UML Documentation

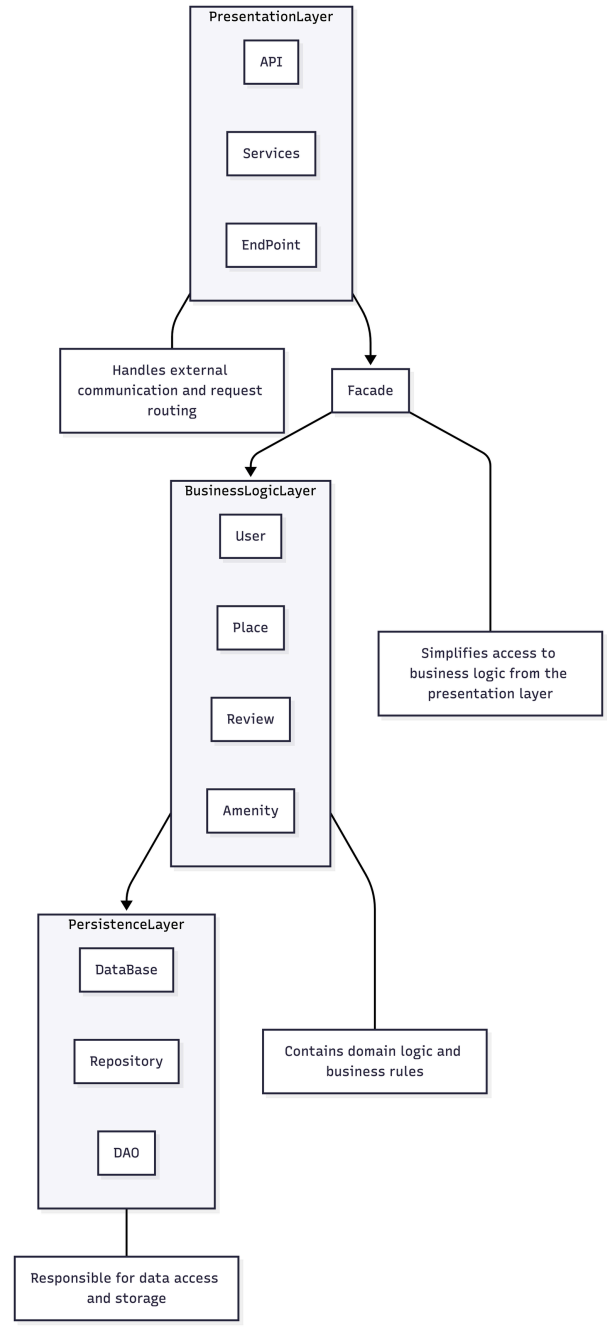
Introduction

Unified Modeling Language (UML) is a standardized modeling language used to visualize, specify, construct, and document the components of a software system. It provides a set of diagrams that represent different aspects of a system's architecture and behavior.

1. Package Diagram

Overview

The Package Diagram illustrates the high-level organization of the system into logical layers or modules. It shows how different packages interact and depend on each other, promoting modularity and separation of concerns.



Description

The system is divided into three main layers:

1. PresentationLayer
   - Components: API, Services, EndPoint
   - Responsibility: Handles external communication, user requests, and routing. It acts as the entry point for clients interacting with the system.
2. BusinessLogicLayer
   - Components: User, Place, Review, Amenity
   - Responsibility: Contains the core business logic and rules. It processes data received from the presentation layer and coordinates with the persistence layer for

data operations.

3. PersistenceLayer
    - Components: Database, Repository, DAO (Data Access Object)
    - Responsibility: Manages data access and storage. It abstracts database operations and provides a consistent interface for data retrieval and manipulation.

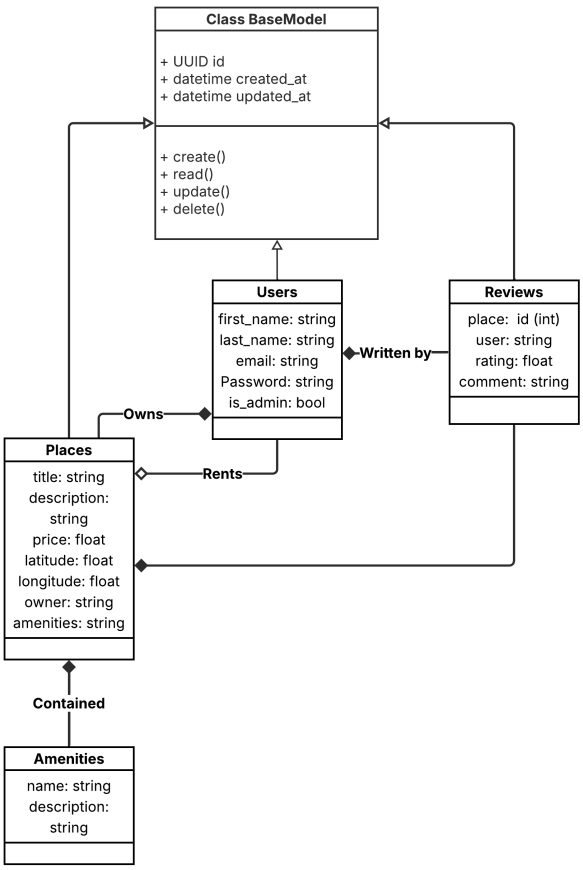
Relationships

- The PresentationLayer communicates with the BusinessLogicLayer through a Facade, which simplifies access to business logic and hides internal complexities.
- The BusinessLogicLayer interacts with the PersistenceLayer to perform data-related operations.
- The dependencies flow downward, ensuring a clear separation of responsibilities and maintainability.

2. Class Diagram

Overview

The Class Diagram provides a detailed view of the system's structure by showing classes, their attributes, methods, and relationships. It defines how objects are structured and how they interact within the system.



Description

The class diagram defines the following main classes:

1. BaseModel
    - Attributes:
        - id: UUID
        - created_at: datetime
        - updated_at: datetime
    - Methods:
        - create()
        - read()
        - update()
        - delete()
    - Description: Serves as the parent class for all other entities, providing common attributes and CRUD operations.
2. Users
    - Attributes:
        - first_name: string
        - last_name: string
        - email: string
        - password: string
        - is_admin: bool
    - Description: Represents system users, including both regular users and

administrators. Inherits from BaseModel.

3. Places
    - Attributes:
        - title: string
        - description: string
        - price: float
        - latitude: float
        - longitude: float
        - owner: string
        - amenities: string
    - Description: Represents a property or location that can be owned or rented by users. Inherits from BaseModel.

4. Reviews

    - Attributes:
        - place: id (int)
        - user: string
        - rating: float
        - comment: string
    - Description: Represents user feedback on places. Each review is associated with a user and a place. Inherits from BaseModel.
5. Amenities
    - Attributes:
        - name: string
        - description: string
    - Description: Represents features or services available at a place. Inherits from BaseModel.

Relationships

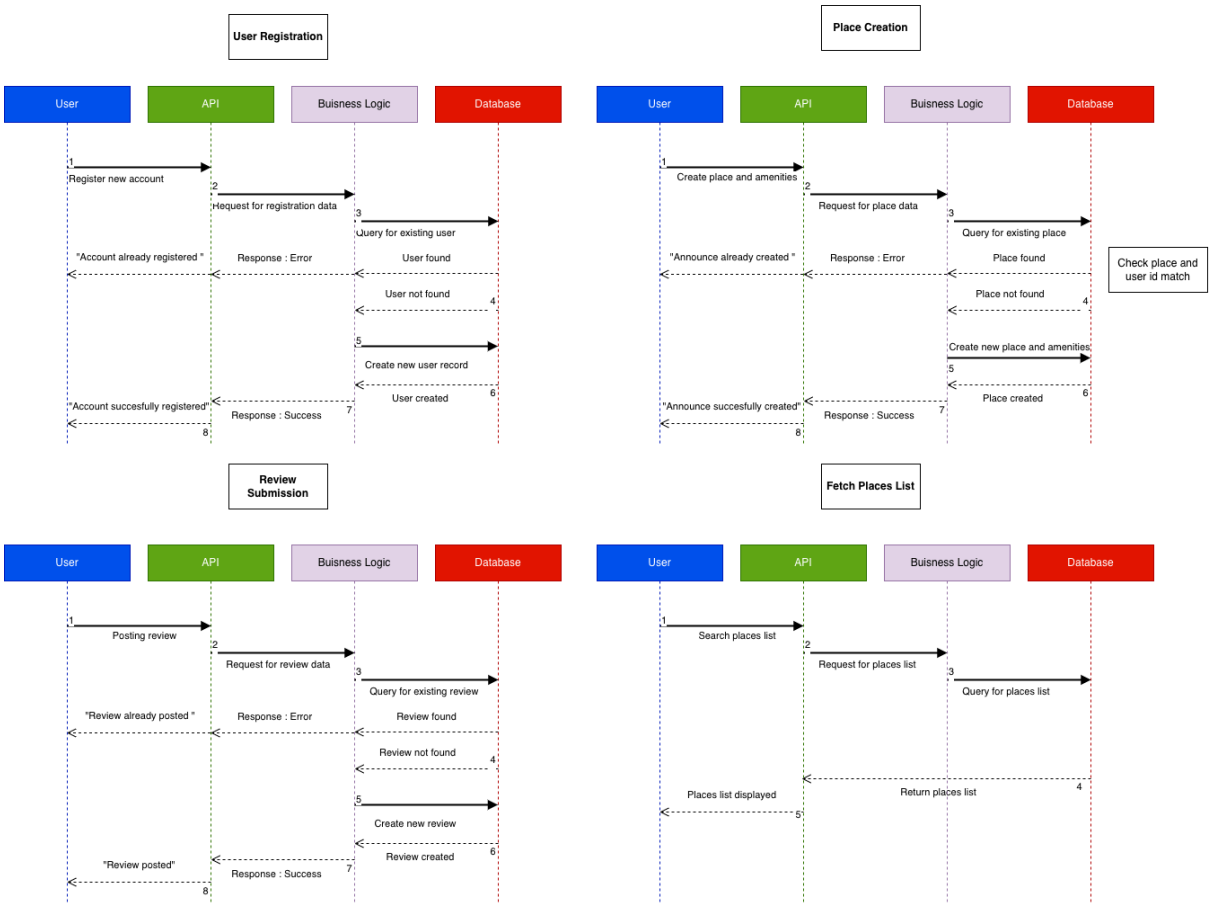
- Users → Places: A user owns one or more places.
- Users → Reviews: A user writes one or more reviews.
- Places → Amenities: A place contains one or more amenities.
- Places → Users: A place can be rented by a user.
- Places → Reviews: A place can have multiple reviews written by different users.
- All classes inherit from BaseModel, ensuring consistency in data management and operations.

3. Sequence Diagram

Overview

The Sequence Diagram models the dynamic behavior of the system by showing how objects interact over time to accomplish specific tasks. It focuses on the order of messages

exchanged between components.



Description

The sequence diagram illustrates four main scenarios that represent typical system interactions:

a. User Registration

1. The User initiates a registration request through the API.
2. The API forwards the request to the Business Logic layer.
3. The Business Logic queries the Database to check if the user already exists.
4. If the user exists, an error response is returned.
5. If not, a new user record is created in the Database.
6. The API sends a success message back to the User confirming registration.

b. Place Creation

1. The User sends a request to create a new place and its amenities through the API.
2. The API forwards the request to the Business Logic layer.
3. The Business Logic queries the Database to check for existing places.
4. If a duplicate is found, an error response is returned.
5. If not, the Business Logic creates a new place and associated amenities in the

Database.
6. A success message is returned to the User confirming the creation.

c. Review Submission

1. The User submits a review through the API.
2. The API sends the request to the Business Logic layer.
3. The Business Logic queries the Database to check if a review already exists for the same user and place.
4. If a review exists, an error response is returned.
5. If not, a new review is created and stored in the Database.
6. The API returns a success message to the User confirming the review submission.

d. Fetch Places List

1. The User requests a list of available places through the API.
2. The API forwards the request to the Business Logic layer.
3. The Business Logic queries the Database for all available places.
4. The Database returns the list of places.
5. The API sends the list back to the User for display.

Key Interactions

- Each scenario demonstrates the layered communication flow: User → API → Business Logic → Database.
- The API acts as the intermediary between the user interface and the backend logic.
- The Business Logic layer ensures validation, rule enforcement, and coordination with the Database.
- The Database layer handles persistent data storage and retrieval.

Conclusion

The combination of the Package, Class, and Sequence diagrams provides a comprehensive understanding of the system's architecture. The package diagram defines the modular structure, the class diagram details the internal composition and relationships, and the sequence diagram illustrates runtime interactions across multiple use cases. Together, they ensure a well-organized, maintainable, and scalable software design.