HBnB – UML

Introduction :

Technical Documentation - HBnB Evolution Introduction

HBnB Evolution is an AirBnB inspired web application designed to allow users to manage property rentals online.

This technical documentation presents the architecture, design and key components of the system. Document Purpose This documentation aims to provide a thorough understanding of the architecture and operation of HBnB Evolution.

It will serve as a primary reference for : Developers working on the implementation of the system Architects wishing to understand the design choices Maintenance teams for future evolutions Project Scope HBnB Evolution includes four main functionalities:

User Management : Registration, profile update, and role management (regular users and administrators) Property Management : Publication, modification and deletion of rental listings Rating System : Creation and management of property reviews Amenity Management : Administration of amenities associated with properties General Architecture.

The system is built according to a three-layer architecture :

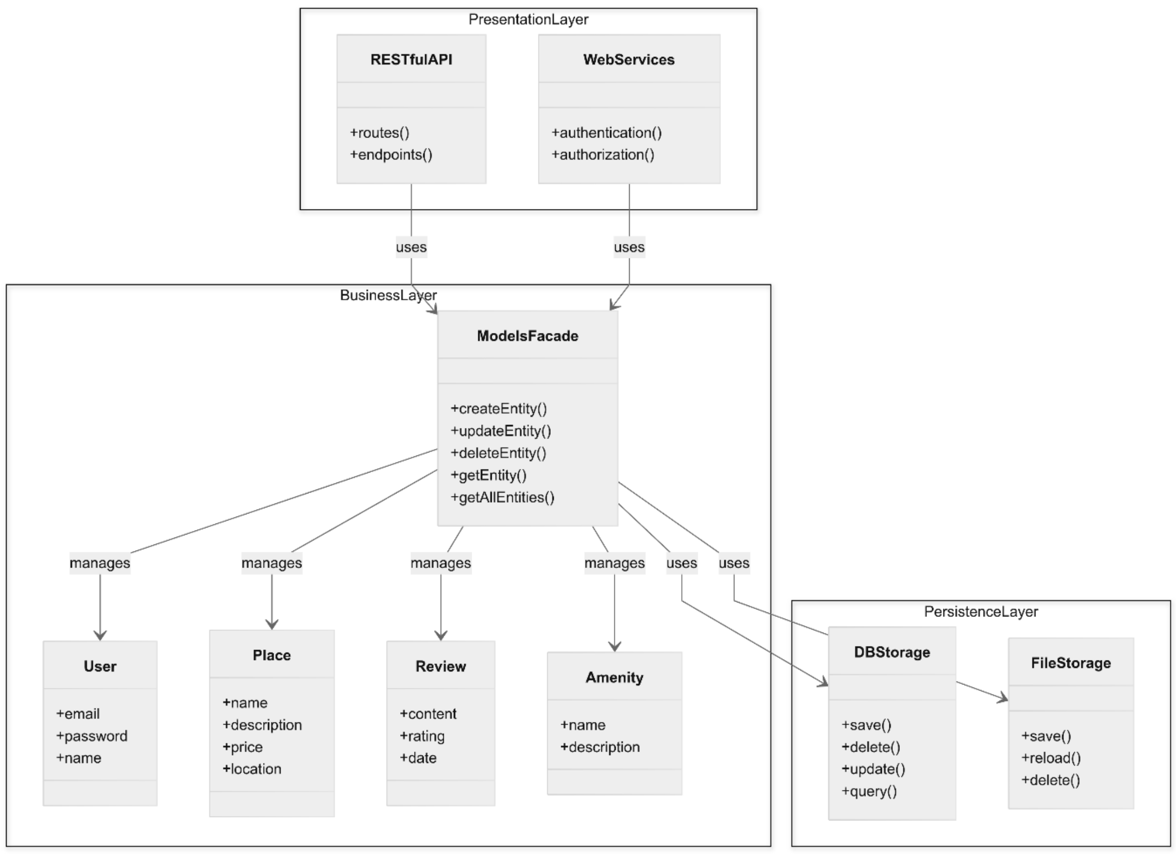
-Presentation Layer : User interface and API services

-Business Layer : Business logic and data models

- Persistence Layer : Data storage and retrieval Documentation Structure.

This documentation is organized into several sections : Package Diagram : Architecture overview Class Diagram : Layer detail business Sequence Diagrams: Interaction Flows for Key Operations This architecture was designed to ensure: A clear separation of responsibilities A simplified maintenance An optimal scalability A robustness in data management.

High-Level Architecture



Objective of this diagram :

This diagram is intended to provide a conceptual overview of how the different components of the HBnB application are organized and how they interact with each other.

Explanation of the diagram :

The diagram is differentiated with 3 distinct parts : Presentation Layer (Services, API), Business Logic Layer (Models) and Persistence Layer.

- <u>In the first part, Presentation Layer (Services, API) :</u>

There are 2 distinct parts, the RestfullAPi who handles HTTP requests and responses and the Webservices who takes care of authentication and authorization.

These two components communicate with the business layer via the façade (ModelsFaçade).

- <u>In the second part, the business layer :</u>

For this part we have 4 sub-parts that contain different element management :

- User : User management
- Place : Venue Management
- Review : Review Management
- Amenity : Asset Management

- <u>In the in the third part, the Persistence Layer :</u>

In this part we have 2 parts the DBstorage and the Filestorage.

- The Dbstorage is a Database
- The Filestorage storing a files

Both storage classes provide consistent interfaces for data operations.

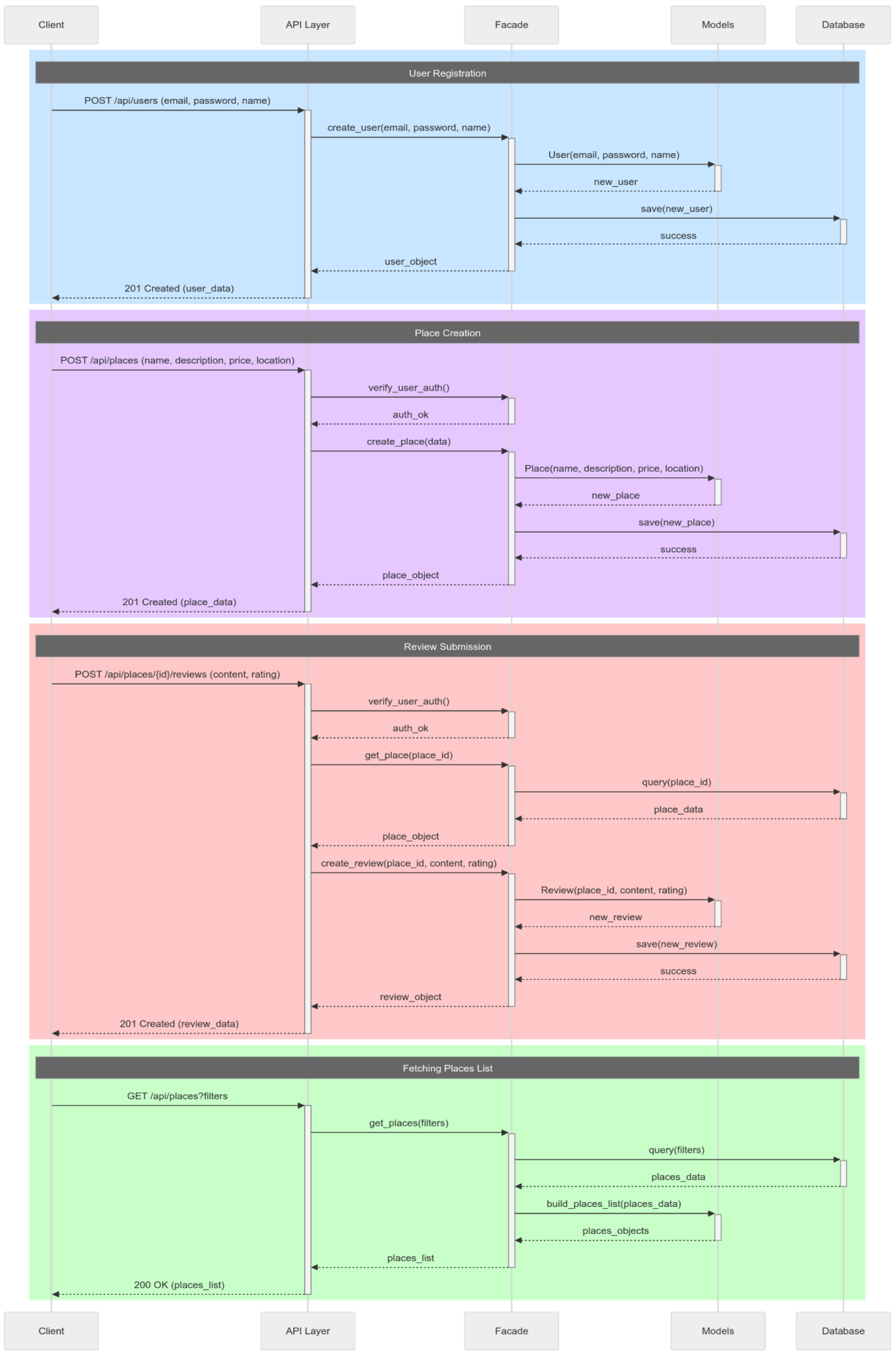
<u>Why we use this pathern ?</u>

- Simplifies the interface between layers
- Encapsulates complex interactions
- Provides a single entry point for the presentation layer
- Clear segregation of responsibilities

<u>Relationships :</u>

- Dotted arrows (..): indicate dependencies
- Solid arrows (-): represent direct relationships
- All communication between the layers passes through the façade

Sequence Diagrams for API Calls



Introduction for api call sequence diagram. this is a sequence diagram that illustrates 4 operations in 4 parts using RESTfull API type API that uses http.

<u>Here are the 4 steps of an api call:</u>

- The request
- The processing
- The response
- The display

<u>The first is the user registration in (blue):</u>

- The client sends the information using POST
- The API layer transmits the information to facade to create the user
- facade creates a new user via models
- We return the confirmation

<u>The second is the property creation in (purple):</u>

- The client sends the information using POST
- We verify the identification
- We create the location
- We save in the database
- We return the confirmation

<u>The third is the submission of reviews in (pink):</u>

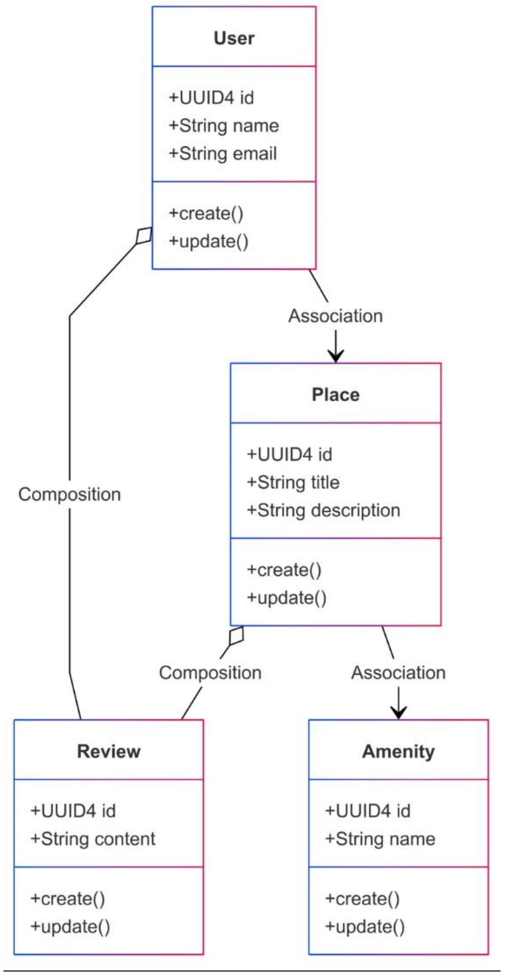
- The client sends the review using POST
- We verify the identification
- We retrieve the property details
- We create the review
- We save in the database
- We return the confirmation

<u>The fourth is the filtering of properties in (green):</u>

- The client filters with GET
- We retrieve the filtered data
- We build the list
- We return the results

Each operation follows a logical flow of validation, processing and response.

Class Diagram for Business Logic Layer



Objective of this diagram

This UML diagram represents the architecture of a place evaluation and management system.

This diagram contains 4 main objectives :

1) User Management and Authentication
2) Venue Management
3) Evaluation System
4) Amenities Management

<u>Composing</u>

<u>The diagram shows 4 main classes :</u>

- The Users
- The Place
- Review
- Amenity

<u>Explanation of the diagram</u>

- **<u>At the top we have the User :</u>**

Representing the users of the system. Each user is characterized by three essential attributes: a unique identifier in UUID4 format, a name (String name) and an email address (String email). This class provides two basic methods: create() for creating a new user, and update() for editing existing information.

- **<u>After we have the place :</u>**

The Place class defines places . A place is identified by a unique UUID4 and includes a String title and a String description.

- **<u>The Review Class :</u>**

Manage user reviews. It is linked by compositional relations to both User and Place, each review has a UUID4 identifier and a textual content (String content).

- **<u>The Amenity class :</u>**

Represents the facilities or services available in the premises. An amenity is characterized by a UUID4 identifier and a String name.