Diagramme UML

Package Diagram – Représentation des architectures en couches

API : point d'entrée du code.

Ce diagramme illustre les différentes couches constituant le back-end de notre future application (HBNB – clone d'Airbnb).

Presentation Layer (couche de présentation) : correspond à l'interface utilisateur. Elle permet de recueillir les entrées ou données de navigation de l'utilisateur via une API REST (le contrôleur).

Business Layer (couche métier) : contient la *Façade* et les *Modèles*.

- **Façade** : gère la logique métier et coordonne les *Modèles* selon les besoins (agit comme intermédiaire entre l'*API* et les *Models*).
- **Model** : applique les opérations CRUD sur la base de données.

Base de données : assure le stockage et la persistance des données.

Class Diagram – Représentation des classes envisagées

- **CoreClass** : classe générale qui regroupe les éléments communs à toutes les autres classes (ex. : ID, opérations CRUD). Son rôle est de limiter les répétitions grâce à l'héritage.
- **User** : hérite des attributs et méthodes de la classe mère. Il possède aussi ses propres attributs spécifiques, publics et privés.
- **Place** : représente les hébergements publiés par les utilisateurs. Elle dépend de l'existence d'un utilisateur (composition).

- **PlaceAmenity** : joue le rôle de classe intermédiaire (*passerelle*), permettant de respecter le principe de responsabilité unique. De plus, cette classe permet de gérer les doubles ID (PlaceId et AmenityID) dans notre future base de données, car il s'agit de deux clés étrangères.

- **Review** : nécessite à la fois un *User* (pour rédiger l'avis) et un *Place* (objet de l'avis). Sans ces deux éléments, le commentaire ne peut exister.

Sequence Diagram – Parcours et traitement d'une requête utilisateur

1. **User (client)** : l'utilisateur navigue dans l'interface et déclenche une action (clic sur un bouton). Une requête HTTP (GET, POST, …) est alors envoyée.
2. **API** : reçoit la requête, valide les données, puis transmet les informations à la *Facade*.
3. **Façade (intermédiaire entre API et logique métier)** : vérifie les contraintes et orchestre les opérations nécessaires en appelant le ou les *Models* appropriés.
4. **Model (logique métier)** : prépare et exécute la requête SQL vers la base de données.
5. **Base de données** : stocke ou récupère les données et renvoie le résultat ou une erreur.

Le traitement des entrées utilisateur reste similaire entre les différents diagrammes. La principale différence concerne les codes de retour HTTP, adaptés en fonction des besoins et des différentes couches.

Codes de retour HTTP

User Register

User vers API :

- **201** : Created (nouvel utilisateur créé)
- **204** : No Content (la ressource demandée n'existe pas en base de données)
- **400** : Bad Request (données invalides ou manquantes)

API vers Logique métier :

- **200** : OK (la requête a été validée)
- **409** : Conflict (ex. : utilisateur déjà existant)

Place

User vers API :

- **201** : Created (nouvel hébergement publié)
- **400** : Bad Request (mauvaises données fournies)
- **409** : Conflict (place déjà existante ou incohérence)

API vers Logique métier :

- **200** : OK

Review

User vers API :

- **201** : Created (nouvel avis ajouté)
- **400** : Bad Request
- **403** : Forbidden (utilisateur non autorisé à poster un avis)
- **409** : Conflict (avis en doublon)

API vers Logique métier :

- **200** : OK

Fetching (consultation de données)

User vers API :

- **200** : OK
- **204** : No Content (aucune donnée trouvée)
- **400** : Bad Request

API vers Logique métier :

- **200 :** Ok