utilisateur	Groupement Catalogue	Groupement Configuration	Groupement Réservation	Groupement Utilisateur
Non authentifié	Accès en lecture seulement	Accès seulement aux catégories de chambre (afin de voir les prix et autres informations concernant les chambres).	Accès seulement à la visualisation des disponibilité d'une chambre.	Aucun accès
Authentifié standard	Accès en lecture seulement	Accès seulement aux catégories de chambre (afin de voir les prix et autres informations concernant les chambres).	Accès à la visualisation des disponibilités, ainsi qu'à la création, la visualisation, la mise à jour, et la suppression d'une réservation UNIQUEMENT concernant le dit utilisateur.	Accès à la création, visualisation, mise à jour et suppression UNIQUEMENT concernant le dit utilisateur.

Utilisateur	Groupement Catalogue	Groupement Configuration	Groupement Réservation	Groupement Utilisateur
Authentifié administrateur	Accès à la création, visualisation, modification et suppression.	Accès à la visualisation, modification, et suppression.	Accès à la visualisation, modification, et suppression.	Accès à la visualisation, modification, et suppression.

Une route d'authentification appelée 'login' vous permettra de vous authentifier, en renseignant un username et un mot de passe. Cette route renverra un token d'authentification, valable 24H, qu'il faudra ajouter dans le header de vos requêtes, sur le champ 'Authorization' :
'Authorization' : 'Bearer <insérer le token d'authentification>'.

Ce token contiendra les informations nécessaires à l'API pour vérifier vos droits et votre identité, et vous permettra d'accéder aux informations dont l'accès vous est authorisé (cf. ***Présentation technique*** pour plus d'informations).

L'idée derrière cette manière de fonctionné est de donner la possibilité à quiconque de visualiser les choix qui leurs sont proposés, sans pour autant créer un compte, de donner la possibilité aux utilisateurs de réserver aux même, et enfin d'avoir un contrôle plus générale en tant qu'administrateur.

Présentation technique

Voici maintenant la liste exhaustive des routes mises en place par l'API. Nous l'avons représentés sous la forme d'un tableau, qui sera divisé selon les différents microservices, et donc regroupement métier.

Microservice Catalog :

Endpoint à utiliser : "http://localhost:4001"

Types de données :

- Hotel :
  - id : INT unsigned
  - name : VARCHAR(255)
  - address : VARCHAR(255)
  - phoneNumber : VARCHAR(10)
- Room :
  - id: INT unsigned
  - hotelId: INT unsigned
  - categoryCode: VARCHAR(2)

Route	Méthode	Paramètres	Body	Description	Accès
/hotels	GET	*NONE*	*NONE*	Renvoie la liste de tout les objets de type Hotel.	TOUS, même sans token d'authentification.

Route	Méthode	Paramètres	Body	Description	Accès
/hotels/<id>	GET	Id de l'hotel.	*NONE*	Renvoie un objet de type Hotel correspondant à l'id.	TOUS, même sans token d'authentification.
/hotels	POST	*NONE*	Un objet de type Hotel, sans l'id (id associé automatiquement).	Crée l'objet Hotel renseigné, et renvoie le nouvel objet.	ADMINISTRATEUR, avec un token d'authentification.
/hotels/<id>	PUT	Id de l'hotel.	Un objet de type Hotel partiel.	Modifie l'objet Hotel correspondant à l'id, et renvoie l'objet Hotel obtenu.	ADMINISTRATEUR, avec un token d'authentification.
/hotels/<id>	DELETE	Id de l'hotel.	*NONE*	Supprime l'objet Hotel correspondant à l'id, et renvoie une réponse vide.	ADMINISTRATEUR, avec un token d'authentification.
/rooms	GET	*NONE*	*NONE*	Renvoie la liste de tout les objets de type Room.	TOUS, même sans token d'authentification.
/rooms/<id>	GET	Id de la room.	*NONE*	Renvoie un objet de type Room correspondant à l'id.	TOUS, même sans token d'authentification.
/rooms	POST	*NONE*	Un objet de type Room, sans l'id (id associé automatiquement).	Crée l'objet Room renseigné, et renvoie le nouvel objet.	ADMINISTRATEUR, avec un token d'authentification.
/rooms/<id>	PUT	Id de la room.	Un objet de type Room partiel.	Modifie l'objet Room correspondant à l'id, et renvoie l'objet Room obtenu.	ADMINISTRATEUR, avec un token d'authentification.
/rooms/<id>	DELETE	Id de la room.	*NONE*	Supprime l'objet Room correspondant à l'id, et renvoie une réponse vide.	ADMINISTRATEUR, avec un token d'authentification.

Microservice Configuration :

Endpoint à utiliser : “http://localhost:4002”

Types de données :

- Category :
  - id : INT unsigned
  - code : VARCHAR(2)
  - name : VARCHAR(255)
  - capacity : INT unsigned
  - basePrice : FLOAT unsigned
- PricePolicy :
  - id : INT unsigned
  - code : VARCHAR(2)
  - type : VARCHAR(10) : ‘FIX’ ou ‘PERCENTAGE’
  - price : FLOAT unsigned
  - percentage : FLOAT signed

Route	Méthode	Paramètres	Body	Description	Accès
/categories	GET	*NONE*	*NONE*	Renvoie la liste de tout les objets de type Category.	TOUS, même sans token d’authentification.
/categories/<id>	GET	Id de la category.	*NONE*	Renvoie un objet de type Category correspondant à l’id.	TOUS, même sans token d’authentification.
/categories/code/<code>	GET	Code de la category.	*NONE*	Renvoie un objet de type Category correspondant au code.	TOUS, même sans token d’authentification.
/categories	POST	*NONE*	Un objet de type Category, sans l’id (id associé automatiquement).	Crée l’objet Category renseigné, et renvoie le nouvel objet.	ADMINISTRATEUR, avec un token d’authentification.
/categories/<id>	PUT	Id de la category.	Un objet de type Category partiel.	Modifie l’objet Category correspondant à l’id, et renvoie l’objet Category obtenu.	ADMINISTRATEUR, avec un token d’authentification.

Route	Méthode	Paramètres	Body	Description	Accès
/categories/<id>	DELETE	Id de la category.	*NONE*	Supprime l'objet Category correspondant à l'id, et renvoie une réponse vide.	ADMINISTRATEUR, avec un token d'authentification.
/pricepolicy	GET	*NONE*	*NONE*	Renvoie la liste de tout les objets de type PricePolicy.	STANDARD, avec un token d'authentification.
/pricepolicy/<id>	GET	Id de la price policy.	*NONE*	Renvoie un objet de type PricePolicy correspondant à l'id.	STANDARD, avec un token d'authentification.
/pricepolicy/<code>	GET	Code de la price policy.	*NONE*	Renvoie un objet de type PricePolicy correspondant au code.	STANDARD, avec un token d'authentification.
/pricepolicy/<code>	PUT	Code de la price policy.	Objet json : { price: INT unsigned } Ou : { percentage: FLOAT signed }	Modifie le champ *price* ou le champ *percentage* de l'objet PricePolicy correspondant au code, et renvoie l'objet PricePolicy obtenu.	ADMINISTRATEUR, avec un token d'authentification.

Microservice Reservation :

Endpoint à utiliser : "http://localhost:4003"

Types de données :

- Reservation :
  - id : INT unsigned
  - userFullName : VARCHAR(255)
  - userId : INT unsigned
  - numberPerson : INT unsigned
  - roomId : INT unsigned
  - moduledPrice : FLOAT unsigned
  - totalPrice : FLOAT unsigned
  - parking : BOOLEAN

- romancePack : BOOLEAN
- kidBed : BOOLEAN
- breakfast : BOOLEAN
- checkInDate : DATE
- checkOutDate : DATE

Route	Méthode	Paramètres	Body	Description	Accès
/reservations	GET	*NONE*	*NONE*	Renvoie la liste de tout les objets de type Reservation.	STANDARD, avec un token d'authentification, uniquement si la réservation vous appartient (userId de la réservation). ADMINISTRATEUR, avec un token d'authentification, tout le temps.
/reservations/<id>	GET	Id de la reservation.	*NONE*	Renvoie un objet de type Reservation correspondant à l'id.	STANDARD, avec un token d'authentification, uniquement si la réservation vous appartient (userId de la réservation). ADMINISTRATEUR, avec un token d'authentification, tout le temps.
/reservations/userId	GET	Id du user de la reservation.	*NONE*	Renvoie un objet de type Reservation correspondant au userId.	STANDARD, avec un token d'authentification, uniquement si la réservation vous appartient (userId de la réservation). ADMINISTRATEUR, avec un token d'authentification, tout le temps.

Route	Méthode	Paramètres	Body	Description	Accès
/reservations	POST	*NONE*	Un objet de type Reservation, sans l'id (id associé automatiquement), ni le moduledPrice et le totalPrice (calculé automatiquement).	Crée l'objet Reservation renseigné, et renvoie le nouvel objet.	STANDARD, avec un token d'authentification, uniquement si la réservation vous appartient (userId de la réservation). ADMINISTRATEUR, avec un token d'authentification, tout le temps.
/reservations/<id>	PUT	Id de la reservation.	Un objet de type Reservation partiel.	Modifie l'objet Reservation correspondant à l'id, et renvoie l'objet Reservation obtenu.	STANDARD, avec un token d'authentification, uniquement si la réservation vous appartient (userId de la réservation). ADMINISTRATEUR, avec un token d'authentification, tout le temps.
/reservations/<id>	DELETE	Id de la reservation.	*NONE*	Supprime l'objet Reservation correspondant à l'id, et renvoie une réponse vide.	STANDARD, avec un token d'authentification, uniquement si la réservation vous appartient (userId de la réservation). ADMINISTRATEUR, avec un token d'authentification, tout le temps.
/availbalilities/<hotelId>/?periodStart=<dateIn>&periodEnd=<dateOut>	GET	hotelId : Id de l'hotel dans lequel chercher les disponibilité. periodStart : début de la période dans laquelle chercher. periodEnd : fin de la période dans laquelle chercher.	*NONE*	Renvoie la liste de tout les objets Room appartenant à l'hotel correspondant au hotelId, qui sont libre durant la totalité de la période [periodStart; periodEnd].	TOUS, même sans token d'authentification.

Microservice User :

Endpoint à utiliser : "http://localhost:4004"

Types de données :

- User :
  - id : INT unsigned
  - first_name : VARCHAR(255)
  - last_name : VARCHAR(255)
  - username : VARCHAR(255)
  - email : VARCHAR(255)
  - password : VARCHAR(255)
  - role : VARCHAR(8) : ‘STANDARD‘ ou ‘ADMIN’

Route	Méthode	Paramètres	Body	Description	Accès
/users	GET	*NONE*	*NONE*	Renvoie la liste de tout les objets de type User (sans le mot de passe).	STANDARD, avec un token d’authentification, uniquement si le user vous correspond. ADMINISTRATEUR, avec un token d’authentification, tout le temps.
/users/<id>	GET	Id du user.	*NONE*	Renvoie un objet de type User correspondant à l’id (sans le mot de passe).	STANDARD, avec un token d’authentification, uniquement si le user vous correspond. ADMINISTRATEUR, avec un token d’authentification, tout le temps.
/users/me	GET	*NONE*	*NONE*	Renvoie un objet de type User correspondant au token.	STANDARD, avec un token d’authentification. ADMINISTRATEUR, avec un token d’authentification.
/users	POST	*NONE*	Un objet de type User, sans l’id (id associé automatiquement).	Crée l’objet User renseigné, et renvoie le nouvel objet (sans le mot de passe).	TOUS, sans token d’authentification. ADMINISTRATEUR, même avec token d’authentification. STANDARD ne peuvent créer qu’un compte, avec un email unique.

Route	Méthode	Paramètres	Body	Description	Accès
/users/<id>	PUT	Id du user.	Un objet de type User partiel.	Modifie l'objet User correspondant à l'id, et renvoie l'objet User obtenu.	STANDARD, avec un token d'authentification, uniquement si le user vous correspond. ADMINISTRATEUR, avec un token d'authentification, tout le temps.
/users/<id>	DELETE	Id du user.	*NONE*	Supprime l'objet User correspondant à l'id, et renvoie une réponse vide.	STANDARD, avec un token d'authentification, uniquement si le user vous correspond. ADMINISTRATEUR, avec un token d'authentification, tout le temps.
/login	POST	*NONE*	Objet json : { username: <votre username>, password: <votre mot de passe> }	Renvoie un objet json : { token : <le token d'authentification> } Il vous servira à vous authentifier pour faire les requêtes, et sera valable 24H.	TOUS, sans token d'authentification.

Architecture & fonctionnement

Nous avons vue que l'architecture du projet est structurée en microservices. Nous allons maintenant voir comment s'organise cette architecture, et comment fonctionnent ces microservices.

Chaque microservice est implémenté dans un dossier différent, contenant tous un projet NodeJs différent. Ce projet est structuré en un server Express, organisé dans un dossier ***microservices/<nom du microservice>/src***, et une partie base de donnée, dans un dossier ***microservices/<nom du microservice>/database***.

Le server Express exposera donc un endpoint accessible de l'exterieur, et importera des *routeurs* déclaré dans un dossier ***src/routes***. C'est ici que l'on déclare les différentes url, permettant chacune une action différente sur le microservice (Ex: ajouter un hotel, supprimer une chambre,…).
A noter que ces routeurs mettent en place (comme vue pour la rpésentation technique des routes) une interface REST.

Ces routes vont, une fois la requête reçu, effectuer deux actions :

1. Authentifier la requête via un *middleware*
2. Traiter la requête, grâce aux *controllers*

Middleware :
Le middleware d'authentification va intercepter la requête, en extraire le champ **Authorization** du header, et s'assurer qu'il contient un token d'authentification valide pour la requête faite. Pour ce faire, le middleware va se

servir du microservice User, et donc effectuer une requête *http* (via le client http **Axios**). Le résultat de cette requête détermine si la requête est authorisée à l'utilisateur, et donc si elle peut transiter vers le *controller*, ou non.

Controller :
Le controller est la brique technique de la requête. Il va mettre à disposition des fonctions qui vont se charger de la logique métier de la requête, et renvoyer le résultat escompté.
Parfois, le controller va s'aider de la section ***database*** pour obtenir/écrire des données sur la base de donnée du microservice, mais parfois le controller aura besoin de donnée située dans d'autres microservices. Bien que ces cas soient rare, lorsqu'il en est question, alors les controllers utilisent des ***linkers***, qui sont en fait des fichiers mettant à disposition des fonctions permettant d'effectuer des requêtes http (via le client http **Axios**), aux autres microservices (dans un but de sécurité, le token utilisé précédemment est réutilisé lors de ces requêtes).

Database :
Le dossier ***database*** contient les fichiers nécessaires à la connexion entre le microservice, et la base de donnée qui lui est associée. On y retrouve donc un fichier ***database/config/database.config.ts*** qui se charge d'établir la connexion et l'authentification avec la base de donnée, puis des fichiers ***database/models/<nom du modèle>.ts***, qui permettent d'utiliser les données de la base de donnée dans l'application. Enfin, le dossier ***database/service*** contient finalement des fichiers déclarant les fonctions basique *CRUD* utilisées par les controllers, et qui permettent lecture et écriture sur les tables de la base de donnée.

Voici finalement deux schéma récapitulatif de notre architecture Microservice :

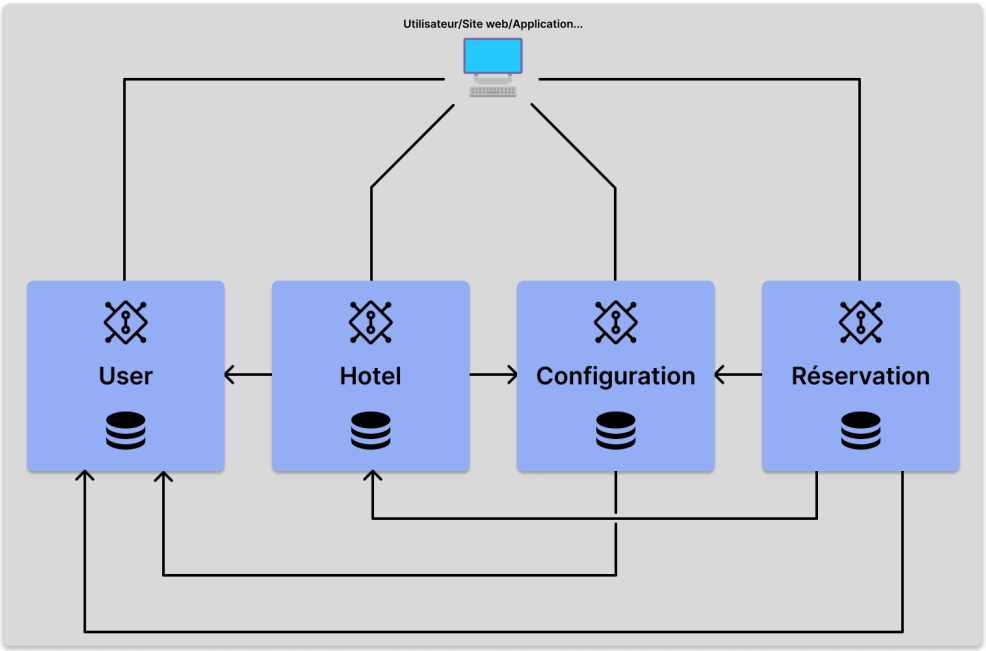

Schéma simplifié de l'architecture de l'API

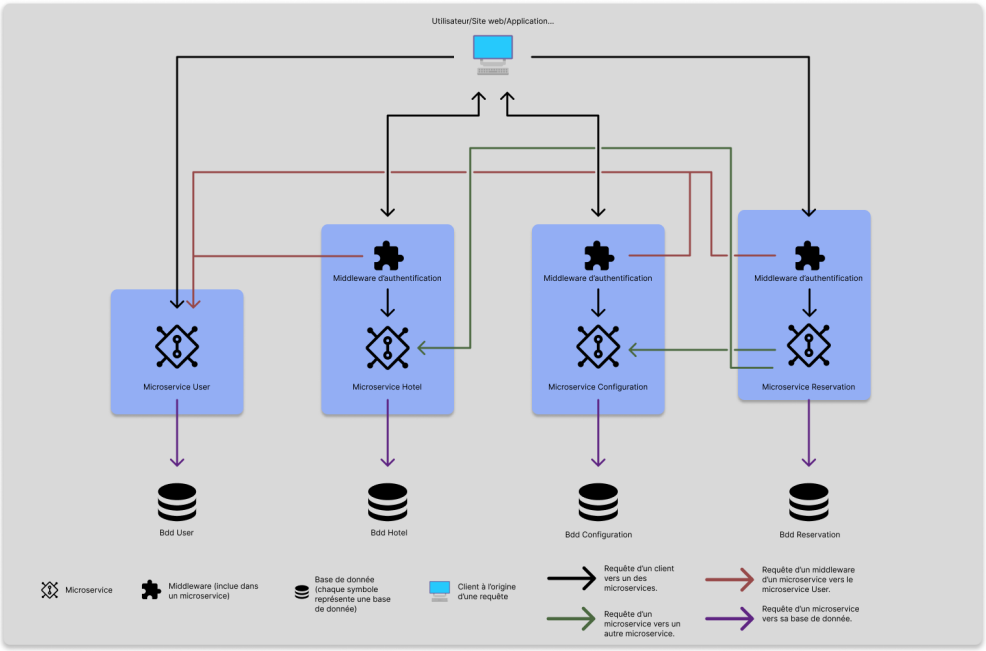

Schéma plus complet de l'architecture de l'API