HBnB - Documentation Technique UML

Introduction - Documentation Technique HBnB

À Propos de ce Document

Ce document constitue une **documentation technique complète** du projet **HBnB (Holberton Airbnb Clone)**. Il compile l'ensemble des diagrammes UML et des explications détaillées nécessaires pour comprendre et implémenter l'architecture de l'application.

Objectif du Projet HBnB

HBnB est une application de location de propriétés inspirée d'Airbnb. Elle permet aux utilisateurs de :

- **Créer et gérer des annonces** de propriétés à louer
- **Consulter les propriétés disponibles** avec leurs détails et équipements
- **Laisser des avis et des évaluations** sur les propriétés
- **Gérer leur profil utilisateur** et leurs réservations

Portée de cette Documentation

Cette documentation technique couvre :

1. Architecture Générale

- Présentation de l'architecture à trois couches
- Explication du pattern Facade
- Flux de communication entre les couches

2. Couche Métier (Business Logic)

- Diagramme de classes détaillé
- Entités principales (User, Place, Review, Amenity)
- Relations entre les entités
- Responsabilités de chaque classe

3. Flux d'Interaction API

- Diagrammes de séquence pour les appels API
- Flux de données complet
- Interactions entre les composants
- Gestion des erreurs et validations

Structure du Document

1. Introduction (ce document)
2. Tâche 0 : Diagramme de Paquets Haute Niveau
   - Architecture à trois couches
   - Pattern Facade
   - Composants principaux
3. Tâche 1 : Diagramme de Classes Détaillé
   - Entités métier (User, Place, Review, Amenity)
   - Relations et associations
   - Attributs et méthodes
4. Tâche 2 : Diagrammes de Séquence
   - Flux API pour la création d'utilisateur
   - Flux API pour la création de propriété
   - Flux API pour la création d'avis
   - Fetching a List of Places

Principes de Conception

Séparation des Responsabilités

Chaque couche a un rôle bien défini et communique uniquement avec ses voisines, garantissant une maintenance et une évolution faciles.

Pattern Facade

Le Facade agit comme une interface unifiée entre la couche Présentation et la couche Métier, simplifiant les interactions et réduisant le couplage.

Modularité

L'architecture permet d'ajouter facilement de nouvelles fonctionnalités sans affecter les composants existants.

Testabilité

Chaque couche peut être testée indépendamment, facilitant la détection et la correction des bugs.

Public Cible

Cette documentation s'adresse à :

- **Développeurs** : Pour comprendre l'architecture et implémenter les fonctionnalités
- **Architectes Système** : Pour valider les choix de conception
- **Testeurs** : Pour concevoir des tests appropriés
- **Mainteneurs** : Pour comprendre et maintenir le code

Comment Utiliser ce Document

1. **Commencez par cette introduction** pour comprendre le contexte global
2. **Consultez le diagramme de paquets** pour une vue d'ensemble de l'architecture
3. **Étudiez le diagramme de classes** pour comprendre les entités et leurs relations
4. **Analysez les diagrammes de séquence** pour comprendre les flux d'interaction
5. **Référencez les explications détaillées** pour chaque section

Conventions et Notations

Notation UML

- **Classes** : Représentées par des rectangles avec attributs et méthodes
- **Relations** : Flèches indiquant l'héritage, l'association ou la dépendance
- **Multiplicités** : Indiquant le nombre d'instances dans une relation

Nommage

- **Classes** : PascalCase (ex: UserRepository)
- **Méthodes** : camelCase (ex: getUserById)
- **Attributs** : snake_case (ex: user_id)

Notes Importantes

⚠️ **Évolution du Document** : Ce document peut être mis à jour au fur et à mesure de l'évolution du projet.

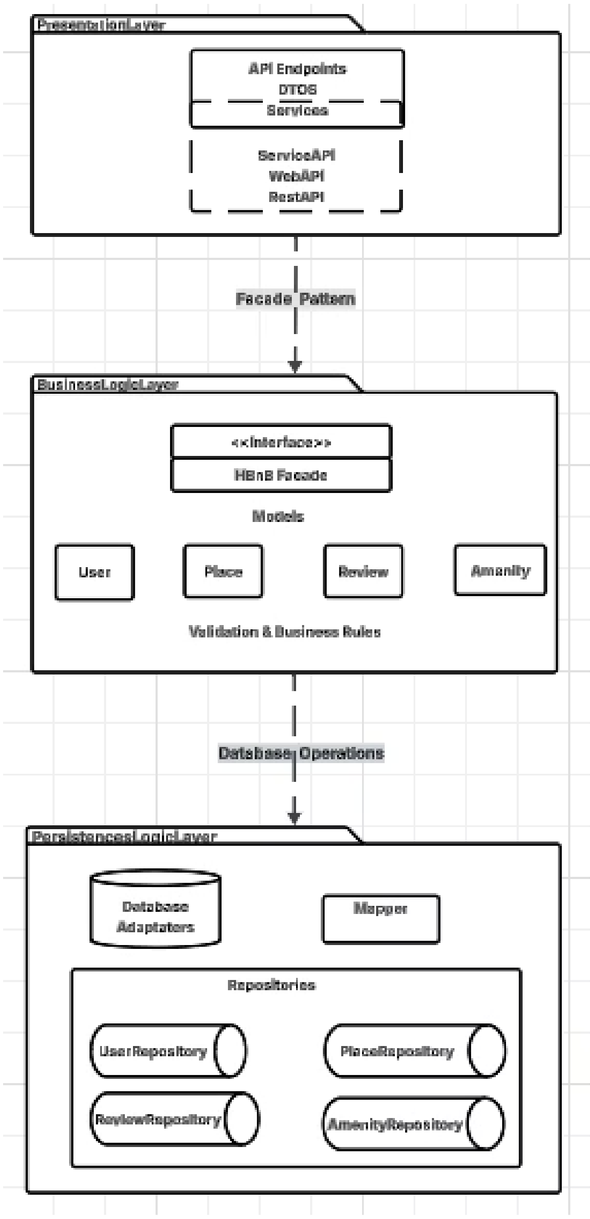
✓ **Validation** : Tous les diagrammes ont été validés pour assurer leur exactitude et leur clarté.

📚 **Ressources** : Des références et des ressources d'apprentissage sont fournies pour approfondir la compréhension de chaque concept.

Tâche 0 : Diagramme de Paquets Haute Niveau

Architecture à Trois Couches

Ce diagramme illustre l'architecture générale du projet HBnB, organisée en trois couches distinctes : Présentation, Logique Métier (Business Logic) et Persistance. Le pattern Facade est au cœur de cette architecture, agissant comme l'interface de communication unifiée entre la couche Présentation et la couche Logique Métier.



Couche Présentation (Presentation Layer)

- API Endpoints : Ce sont les points d'entrée du système, accessibles via HTTP. Ils reçoivent les requêtes des utilisateurs ou d'autres systèmes.
- DTOS (Data Transfer Objects) : Un DTO est un objet structuré servant à transporter des données entre couches sans exposer la logique interne.
- Services : Contiennent la logique pour orchestrer les appels entre les API et la couche métier. Ils traduisent les intentions de l'utilisateur en actions techniques.

Cette couche gère l'interaction avec l'extérieur : elle reçoit les demandes, les prépare, et les transmet à la logique métier

Couche Métier (Business Logic Layer)

- **«Interface» HBNB Facade :** Interface qui centralise l'accès aux fonctionnalités principales du système. Elle simplifie l'usage des composants internes.
- **Models (User, Review, Amenity) :** Représentations des entités métier. Chaque modèle contient les données et les comportements associés.
- **Validation & Business Rules :** Vérifie que les données sont valides et applique les règles métier (ex. : un utilisateur ne peut pas laisser deux avis identiques).

Cette couche contient l'intelligence du système : elle décide quoi faire et comment, selon les règles métier.

Couche Persistance (Persistence Layer)

• Repositories : Interfaces pour accéder aux données. Elles permettent de lire, écrire, modifier ou supprimer des objets

• Database Adapters : Composants qui traduisent les appels des repositories en requêtes spécifiques à la base de données. • Mappers : Convertissent les objets métier en formats compatibles avec la base de données, et inversement.

Pattern Facade

Le Facade agit comme l'interface de communication entre les couches Présentation et Logique Métier.

Flux de données

Les flèches indiquent que les données circulent du haut vers le bas :

• L'utilisateur interagit avec l'API (PresentationLayer).

• L'API appelle la logique métier (BusinessLogicLayer).

• La logique métier utilise la persistance pour accéder aux données (PersistenceLayer).

Composants Clés

Détails de Chaque Couche

Couche Présentation

- ServiceAPI: Point d'entrée principal pour les services, gère les requêtes HTTP.
- WebAPI: Interface web pour les clients, gère les sessions, l'authentification et l'autorisation.
- RestAPI: Architecture REST complète avec endpoints pour chaque ressource (Users, Places, Reviews, Amenities).

Couche Métier - Modèles Principaux

User

- Attributs: id, email, password, first_name, last_name
- Méthodes: create(), update(), delete(), validate()

Place

- Attributs: id, owner_id, name, description, price, location
- Méthodes: create(), update(), delete(), get_reviews()

Review

- Attributs: id, place_id, user_id, rating, comment
- Méthodes: create(), update(), delete()

Amenity

- Attributs: id, name, description
- Méthodes: create(), update(), delete()

Couche Persistance - Repositories

→ **UserRepository**

- Opérations CRUD pour les utilisateurs
- Recherche par email

→ **PlaceRepository**

- Opérations CRUD pour les propriétés
- Filtrage par localisation

→ **ReviewRepository**

- Opérations CRUD pour les avis
- Récupération des avis par propriété

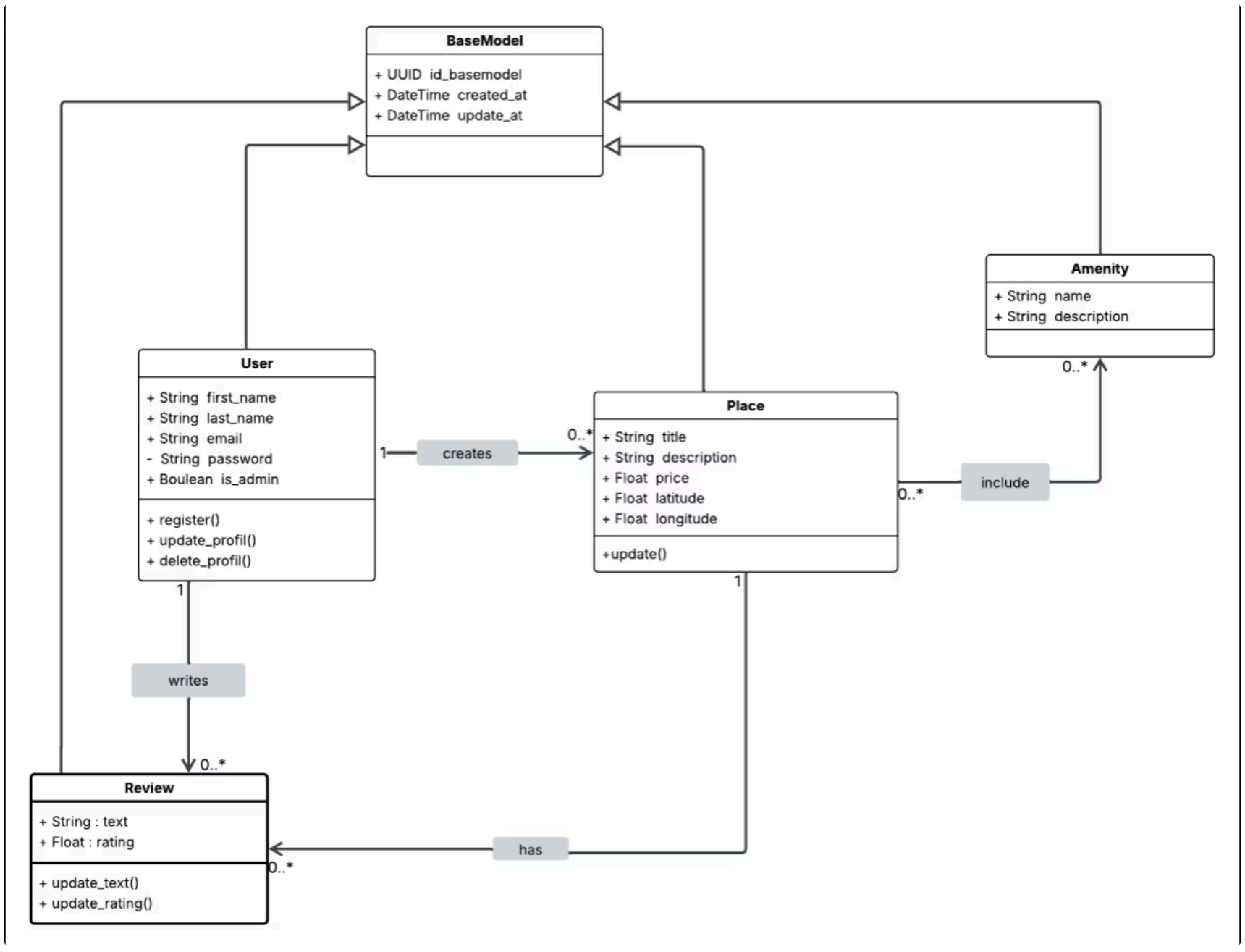
→ **AmenityRepository**

- Opérations CRUD pour les équipements
- Association avec les propriétés

Tâche 1 : Diagramme de Classes Détaillé

Couche Métier (Business Logic)

Ce diagramme présente la structure détaillée des classes au sein de la couche métier. Il met en évidence les entités principales telles que User, Place, Review et Amenity, ainsi que les relations et interactions entre elles. L'objectif est de fournir une vue claire de l'organisation des données et de la logique centrale de l'application.



Entités Principales

- **User**: Représente un utilisateur avec des attributs tels que id, email, password, first_name, last_name.
- **Place**: Décrit une propriété immobilière, incluant id, owner_id (propriétaire), name, description, price, location.
- **Review**: Enregistre un avis sur une propriété, avec id, place_id, user_id, rating et comment.
- **Amenity**: Représente un équipement ou service disponible, avec id, name, description.

Relations et Associations

- **User et Place**: Un utilisateur peut être le propriétaire de plusieurs propriétés (User 1:N Place).
- **Place et Review**: Une propriété peut recevoir plusieurs avis (Place 1:N Review).
- **User et Review**: Un utilisateur peut publier plusieurs avis (User 1:N Review).
- **Place et Amenity**: Une propriété peut être associée à plusieurs équipements (Place M:N Amenity - relation via une table intermédiaire non montrée).

Responsabilités des Classes

- **User**: Gérer l'authentification, les informations de profil et l'historique des activités de l'utilisateur.
- **Place**: Gérer les détails de la propriété, les recherches par localisation et la gestion des réservations.
- **Review**: Permettre la création, la modification et la suppression d'avis, et calculer les notes moyennes.
- **Amenity**: Gérer le catalogue des équipements et leurs associations avec les propriétés.

Tâche 2 : Diagrammes de Séquence

Flux d'Interaction API

Cette section est dédiée à l'élaboration des diagrammes de séquence, un outil essentiel pour visualiser le flux de communication et les interactions entre les différents composants de notre application. Ils nous permettront de comprendre comment les requêtes API sont traitées, depuis leur réception par la couche de présentation jusqu'à la persistance des données et le retour de la réponse au client.

Les diagrammes de séquence illustrent le déroulement chronologique des messages échangés entre les objets ou les systèmes.

Nous allons couvrir les quatres principaux flux de création via l'API :

- Création d'un utilisateur
- Création d'une propriété (Place)
- Création d'un avis (Review)
- Récupération d'une liste de propriétés

Flux API pour la Création d'Utilisateur

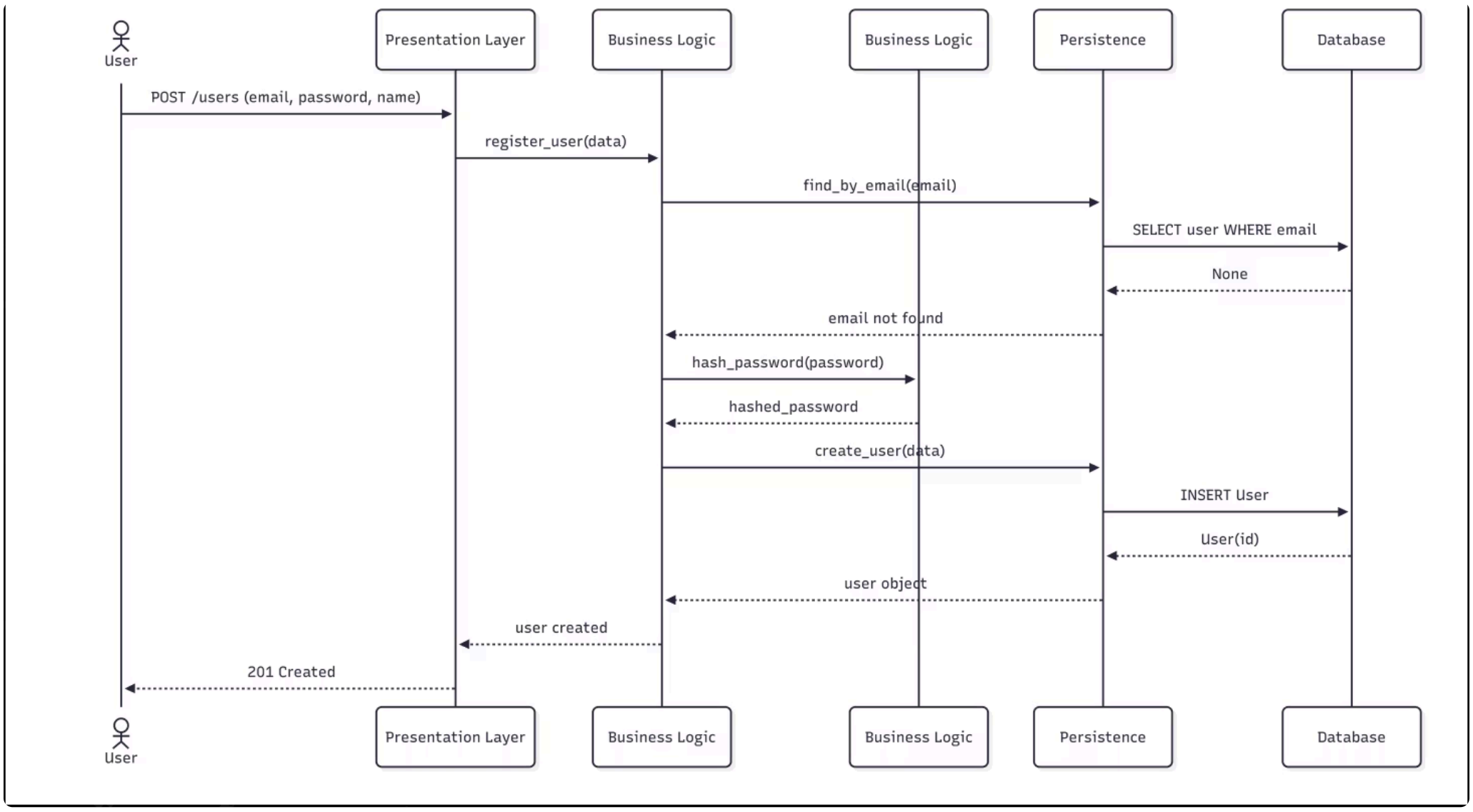
Ce flux décrit les interactions détaillées qui se produisent lorsqu'un nouveau client s'inscrit via l'interface API. Il met en lumière le cheminement d'une requête de création d'utilisateur à travers les couches de l'application, incluant la validation, la logique métier et la persistance des données.

Voici le déroulement pas à pas :

1. **Requête Client** : Le client envoie une requête HTTP POST à l'endpoint `/users` avec les données de l'utilisateur (email, mot de passe, prénom, nom) dans le corps de la requête.
2. **Réception par la Couche de Présentation (RestAPI)** : Le contrôleur `UserController` de la couche de présentation (RestAPI) intercepte la requête entrante.
3. **Validation Initiale** : La RestAPI effectue une validation syntaxique et sémantique de base des données (par exemple, format de l'email, présence des champs obligatoires).
4. **Appel à la Couche Métier (UserFacade)** : Si la validation initiale est réussie, la RestAPI délègue la requête au `UserFacade` (composant de la couche métier).
5. **Validation Métier** : Le `UserFacade` applique la logique métier spécifique à la création d'utilisateur, notamment la vérification de l'unicité de l'email dans la base de données. Il peut aussi hasher le mot de passe.
6. **Création de l'Objet Métier** : Si toutes les validations métier passent, le `UserFacade` crée une instance de l'objet métier `User`.
7. **Appel à la Couche de Persistance (UserRepository)** : Le `UserFacade` invoque la méthode `save()` du `UserRepository` pour persister le nouvel objet `User`.
8. **Interaction avec la Base de Données** : Le `UserRepository` interagit avec le système de gestion de base de données (SGBD) pour insérer les données de l'utilisateur.
9. **Retour de la Persistance** : La base de données retourne un statut de succès ou d'échec au `UserRepository`, qui le transmet au `UserFacade`.
10. **Traitement du Succès/Échec Métier** : Le `UserFacade` gère la réponse. En cas de succès, il peut générer un token d'authentification ou toute autre information pertinente. En cas d'échec (par exemple, email déjà utilisé), il génère une exception métier appropriée.
11. **Retour à la Couche de Présentation** : Le `UserFacade` retourne le résultat (objet `User` créé ou erreur) à la RestAPI.
12. **Construction de la Réponse HTTP** : La RestAPI formate la réponse HTTP. En cas de succès, elle renvoie un statut `201 Created` avec l'objet `User` (potentiellement sans le mot de passe) dans le corps de la réponse. En cas d'erreur, elle renvoie un statut approprié (`400 Bad Request`, `409 Conflict`, etc.) avec un message d'erreur clair.
13. **Envoi de la Réponse au Client** : La RestAPI envoie la réponse HTTP finalisée au client.

Diagramme de Séquence : Création d'Utilisateur

Ce diagramme illustre le flux détaillé des interactions lors de la création d'un utilisateur, de la requête initiale du client à la persistance des données dans la base de données, et le retour de la réponse.



Ce diagramme de séquence décompose le processus de création d'un utilisateur, soulignant les étapes critiques depuis l'initiation de la requête par le client jusqu'à la persistance sécurisée des données. Il met en évidence la collaboration entre la couche de présentation, la logique métier et le système de base de données. Chaque phase joue un rôle essentiel pour garantir l'intégrité et la sécurité des informations utilisateur tout en fournissant une réponse claire au client.

Validation de la Requête

- Requête POST du client à l'endpoint `/users`.
- RestAPI effectue une validation initiale des données (format, champs obligatoires).

Logique Métier

- Le `UserFacade` applique les règles métier.
- Vérification de l'unicité de l'email et hachage du mot de passe.
- Création de l'objet `User`.

Persistance des Données

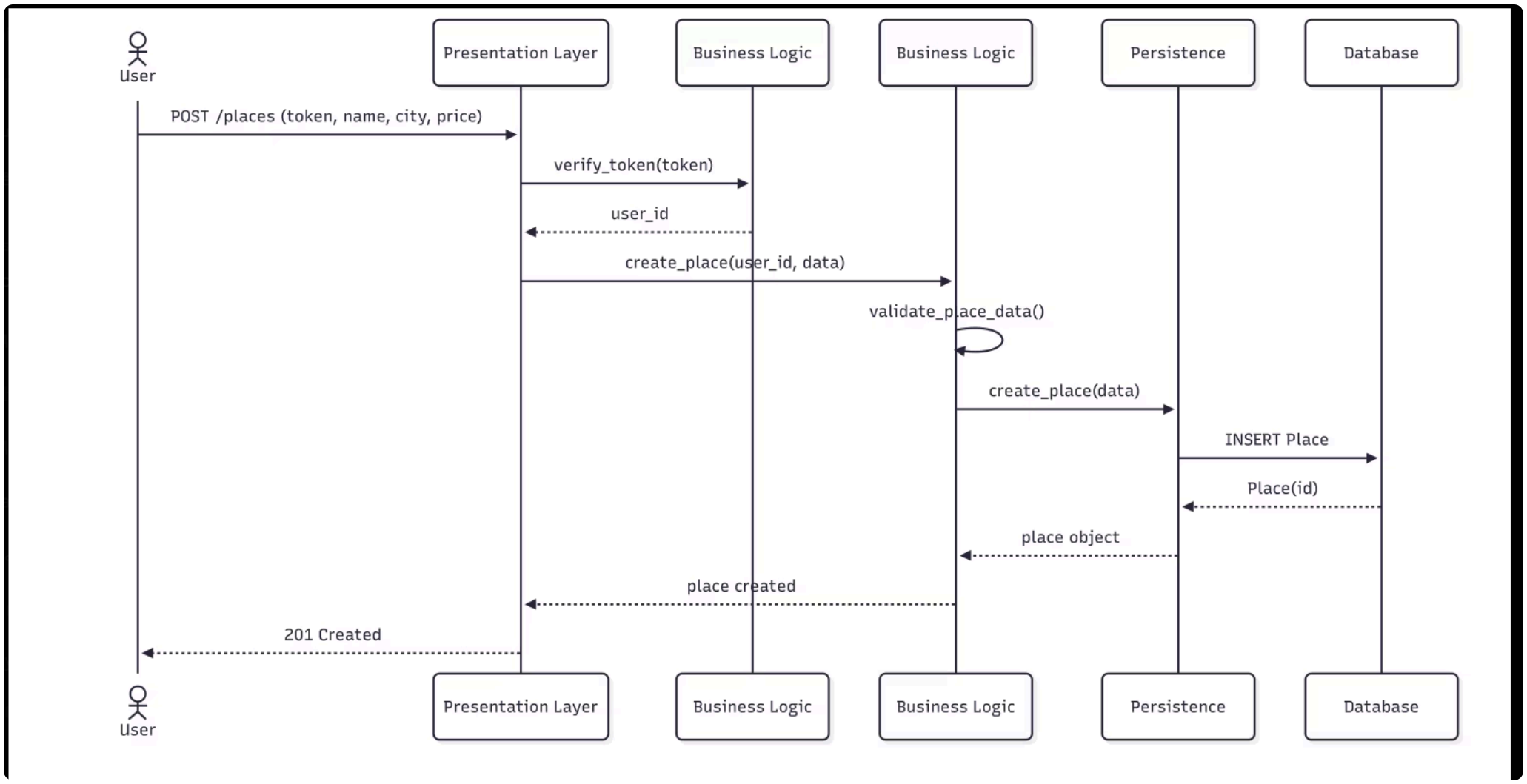
- Le `UserRepository` sauvegarde le nouvel utilisateur.
- Interaction avec la base de données pour insertion.

Réponse HTTP

- La RestAPI construit la réponse HTTP finale.
- Retourne `201 Created` (avec données utilisateur) ou une erreur `4xx`.

Diagramme de Séquence : Création de Propriété

Ce diagramme illustre le flux détaillé des interactions et des responsabilités logicielles lors de la création d'une nouvelle propriété dans le système, de la réception de la requête client à la persistance des données et à la confirmation de la création.



Ce processus de création de propriété implique plusieurs étapes critiques, allant de la réception sécurisée des données client à leur persistance et validation. Il garantit que chaque nouvelle propriété est correctement enregistrée, validée par la logique métier et associée à son propriétaire, avant de confirmer la création au client.

Validation et Vérification

Le client initie la requête, qui est validée par la RestAPI pour la syntaxe et la présence des données, suivie d'une vérification de l'existence et de l'autorisation du propriétaire.

Logique Métier

Le `Facade` orchestre la création de l'objet `Place`, appliquant les règles métier et validant les données spécifiques de la propriété (ex: prix positif).

Persistance et Associations

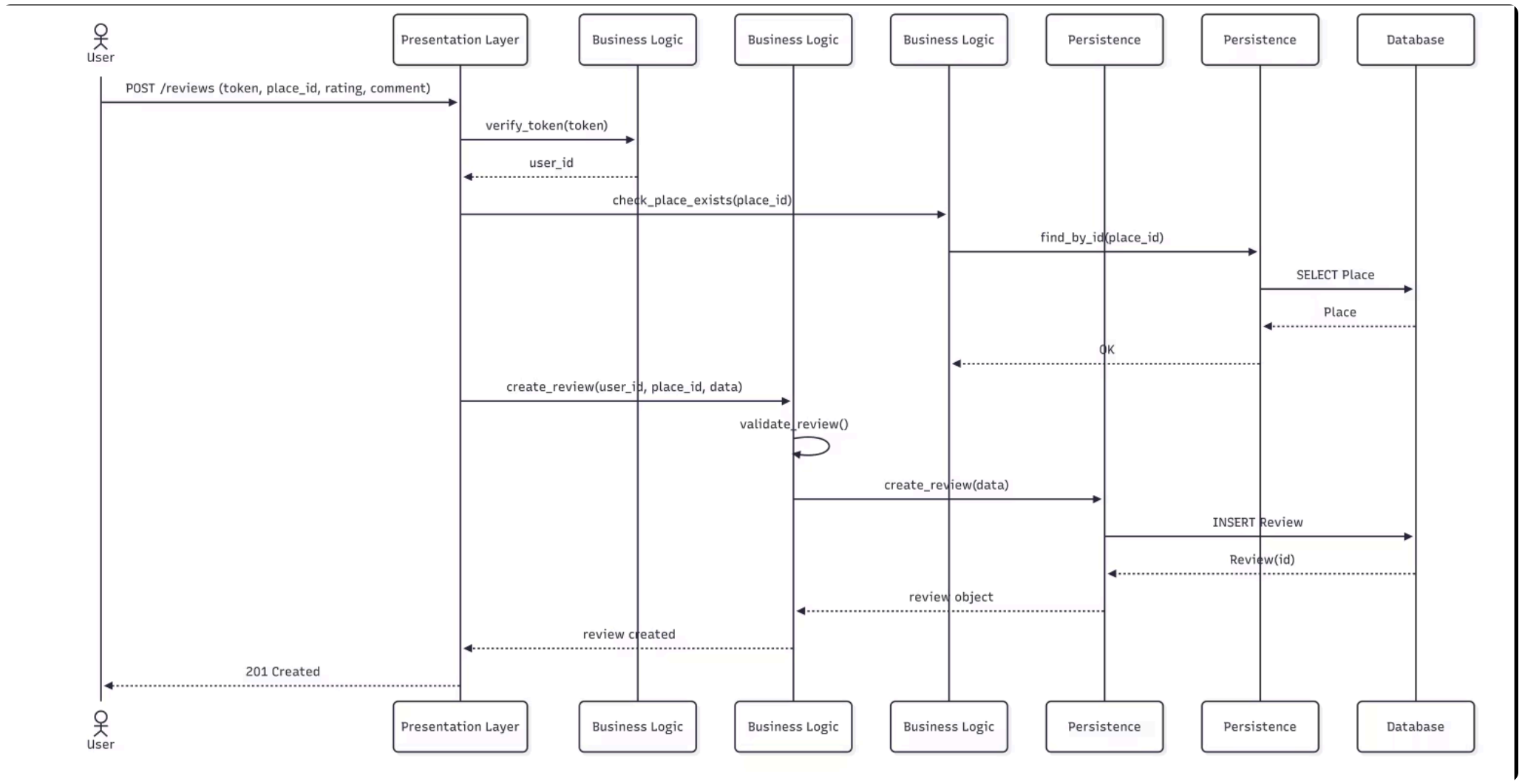
Le `PlaceRepository` sauvegarde l'objet `Place` dans la base de données, récupère son ID unique, et le `Facade` gère l'association des commodités.

Réponse au Client

La RestAPI construit et envoie une réponse HTTP `201 Created` au client, incluant l'objet `Place` entièrement créé et validé.

Diagramme de Séquence : Création d'Avis

Ce diagramme présente le flux détaillé des interactions logicielles et des validations nécessaires pour la création d'un avis client. Il illustre comment la requête du client est traitée, validée et persistée à travers les différentes couches de l'application, jusqu'à la mise à jour des données pertinentes.



Le processus de création d'avis est crucial pour maintenir l'intégrité des informations et la fiabilité des évaluations. Il commence par une validation rigoureuse de la requête et des données associées, puis passe par l'application de la logique métier pour s'assurer de la pertinence et de l'unicité de l'avis. Enfin, l'avis est sauvegardé et les agrégats comme la note moyenne sont mis à jour avant de confirmer l'opération au client.

Validation & Vérification

Le client soumet un avis qui est d'abord validé par la RestAPI. Cette étape inclut la vérification de l'existence de l'utilisateur et de la propriété associée à l'avis.

Logique Métier

Le `Facade` applique les règles métier, crée l'objet `Review` et vérifie l'absence d'avis duplicata pour le même utilisateur et la même propriété.

Persistance & Mises à Jour

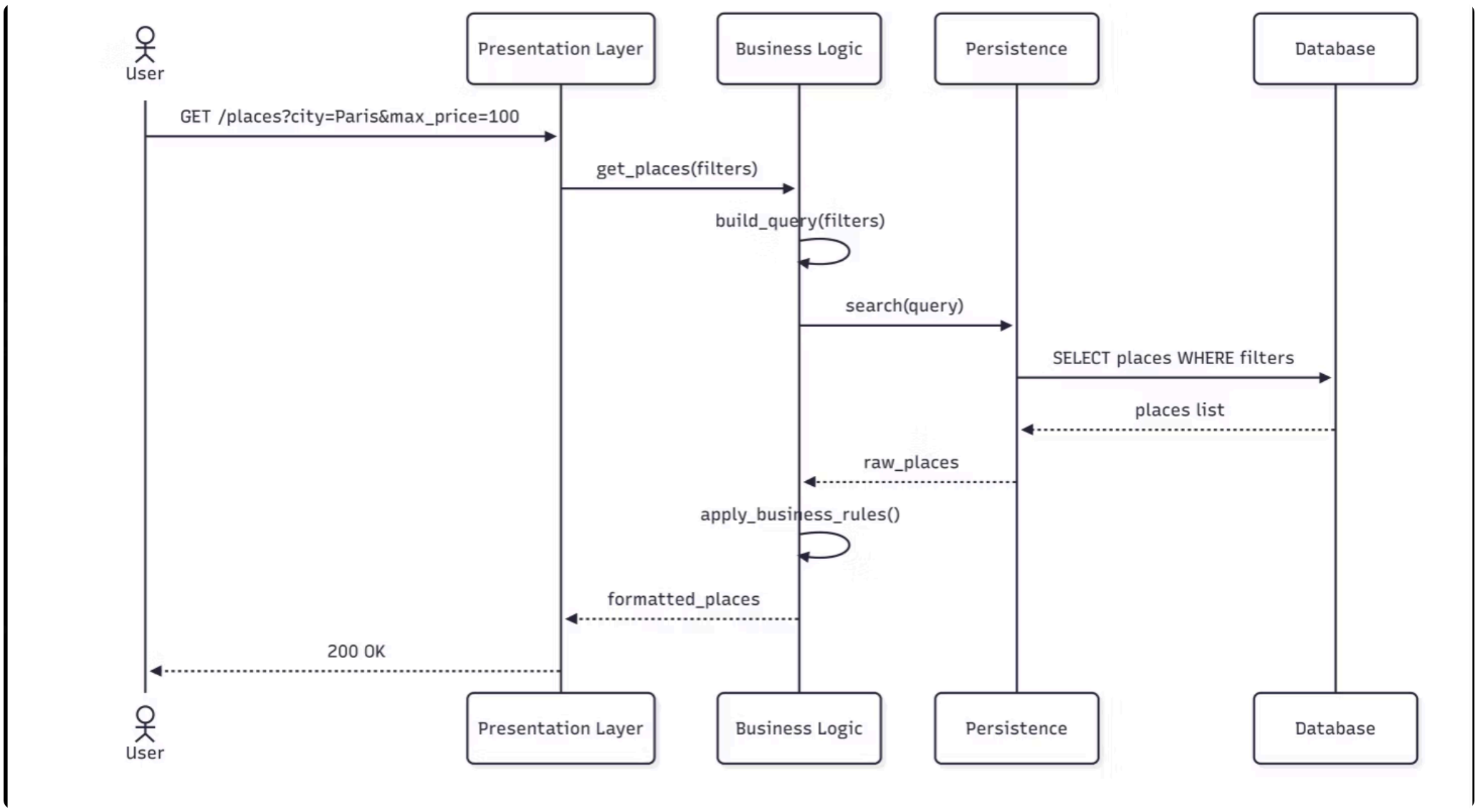
Le `ReviewRepository` sauvegarde l'avis dans la base de données. Il déclenche également la mise à jour de la note moyenne de la propriété concernée.

Réponse au Client

La RestAPI construit et envoie une réponse HTTP `201 Created` au client, confirmant la réussite de la création de l'avis et fournissant les données de l'avis créé.

Diagramme de Séquence : Récupération d'une Liste de Propriétés

Ce diagramme de séquence illustre le flux détaillé de récupération d'une liste de propriétés, y compris les interactions entre le client et les différentes couches de l'application. Il couvre la gestion des requêtes, l'application de la logique métier et la récupération des données, avec la possibilité d'utiliser des filtres et la pagination.



Le processus de récupération d'une liste de propriétés débute par une requête client qui peut inclure des critères de recherche et de pagination. La RestAPI valide ces paramètres avant de transmettre la demande au Facade, qui applique la logique métier et enrichit les données. Enfin, le Repository interroge la base de données pour renvoyer les résultats filtrés et paginés au client, assurant une expérience utilisateur rapide et pertinente.

1. Requête & Validation

Le client envoie une requête GET avec filtres (localisation, prix) et pagination.

La RestAPI valide ces paramètres pour garantir leur conformité.

2. Logique Métier

Le Facade applique les filtres métier pour affiner la recherche.

Il enrichit les données, par exemple en calculant les notes moyennes ou en ajoutant les équipements.

3. Récupération des Données

Le PlaceRepository interroge la base de données avec les critères spécifiques.

La base de données retourne la liste des propriétés correspondant aux filtres.

4. Réponse au Client

La RestAPI envoie une réponse HTTP 200 OK.

La réponse contient la liste paginée des propriétés et toutes les informations enrichies.

HBnB - Documentation Technique UML

Projet Complet

Documentation Technique Complète

Cette documentation exhaustive présente l'architecture technique, les diagrammes UML détaillés et les flux API complets du projet HBnB. Elle couvre l'ensemble des aspects fondamentaux de la conception et de l'implémentation, offrant une vue d'ensemble claire et approfondie des systèmes et de leurs interactions.

Architecture à 3 couches

Conception robuste et modulable pour une meilleure maintenabilité et évolutivité.

Diagrammes UML détaillés

Représentations visuelles claires des classes, séquences et cas d'utilisation.

Flux API complets

Descriptions précises des endpoints, requêtes et réponses pour une intégration fluide.

Patterns de conception

Application de solutions éprouvées pour résoudre des problèmes de design récurrents.

Réalisé par:

JAMES ROUSSEL
TOMMY JOUHANS

Date: - Projet HBnB