Documentation technique - Architecture et conception UML “Projet HBnB Évolution”

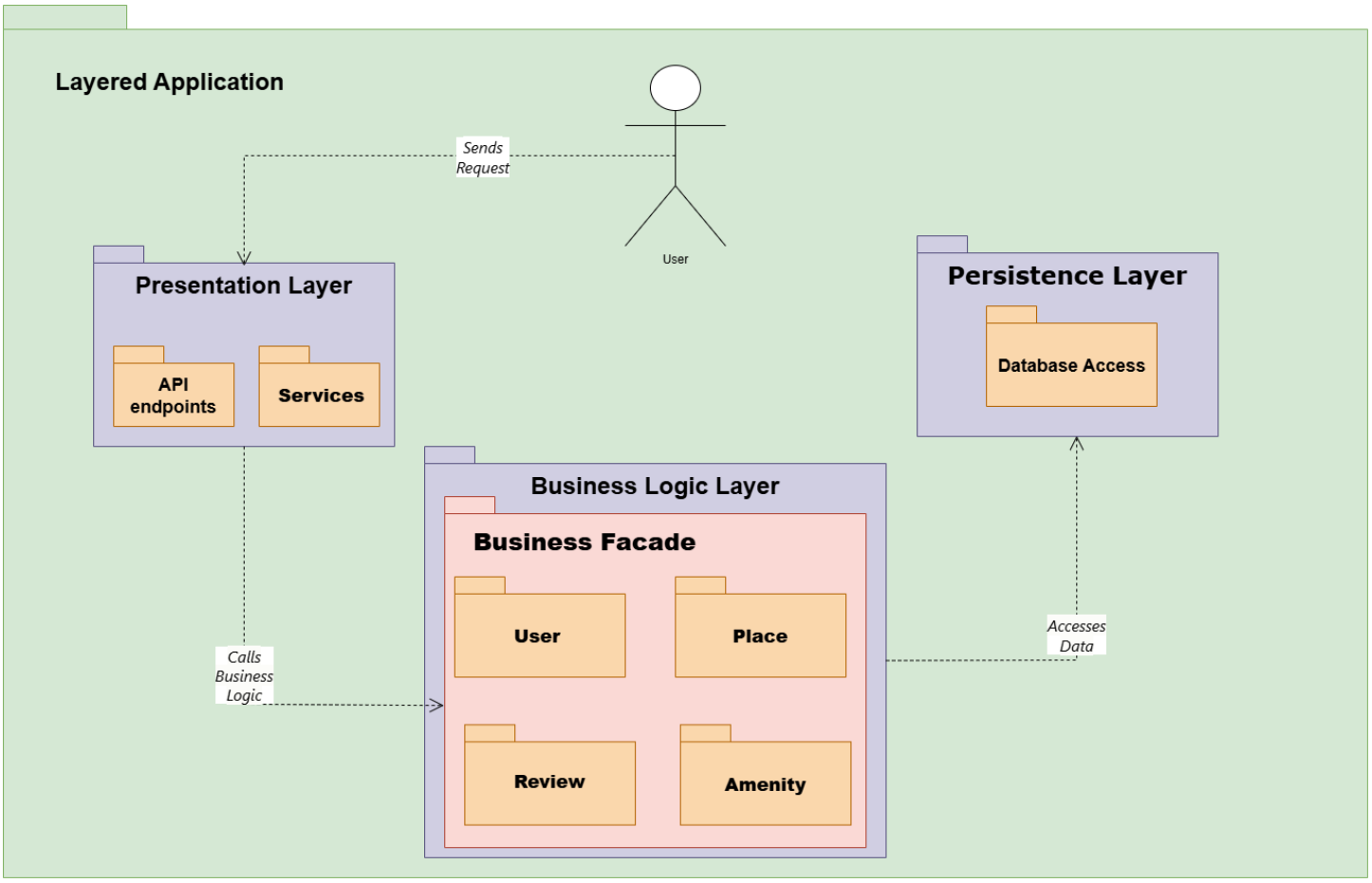
Introduction et Contexte

HBnB Évolution est une application web inspirée d’Airbnb, permettant la gestion de logements, d’avis et d’utilisateurs.

L’objectif de cette documentation est d’expliquer l’architecture logicielle à l’aide de plusieurs diagrammes UML, afin de clarifier la structure du code, les responsabilités de chaque composant, ainsi que les principaux flux de traitement.

Ce document sert de guide pour l’implémentation et sert de référence pour l’équipe.

Explication du diagramme : Hight Level Package Diagram



Ce diagramme représente l'architecture de l'application HBNB Evolution. L'objectif est de montrer comment les différentes parties de l'application communiquent entre elles et se répartissent les responsabilités.

User

- **Rôle** : L'utilisateur est la personne qui interagit avec l'application, par exemple pour réserver un logement ou laisser un avis.
- **Action** : Il envoie des requêtes à l'application (par exemple, via une interface web ou mobile).

Presentation Layer

- **Contenu** : API endpoints, Services
- **Rôle API** : C'est la partie de l'application qui reçoit les requêtes de l'utilisateur. Elle sert d'interface entre l'utilisateur et le reste du système.
- **Fonctionnement** :
    - Elle reçoit la demande de l'utilisateur (par exemple, "Recherche des logements disponibles").
    - Elle transmet cette demande à la couche suivante, la logique métier.
- **Rôle Services :** contient la logique de traitement propre à l'application, mais restent dans la couche de présentation.
- **Fonctionnement** :
    - Orchestre les différentes opérations nécessaires pour répondre à une requête : ils appellent la logique métier, coordonnent plusieurs actions, gèrent les exceptions, préparent la réponse à envoyer aux endpoints.

Business Logic Layer

- **Contenu** : Business Facade, User, Place, Review, Amenity
- **Rôle** : Cette couche contient toute la logique métier, c'est-à-dire les règles et traitements qui correspondent aux besoins réels de l'application (ex : vérifier qu'un logement est disponible, créer un utilisateur, ajouter un avis).
- **Fonctionnement** :
    - La **Business Facade** centralise l'accès aux différentes fonctionnalités métier.
    - Les classes comme User, Place, Review, Amenity représentent les entités principales du système.
    - Cette couche exécute les opérations métier demandées par la couche de présentation.

Persistence Layer

- **Contenu** : Database Access
- **Rôle** : Cette couche gère l'accès aux données stockées (base de données).
- **Fonctionnement** :
    - Elle reçoit les demandes de lecture ou d'écriture de la couche de logique métier.
    - Elle s'occupe de sauvegarder, modifier ou récupérer les données nécessaires.

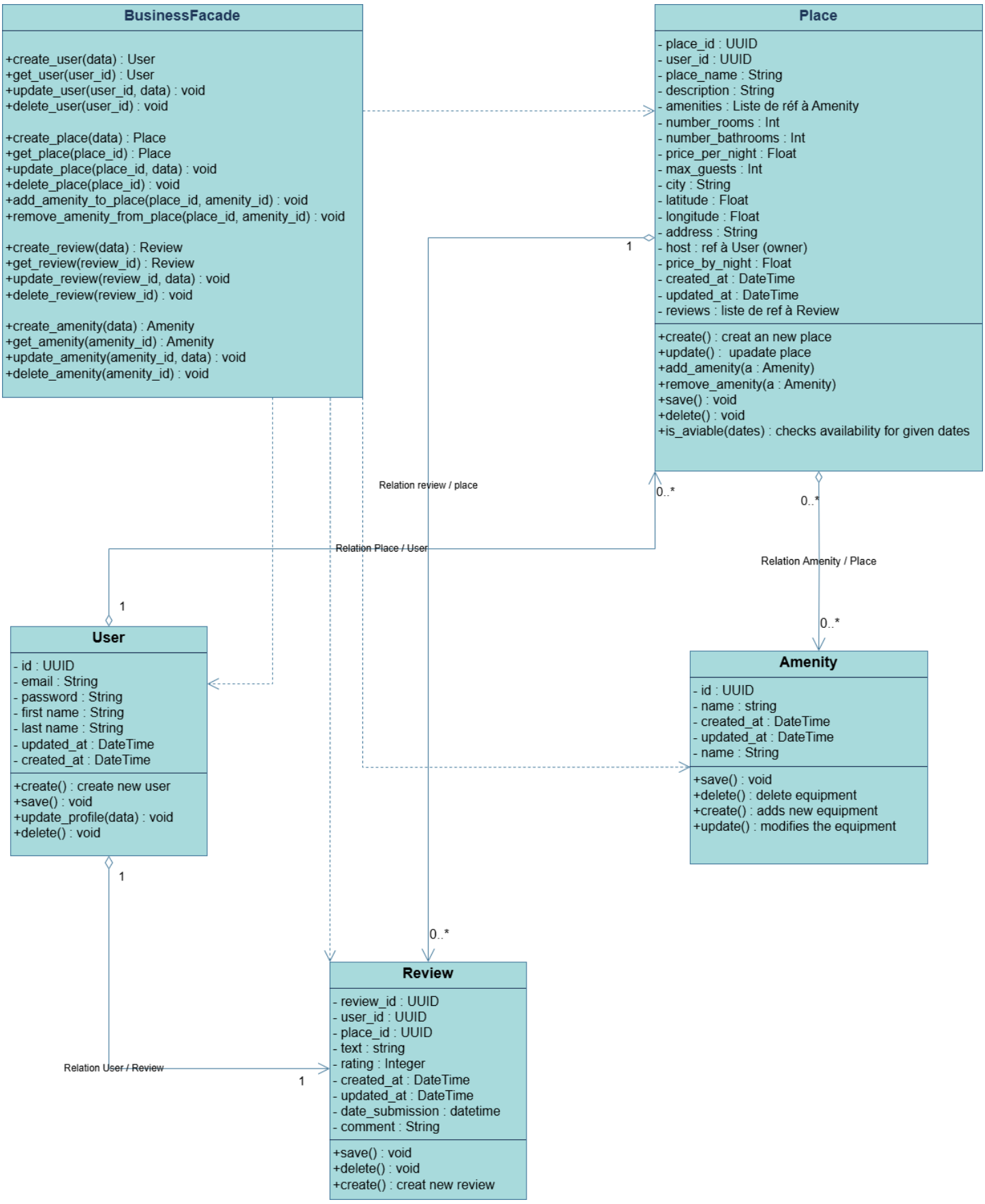
Communication entre les couches

- L'utilisateur envoie une requête à la **Presentation Layer**.
- La **Presentation Layer** appelle la **Business Logic Layer** pour traiter la demande.
- La **Business Logic Layer** utilise la **Persistence Layer** pour accéder ou modifier les données si besoin.
- Le résultat remonte ensuite jusqu'à l'utilisateur.

Points clés

- **Séparation des responsabilités** : chaque couche a un rôle bien précis, ce qui rend l'application plus claire, plus facile à maintenir et à faire évoluer.
- **Centralisation de la Business Logic Layer** : toutes les règles métier sont regroupées dans une seule couche, ce qui évite les duplications et les erreurs.
- **Sécurité et robustesse** : cette architecture permet de mieux contrôler les accès aux données et de sécuriser les traitements.
- **Extensibilité** : il est facile d'ajouter de nouvelles fonctionnalités ou de modifier une partie de l'application sans tout casser.

Explication du Diagramme : Class Diagram for Business Logic Layer



Ce diagramme de classe modélise l'architecture de l'application. Il décrit les principales entités (classes) du système, leurs attributs, méthodes et les relations entre elles.

Les principales classes et leur rôle :

User

- Représente un utilisateur du site (propriétaire ou client).
- Attributs : email, mot de passe, prénom, nom, dates de création et de mise à jour.
- Méthodes : créer, sauvegarder, mettre à jour le profil, supprimer l'utilisateur.

Place

- Représente un logement proposé à la location.
- Attributs : identifiant, nom, description, nombre de chambres, salles de bain, capacité, prix, localisation, dates, etc.
- Liens :
    - Un logement appartient à un utilisateur (propriétaire).
    - Un logement peut avoir plusieurs équipements.
    - Un logement peut recevoir plusieurs avis.
- Méthodes : créer, mettre à jour, ajouter/supprimer un équipement, vérifier disponibilité.

Amenity

- Représente un équipement ou service disponible dans un logement (ex : Wi-Fi, piscine).
- Attributs : identifiant, nom, dates.
- Méthodes : créer, sauvegarder, supprimer, mettre à jour.

Review

- Représente un avis laissé par un utilisateur sur un logement.
- Attributs : identifiant, utilisateur, logement, note, texte, dates.
- Méthodes : créer, sauvegarder, supprimer.

BusinessFacade

- Sert d'interface entre l'application et les opérations métier.

- Contient des méthodes pour gérer les utilisateurs, logements, équipements et avis (création, modification, suppression, etc.).
- Centralise la logique métier pour simplifier l'utilisation des différentes classes.

Les relations entre les classes

- **Un utilisateur peut posséder plusieurs logements (places)** : relation 1 à 0..* entre User et Place.
- **Un logement peut avoir plusieurs équipements** : relation 0..* entre Place et Amenity.
- **Un logement peut recevoir plusieurs avis** : relation 0..* à 1 entre Place et Review.
- **Un avis est lié à un utilisateur et un logement** : relation 1 à 1 entre Review, User et Place.

Fonctionnement global du système

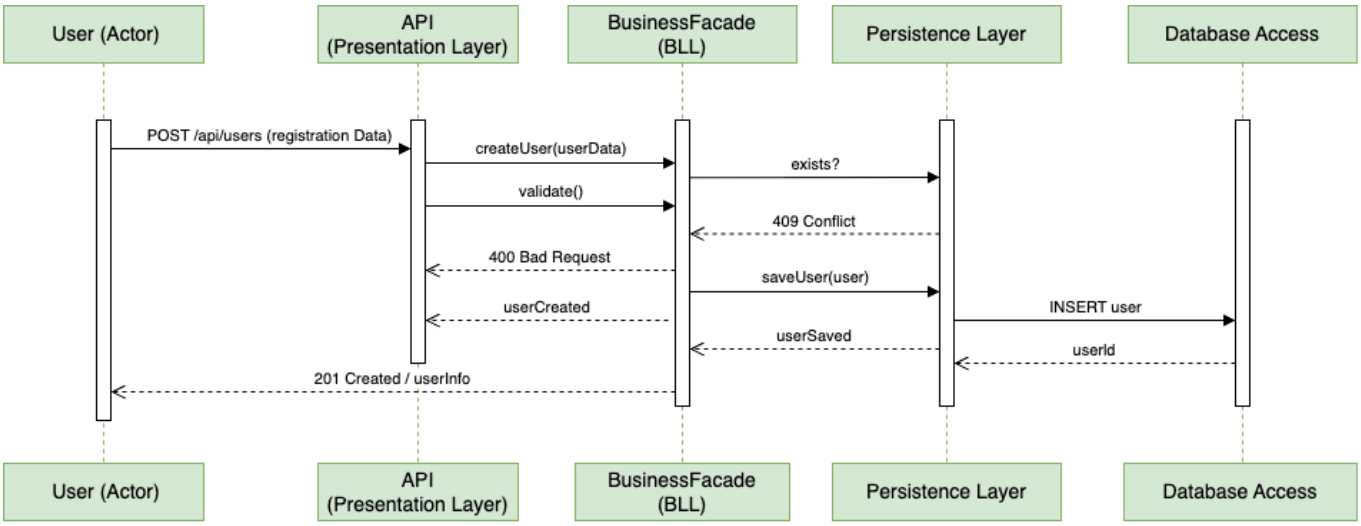
- **Création d'un utilisateur** : via BusinessFacade, qui utilise la classe User.
- **Ajout d'un logement** : un utilisateur peut créer un logement, ajouter des équipements, fixer un prix, etc.
- **Ajout d'un équipement** : via la classe Amenity, puis association à un logement.
- **Ajout d'un avis** : un utilisateur peut laisser un avis sur un logement après son séjour.
- **Gestion centralisée** : toutes les opérations passent par BusinessFacade, qui simplifie l'accès aux fonctionnalités principales et gère les interactions entre les entités.

Points clés

- **Séparation des responsabilités** : chaque classe a un rôle précis (utilisateur, logement, équipement, avis).
- **Relations entre les entités** : explique les cardinalités (un à plusieurs, plusieurs à plusieurs).
- **Centralisation de la logique métier** : BusinessFacade permet de ne pas dupliquer la logique dans chaque classe.
- **Extensibilité** : le modèle permet facilement d'ajouter de nouveaux types d'équipements ou de nouvelles fonctionnalités.

Ce diagramme illustre donc la structure d'une application de location de logements, en mettant en avant les interactions principales et la gestion centralisée des opérations.

Explication du diagramme de séquence : User Registration



Ce diagramme de séquence montre toutes les étapes et interactions entre les différentes couches de l'application lors de l'inscription d'un nouvel utilisateur. Il illustre comment une demande d'inscription (registration) est traitée du début à la fin.

1. Acteurs et couches impliquées

- **User (Actor)** : L'utilisateur qui souhaite s'inscrire.
- **API (Presentation Layer)** : La couche qui reçoit la requête HTTP (par exemple, via une API REST).
- **BusinessFacade (BLL)** : La couche de logique métier, qui gère les règles métier et la coordination des opérations.
- **Persistence Layer** : La couche qui s'occupe de l'accès aux données (base de données).
- **Database Access** : La base de données elle-même.

2. Déroulement étape par étape

1. **L'utilisateur envoie sa demande d'inscription**
    - Il fait un appel (POST vers `/api/users`) avec ses données d'inscription (registration Data).
2. **L'API reçoit la demande**
    - L'API transmet les données à la couche métier via la méthode `createUser(userData)`.
3. **Validation des données**
    - La couche métier (BusinessFacade) valide les données reçues (`validate()`).

- Si les données sont invalides, elle renvoie une erreur (`400 Bad Request`) à l'API, qui la transmet à l'utilisateur.

4. **Vérification d'existence**
   - Si les données sont valides, la couche métier demande à la couche de persistance si l'utilisateur existe déjà (`exists?`).
   - Si l'utilisateur existe déjà, la couche persistance renvoie un code (`409 Conflict`), qui est transmis à l'utilisateur.
5. **Sauvegarde de l'utilisateur**
   - Si l'utilisateur n'existe pas, la couche métier demande à la couche persistance de sauvegarder le nouvel utilisateur (`saveUser(user)`).
   - La couche persistance envoie la commande d'insertion à la base de données (`INSERT user`).
   - La base de données retourne l'identifiant du nouvel utilisateur (`userId`).
6. **Confirmation de création**
   - La couche persistance confirme à la couche métier que l'utilisateur a été sauvegardé (`userSaved`).
   - La couche métier informe l'API que l'utilisateur a été créé (`userCreated`).
   - L'API renvoie la réponse finale à l'utilisateur : (`201 Created / userInfo`) (utilisateur créé avec succès et informations retournées).

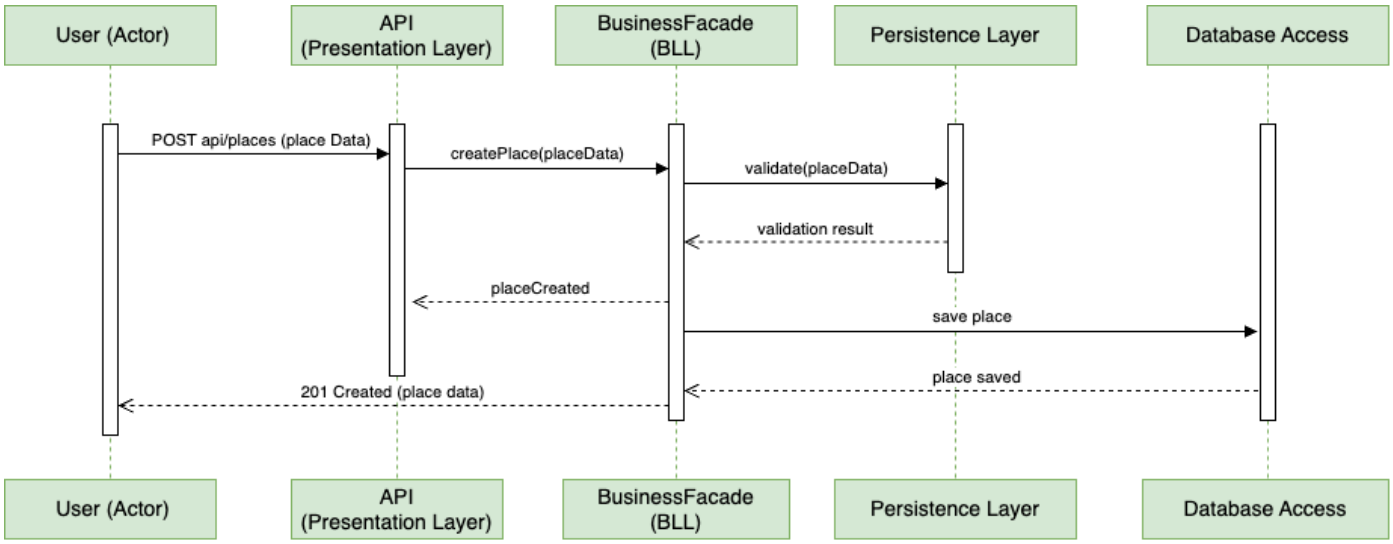
3. Points clés

- **Séparation des responsabilités** : chaque couche a un rôle précis (présentation, logique métier, persistance, base de données).
- **Gestion des erreurs** : le diagramme montre comment sont gérées les erreurs courantes (données invalides, utilisateur déjà existant).
- **Flux de données** : la demande part de l'utilisateur, traverse toutes les couches, et la réponse remonte de la même manière.
- **Respect des bonnes pratiques** : ce type d'architecture facilite la maintenance, la sécurité et l'évolution de l'application.

4. Résumé en une phrase

Ce diagramme de séquence décrit comment l'application HBNB gère l'inscription d'un utilisateur, depuis la réception de la demande jusqu'à la sauvegarde en base de données, en passant par la validation, la vérification d'unicité et la gestion des erreurs.

Explication du diagramme de séquence : Place Create



Ce diagramme de séquence illustre toutes les étapes nécessaires à la création d'un nouveau logement ("place") dans l'application, depuis la demande de l'utilisateur jusqu'à la sauvegarde dans la base de données et la réponse finale.

1. L'utilisateur initie la création d'un logement

- L'utilisateur (User Actor) envoie une requête HTTP POST à l'API, sur l'endpoint `api/places`, avec les données du logement à créer (place Data).

2. L'API reçoit la demande et la transmet à la logique métier

- La couche API (Presentation Layer) reçoit la requête et appelle la méthode `createPlace(placeData)` de la couche BusinessFacade (BLL), qui gère la logique métier.

3. Validation des données du logement

- BusinessFacade transmet les données à Persistence Layer pour validation, via la méthode `validate(placeData)`.
- Persistence Layer renvoie le résultat de la validation à la BusinessFacade (validation result).

4. Sauvegarde du logement dans la base de données

- Si les données sont valides, BusinessFacade demande à Persistence Layer de sauvegarder le logement (`save place`).
- La Persistence Layer interagit alors avec la Database Access pour enregistrer effectivement le logement.
- Une fois la sauvegarde terminée, la base de données confirme la réussite à la Persistence Layer (`place saved`), qui relaie l'information à BusinessFacade.

5. Retour de la réponse à l'utilisateur

- BusinessFacade informe l'API que le logement a bien été créé (`placeCreated`).
- L'API renvoie alors à l'utilisateur une réponse HTTP 201 Created, accompagnée des données du logement créé.

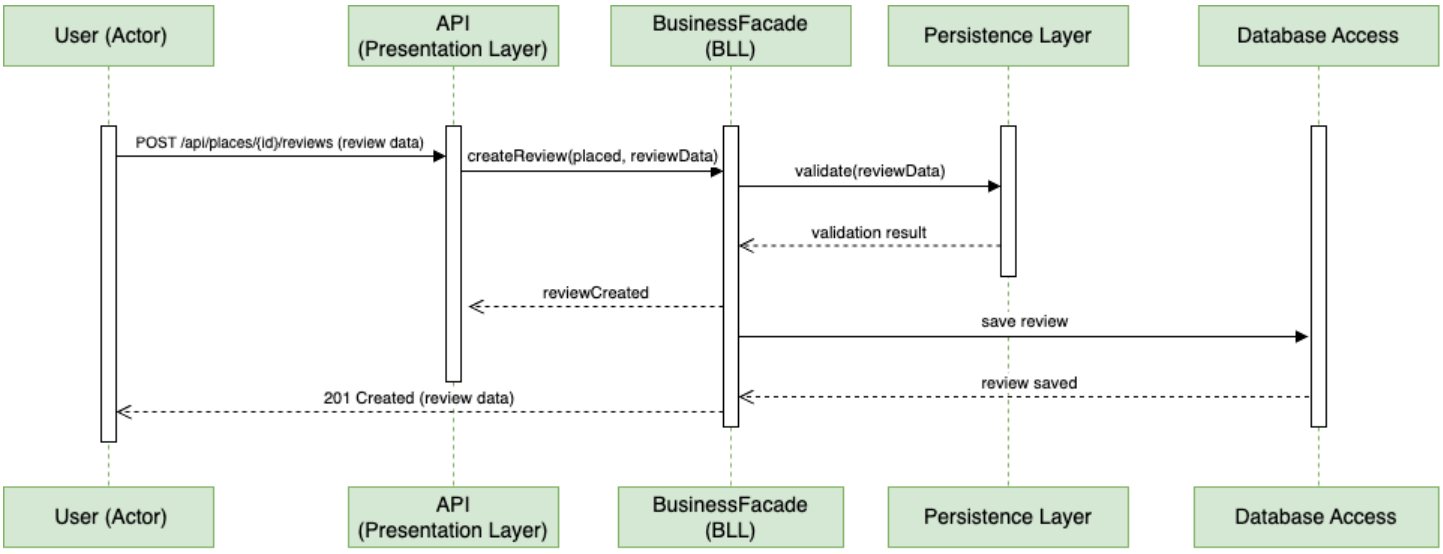
Points clés

- **Séparation des responsabilités** : chaque couche (API, logique métier, persistance, base de données) a un rôle précis, ce qui rend l'application plus claire et maintenable.
- **Validation** : avant toute sauvegarde, les données sont systématiquement validées pour éviter les erreurs.
- **Traçabilité** : chaque étape est représentée, ce qui permet de bien comprendre le cheminement complet de la demande.
- **Réponse structurée** : l'utilisateur reçoit toujours une réponse claire (succès ou échec), ici un code 201 Created en cas de réussite.

Résumé

Ce diagramme montre comment une demande de création de logement est traitée de bout en bout dans l'application HBNB : depuis l'utilisateur, en passant par l'API et la logique métier, jusqu'à la base de données, avec validation et confirmation à chaque étape.

Explication du diagramme de séquence : Review Submission



Ce diagramme illustre le **flux d'envoi d'un avis (review)** par un utilisateur sur un lieu (place) via l'application HBNB. Il montre comment une requête utilisateur traverse les différentes couches de l'architecture logicielle jusqu'à l'enregistrement de l'avis en base de données.

1. Déclenchement par l'utilisateur

- L'utilisateur (User/Actor) envoie une requête HTTP POST vers l'endpoint `/api/places/{id}/reviews` avec les données de l'avis (review data).
- Cette action correspond à la volonté de laisser un commentaire ou une note sur un lieu spécifique du site, comme sur Airbnb.

2. API (Presentation Layer)

- La couche API (Presentation Layer) reçoit la requête.
- Elle extrait et valide les données, puis appelle la couche métier (BusinessFacade/BLL) via la méthode createReview(placed, reviewData).
- Ici, la couche de présentation agit comme intermédiaire entre l'utilisateur et la logique métier, en s'assurant que la requête est bien formée avant de la transmettre.

3. Business Facade (BLL)

- La couche métier (Business Logic Layer) reçoit la demande de création d'avis.
- Elle va orchestrer la suite des opérations, notamment la validation métier des données et la gestion de la persistance, en appelant la Persistence Layer pour valider les données (`validate(reviewData)`).

4. Persistence Layer

- Cette couche est responsable de la gestion des accès aux données.
- Elle valide d'abord les données reçues, puis, si elles sont correctes, elle procède à l'enregistrement de l'avis dans la base de données (`save review`).
- Elle renvoie ensuite un accusé de réception à la couche métier pour indiquer que l'avis a bien été sauvegardé.

5. Database Access

- C'est ici que l'avis est effectivement enregistré dans la base de données.
- Cette couche n'interagit pas directement avec les couches supérieures, elle se contente d'exécuter les opérations de lecture/écriture demandées par Persistence Layer.

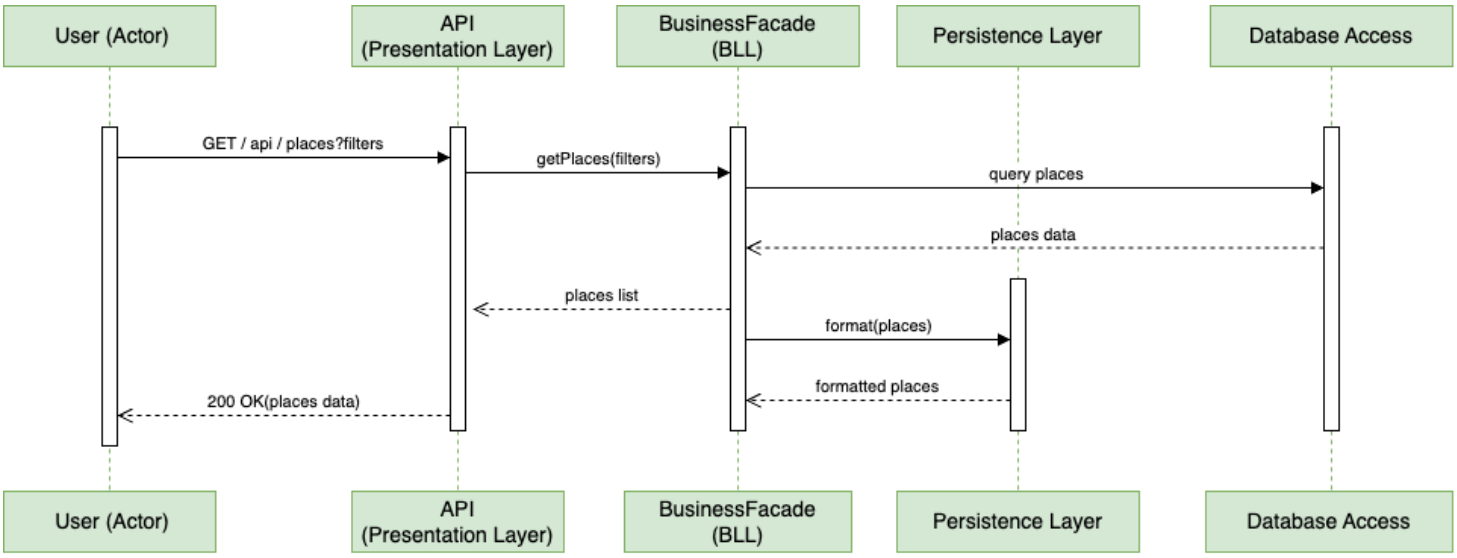
6. Retour de l'information

- Une fois l'avis enregistré, l'information remonte couche par couche jusqu'à l'API.
- L'API renvoie alors une réponse HTTP 201 Created à l'utilisateur, confirmant la création de l'avis, avec éventuellement les données de l'avis nouvellement créé.

Points clés

- **Séparation des responsabilités** : Chaque couche a un rôle précis (présentation, logique métier, persistance, accès aux données), ce qui facilite la maintenance et l'évolution de l'application.
- **Validation à plusieurs niveaux** : Les données sont validées à la fois côté présentation (structure, format) et côté persistance (cohérence métier, contraintes de la base).
- **Flux asynchrone** : Les réponses sont transmises étape par étape, chaque couche ne recevant que l'information nécessaire.
- **Utilisation des principes SOLID** : La conception suit les bonnes pratiques de la programmation orientée objet et de l'architecture logicielle.

Explication du diagramme de séquence : Fetching a List of Places



Ce diagramme illustre le flux de récupération de la liste des "places" (logements) dans l'application HBNB, ce diagramme montre comment les différents composants de l'architecture interagissent pour répondre à une requête de l'utilisateur souhaitant obtenir la liste des logements selon certains filtres.

Voici comment fonctionne ce diagramme étape par étape :

- **User (Acteur) :**
  L'utilisateur (acteur) initie la demande en envoyant une requête HTTP GET à l'API, par exemple : `GET /api/places?filters`.

- **API (Presentation Layer) :**
  L'API reçoit la requête et la transmet à la couche métier (Business Facade) via la méthode `getPlaces(filters)`.
  Cette couche sert d'interface entre l'utilisateur et la logique métier.

- **BusinessFacade (BLL) :**
  La couche Business Logic Layer (BLL) reçoit la demande et appelle la couche de persistance (Persistence Layer) pour interroger la base de données, en passant les filtres reçus.

- **Persistence Layer :**
  Cette couche traduit la demande en une requête adaptée à la base de données et interroge la couche d'accès aux données (Database Access).

- **Database Access :**
  La base de données retourne les données des logements correspondant aux filtres.

- **Retour des données :**
  Les données des logements sont renvoyées à la couche de persistance, puis à la couche métier.
  La couche métier peut alors formater ou transformer ces données si besoin (format(places)).

- **Réponse à l'utilisateur :**
  Les données formatées sont renvoyées à l'API, qui les transmet à l'utilisateur sous forme d'une réponse HTTP 200 OK contenant la liste des logements.

- **Flux de données** :
  Les flèches pleines représentent les appels de méthodes ou de requêtes, tandis que les flèches en pointillés montrent le retour des données ou des réponses.

- **Responsabilités de chaque couche** :
  - L'API gère la communication avec l'utilisateur.
  - La BLL applique les règles métier et orchestre les appels.
  - La couche de persistance gère la communication avec la base de données.
  - La couche d'accès aux données exécute les requêtes sur la base.

- **Respect des principes SOLID** :
  Chaque couche a une responsabilité unique et les dépendances sont dirigées vers l'intérieur, ce qui facilite l'évolution de l'application sans impacter l'ensemble du système