HBnB - Project

Introduction

Purpose

This document serves as a comprehensive technical guide for the development and implementation of the HBnB project. It provides detailed technical specifications, architectural diagrams, API flows, and implementation guidelines to ensure that all stakeholders, developers, and collaborators are aligned throughout the project lifecycle.

Scope

The document covers the core functionalities and technical workflows of the HBnB project, including user registration, place creation, review submission, and fetching a list of available places. It includes class diagrams, sequence diagrams, API flowcharts, and detailed explanations of each module to ensure clarity and consistency throughout the development process.

Overview of the HBnB Project

The HBnB project is a simplified version of a vacation rental platform similar to Airbnb. It allows users to register, list properties, submit reviews, and browse available rentals. The platform emphasizes a seamless user experience for both property owners and travelers. Key functionalities include user management, property listings, booking management, and review systems, all integrated through a robust API architecture.

Role of This Document

This document plays a crucial role in guiding the development process by offering a unified reference for the project's architecture and workflows. It provides detailed insights into the technical components, helping developers and stakeholders collaborate effectively. The document ensures that all team members understand the design principles, coding standards, and expected functionalities, minimizing misunderstandings and streamlining the implementation process.

High-Level Package Diagram

This diagram shows how the application is organised into different layers, each layer, three layer having a specific role in the operation of the system.

1. **First layer**

   Role:
   This is the part that is visible to the user, such as a website or an API. This layer receives user requests (for example, when a user wants to view or add an ad).

   Composant:
   Services and APIs: These are the functionalities that the application offers to the user or to other systems.

   Facade Interface
   Role:
   This is an intermediary that simplifies communication between the presentation layer (which the user sees) and the business logic (which manages the application's rules).

   But: To ensure that the presentation layer does not need to know all the internal details, it simply passes through this interface.

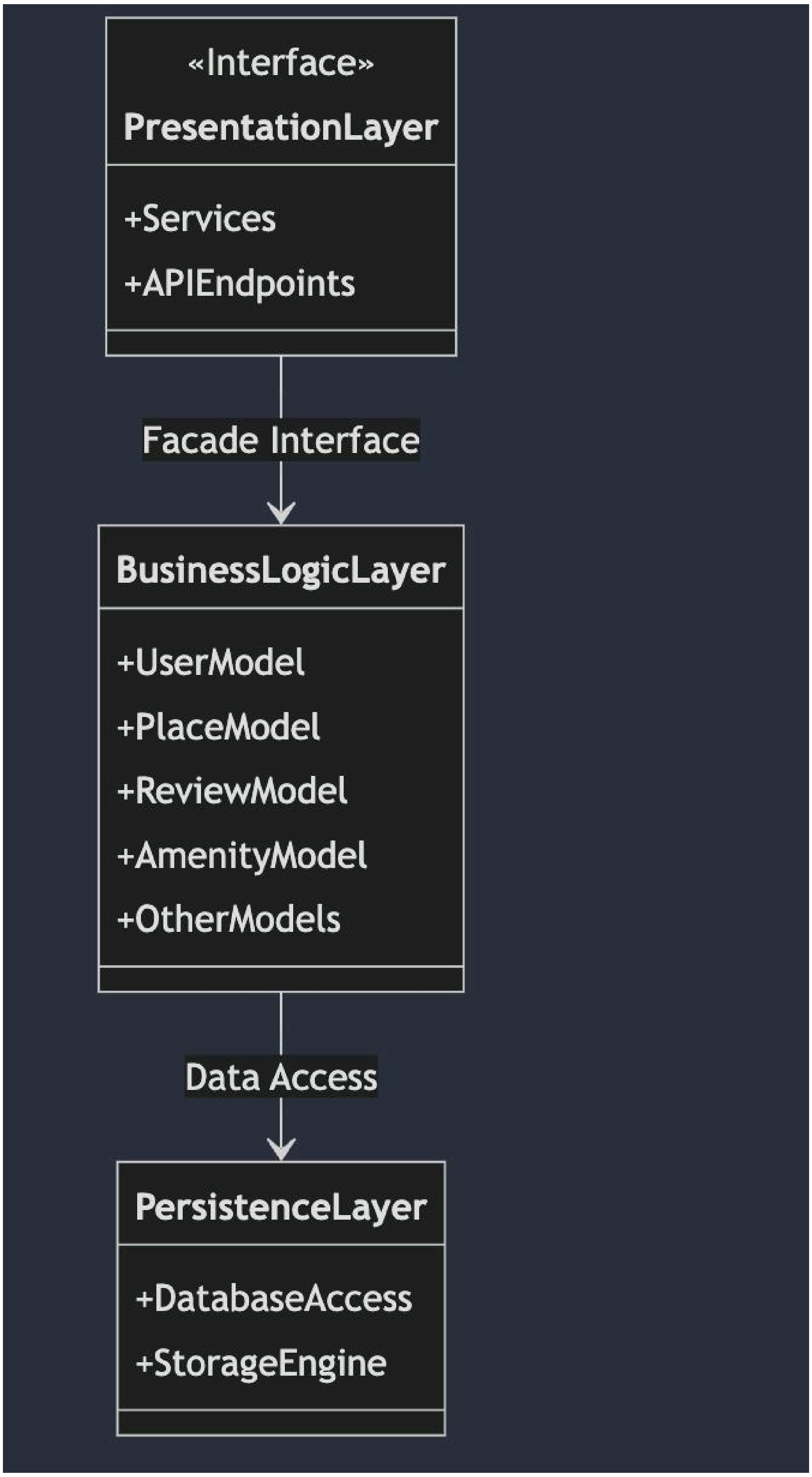
2. **BusinessLogicLayer**
   Rôle : C'est le cerveau de l'application. Elle contient toutes les règles de gestion des annonces, des utilisateurs, des avis, etc.

3. **PersistenceLayer**
   Role: This is where the data is stored (like a database).

   Data access and storage engine: These are used to save and retrieve information (such as adverts or users) in a database.

Conclusion

The presentation (what the user sees) communicates with the front-end to access the business logic (the application rules), which then connects to the database to save or retrieve data


«Interface»
PresentationLayer
+Services
+APIEndpoints
Facade Interface
BusinessLogicLayer
+UserModel
+PlaceModel
+ReviewModel
+AmenityModel
+OtherModels
Data Access
PersistenceLayer
+DatabaseAccess
+StorageEngine

Business Logic Layer: Class Diagram Explanation

The class diagram provided illustrates the core components and relationships of the business logic layer for the HBnB project. The following section details each class and its role in the system.

Classes and Relationships

1. **BaseModel**

   The BaseModel class acts as a foundation for all other entities in the system. It contains the following attributes and methods:

   - `Attributes:`
   - `UUID: A unique identifier for each instance.`
   - `datetime_created_at: The timestamp when the ob`
   - `datetime_update_at: The timestamp when the obj`
   - `Methods:`
   - `save(): Persists changes to the object.`
   - `update(): Updates the timestamp when changes a`

This class is inherited by all other classes in the diagram, providing a consistent way to track creation and modification times for objects.

2. **Class User**

   The User class represents individuals using the HBnB platform. Users can own places and leave reviews for places they visit.

   - `Attributes:`
   - `FirstName: The user’s first name.`
   - `LastName: The user’s last name.`
   - `Email: The user’s email, used for authenticati`
   - `Password: The user’s password, securely stored`
   - `Methods:`
   - `create_user(first_name, last_name, email, pass`
   - `authenticate(email, password): Validates the u`
   - `list_owner_places(): Retrieves a list of place`

Each user can be associated with multiple places, as indicated by the relationship with the Place class.

3. **Class Place**

The Place class represents rental properties listed on the platform. Each place is associated with a specific user (the owner) and has numerous attributes detailing the property.

- Attributes:
- name: The name of the place.
- description: A description of the place.
- country: The country where the place is locate
- address: The address of the place.
- price: The price per night to rent the place.
- latitude and longitude: The geographical coord
- owner: The user who owns the place.
- numRooms: The number of rooms in the place.
- numBathroom: The number of bathrooms in the pl
- max_guests: The maximum number of guests allow
- reviews: Reviews left by users for this place.
- amenities: The amenities available at the plac
- Methods:
- get_info(): Returns all the information about
- list_reviews(): Lists all reviews associated w
- list_amenities(): Lists all amenities availabl
- create_place(): A method to create a new place

Each place is linked to a User who owns it and can be reviewed by other users. Places also have relationships with amenities and reviews.

4. **Class Review**

   The Review class allows users to leave feedback for places they have visited. Each review is associated with both a User and a Place.

- Attributes:
- User_id: The user who wrote the review.
- Place_id: The place being reviewed.
- Rating: An integer rating provided by the user
- Comment: A text comment left by the user as pa
- Methods:
- get_user_review(): Retrieves the review writte
- get_place_review(): Retrieves all reviews for
- add_review(user, place, rating, comment): Adds

Reviews are critical for users to make informed decisions about places to stay. They contribute to the overall rating and reputation of a place.

5. **Class City**

The City class provides information about cities that have places listed on the platform. Each city is associated with a country.

- Attributes:
- Name: The name of the city.
- Country_id: The identifier of the country to w
- Methods:
- get_city_info(): Returns detailed information
- list_places(): Lists all places available in t

Cities are a key entity, helping users locate places based on geographical preferences.

6. **Class Country**

   The Country class represents different countries on the platform, each associated with one or more cities.

- Attributes:
- Name: The name of the country.
- Code: The country code (e.g., “US” for the Uni
- Methods:
- get_country_info(): Provides detailed informat
- list_cities(): Lists all cities within the cou

The Country class allows users to filter places and cities based on the country they are interested in visiting.

7. **Class Amenity**

   The Amenity class represents amenities that can be associated with a place. Examples include Wi-Fi, air conditioning, pool access, and more.

- Attributes:
- Amenity_id: The unique identifier for the amen
- Amenity_Name: The name of the amenity.
- Methods:
- get_info(): Retrieves information about a spec
- list_places(): Lists all places that offer the

Amenities help users refine their search for a place by specifying the features they are looking for in a rental.

Design Decisions

```
•   Inheritance: By using a common BaseModel, the desigr
•   Encapsulation: Each class has its own methods to enc
•   Scalability: The relationships between places, citie
```

Conclusion

The class diagram and the relationships between entities provide a well-structured foundation for the HBnB project. Each class has clearly defined responsibilities and attributes, and the use of inheritance and encapsulation ensures that the design is both scalable and maintainable.

Class Diagram

API Interaction Flow: Sequence Diagram for User Registration

The diagram illustrates the sequence of interactions that occur when a user registers through the API. This is a typical example of the user registration process in the HBnB project. The sequence involves multiple components: User, API, Business Logic, and the Database.

Sequence of Events

1. **User Interaction**:
   - The user initiates a `POST /register` request to the API with their registration details (e.g., name, email, password).
2. **API Layer**:
   - Upon receiving the request, the API forwards the user data to the Business Logic layer for validation. The API acts as an intermediary between the user and the back-end services.
3. **Business Logic Layer**:
   - **Validate User Data**: The business logic is responsible for validating the registration data. It checks if the data meets the required criteria (e.g., valid email, strong password).
   - **Save User Data**: Once validated, the business logic forwards the user data to the database for storage.

4. **Database Layer**:
    - The database stores the user data and returns a confirmation of the save operation back to the business logic.
5. **Response Handling**:
    - The business logic confirms the success or failure of the registration process and returns the appropriate response (e.g., success message, error message) to the API.
6. **API Returns Final Response**:
    - The API sends a response back to the user indicating whether the registration was successful or failed (e.g., due to validation errors).

Explanatory Notes

- **Purpose**: This sequence diagram demonstrates the flow of data during a user registration request. It highlights the interactions between the different components (User, API, Business Logic, Database) involved in this process.
- **Key Components**:
    - **User**: The end-user making the registration request.
    - **API**: Acts as the entry point for the user request, forwarding data to the business logic and sending the response back to the user.
    - **Business Logic**: Handles the core logic of validating and processing user data.
    - **Database**: Responsible for storing user data securely.
- **Design Decisions**:
    - **Data Flow**: The data flows in a logical sequence from the user input, through validation and processing, and finally to the database for storage.
    - **Layered Architecture**: This design ensures separation of concerns. The API handles requests, the business logic processes the data, and the database stores the information.
    - **Error Handling**: In case of validation failure or database issues, error messages can be returned to the user, maintaining the integrity of the registration process.
- **How it Fits into the System**:

- The registration process is fundamental to any user-based system, ensuring that new users can sign up, and their data is stored correctly in the system for future authentication and use. This process is essential to the functioning of the HBnB platform.

Conclusion

This sequence diagram for user registration provides a clear representation of how the registration flow is managed within the HBnB project. By separating the responsibilities between the API, Business Logic, and Database, the design ensures scalability, maintainability, and reliability in handling user registrations.

User Registration Diagram

API GET Place

Sequence diagram, which shows how the different entities (or components) interact in a certain order to accomplish a task. In this case, it illustrates a scenario where a user makes an API request to obtain a list of "places"

User: This is the person who uses the application and makes a request to obtain information about available ads.

API: This is the interface that allows the user to communicate with the application backend via HTTP requests (in this case, a GET /places request).

BusinessLogic: This is the layer that contains all the application's logical rules and decisions. It takes the request from the API, applies filters or search criteria, and communicates with the database to obtain the information requested.

Database: The database stores all the information about the ads. The business layer interacts with the database to retrieve the list of ads matching the criteria.

Description of the interaction flow:

GET /places request: The user sends an HTTP GET /places request to the API. This request asks the API to provide a list of available rental listings.

Processing in the API: The API receives the request and forwards it to the Business Logic layer for processing. The API does not do the processing itself, it only acts as an intermediary.

Filtering by business logic: The Business Logic layer takes the query criteria (for example, filters on location or price) and prepares a query for the database to find the ads that match these criteria.

Answer 200 OK: Once the user has received the list of ads, the API sends an HTTP 200 OK response, which means that the request was successful.

API GET place

API POST places

This diagram is a sequence diagram showing the communication flow between different entities for adding a new ad (place) to a system via a POST /places API request.

Diagram components:

User: It is the user who uses the application to submit a new ad. 2.

API: The interface that allows the user to communicate with the backend via HTTP requests (in this case, a POST /places request).

BusinessLogic: This is the layer that contains the business logic, where validations and business rules are processed before the data is saved in the database.

Database: The database is where the ad data is saved.

Explanation of the interaction flow:

POST request /places: The user sends an HTTP POST /places request to the API to create a new ad (place). This request contains the data for the advert to be added. 2.

Validation and processing of data by the API: The API receives the request and forwards it to the Business Logic layer for processing. The API acts as an intermediary and does not process the data itself.

Validation and processing of data by the business logic : Business Logic receives the ad data, validates it (checking that the information is correct, complete, etc.), then prepares it for saving.

Backing up data in the database: Business Logic sends the validated data to the database for backup. This means that the advert will be saved in the system.

Confirm backup: Once the database has saved the data, it sends a confirmation to the Business Logic, indicating that the operation has been carried out successfully.

Return of the response to the API: The Business Logic returns a response to the API, confirming that the new advert has been added to the database

Return response to user:

The API returns a response to the user indicating whether the operation was successful or unsuccessful. If the backup was successful, the user receives a success confirmation, otherwise an error message.
API POST places

API POST /reviews Sequence Diagram

User Sends POST /reviews Request

User:
This is the person who uses the application and submits a review for a place.

API:
This is the interface that allows the user to communicate with the backend via HTTP requests (in this case, a POST /reviews request).

BusinessLogic:
This layer contains all the business logic of the application, such as validations and decisions, which processes the review data sent by the API.

Database:
The database stores all information about reviews. The business logic interacts with the database to save the review.

Description of the interaction flow:

POST /reviews request

The user sends an HTTP POST request to the API to submit a review. This request includes data such as the rating, review text, and user ID.

Processing in the API

The API receives the request and validates the review data. It checks if all required fields are provided, ensures that the data types are correct (e.g., the rating is a number), and verifies that the user has permission to submit the review.

Forwarding Data to Business Logic

After the validation, the API sends the review data to the BusinessLogic layer. The API acts as an intermediary and does not handle the processing itself.

Processing and Saving Data by Business Logic

The BusinessLogic layer processes the review, applying any relevant business rules or additional checks (e.g., verifying if the user has already submitted a review for the same place). Once processed, it sends the data to the database to be saved.

Backing up Data in the Database

The BusinessLogic layer sends the validated review data to the database for storage. The data includes information such as user ID, review content, and timestamps.

Database Confirms Save

Once the review data is saved, the database sends a confirmation back to the BusinessLogic layer, indicating that the operation was successful.

Returning Response to the API

The BusinessLogic layer sends a response to the API, indicating whether the review submission was successful or if there was an issue (e.g., validation failure or database error).

Returning Response to the User.

The API sends a final response back to the user, informing them of the outcome. If successful, the response might be something like "Review submitted successfully"; otherwise, the user may receive an error message.

API POST reviews