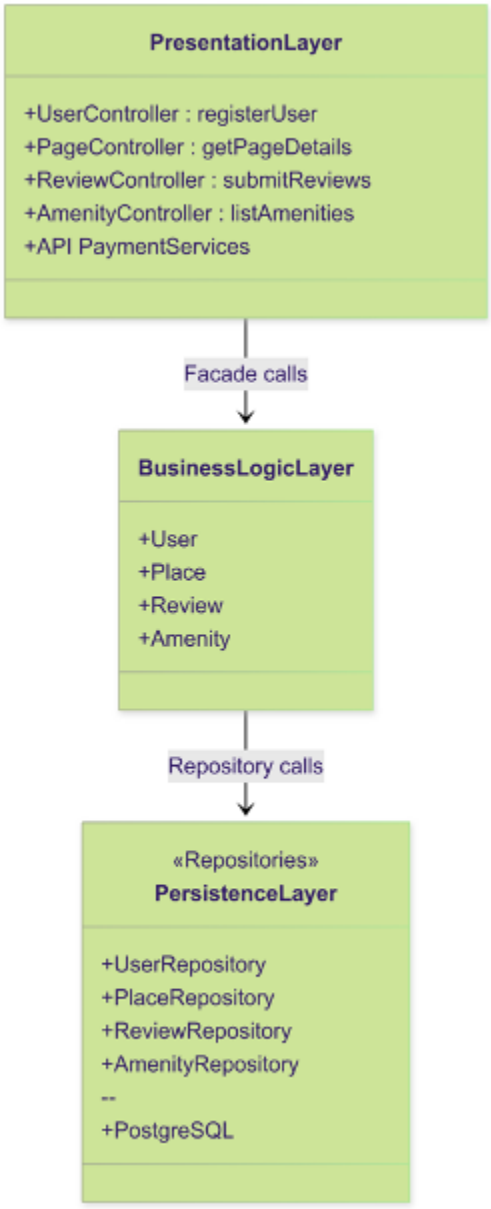
1. Introduction:

Hbnb is an Airbnb-like application that allows users to create and manage listings, reviews, and amenities

This project includes several diagrams representing different aspects of the application architecture, such as:

- A high-level package diagram
- A detailed class diagram of the business layer
- A sequence diagram illustrating API interactions

2. High-Level Architecture:



Purpose:

- Illustrates the overall structure of the HBnB Evolution system, showing how the presentation, business logic, and persistence layers interact.

Key components:

- **Presentation Layer:** Manages user interactions and API endpoints.
- **Business Logic Layer:** Processes data, enforces rules, and facilitates communication.
- **Persistence Layer:** Stores and manages application data reliably.

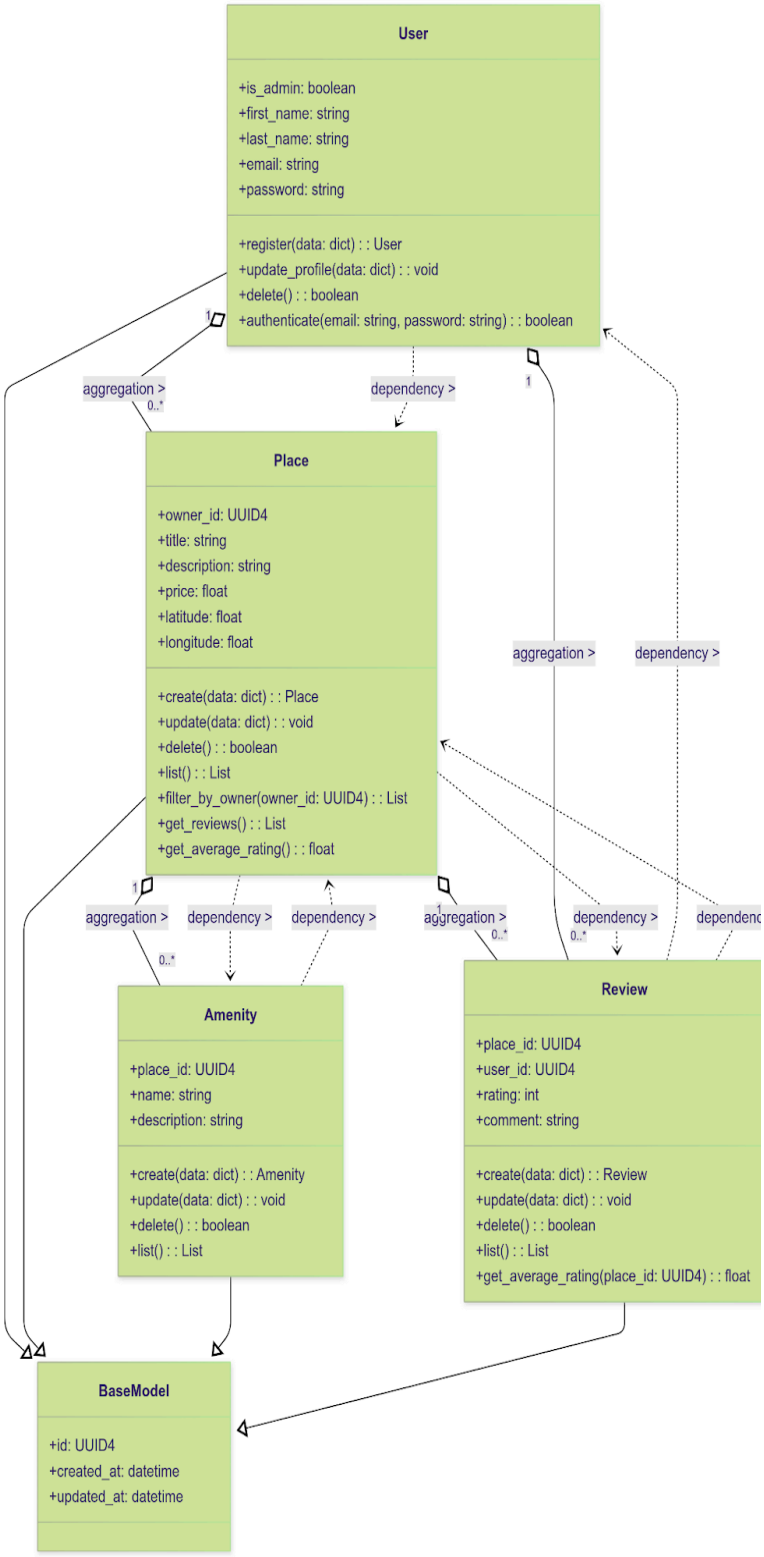
Design Decisions & Rationale:

- **Designed a three-layer architecture** to enhance modularity and simplify maintenance
- **Applied a facade pattern in the Presentation layer** to streamline access to business operations.

Role in overall Architecture:

- **Create the core structure of the application,** directing how components interact and how the system operates as a whole.

3. Business Logic Layer:



Purpose:

Illustrates how classes within the business logic layer collaborate to drive the application's key features.

Key components:

- **User:** handles user details, authentication and relations
- **Place:** contains rental listings and their attributes
- **Review:** manages feedback and ratings
- **Amenity:** defines features like wi-fi, swimming pool, saunas

Design Decisions & Rationale:

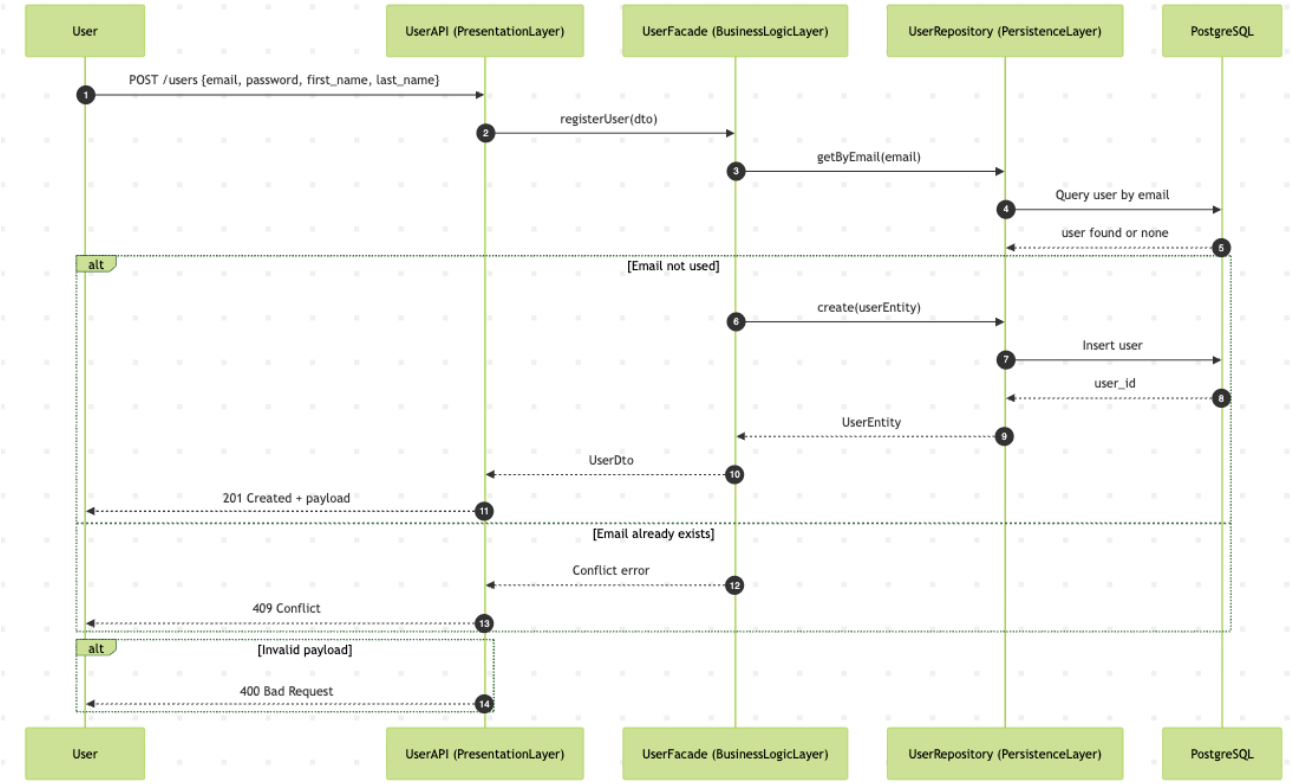
- Applied **object-oriented principles to simplify logic**
- Used **service-oriented approach** to isolate functionality and integrate with APIs

Role in overall Architecture:

- Implements the core application logic, ensuring the business rules and workflows are applied correctly.
- Facilitates the communication between the presentation and persistence layers, coordinating data flow.

4. API Interaction Flow: Sequence Diagrams for API calls with explanations.

4.1 User Registration Sequence Diagram - POST/users



Purpose:

This diagram illustrates the process of registering a new user in the system.

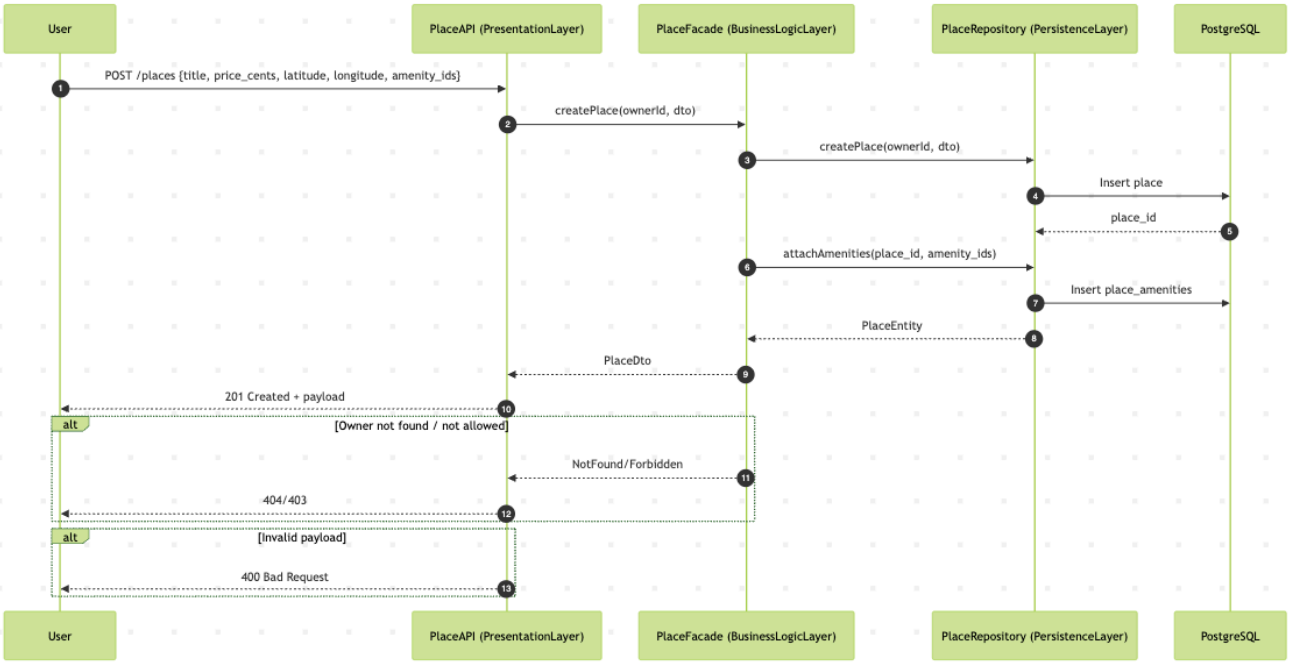
Key Components Involved

- **User:** Initiates the registration request with email, password, and personal details.
- **UserAPI (Presentation Layer)**: Receives the registration request, performs basic validation, and forwards the data to the business logic layer.
- **UserFacade (Business Logic Layer)**: Orchestrates the registration process, enforces business rules (e.g., email uniqueness), and delegates persistence operations to the repository.
- **UserRepository (Persistence Layer)**: Handles database queries and insertions, such as checking whether an email already exists and creating new user records.
- **PostgreSQL (Database)**: Stores the user data securely, including credentials and personal information.

Error cases:

- **400** Bad Request for invalid input.
- **409** Conflict when the email is already in use.

4.2 Create Place Sequence Diagram - POST/places



Purpose:

This diagram demonstrates the process by which a registered user creates a new place listing in the system. It outlines the end-to-end workflow.

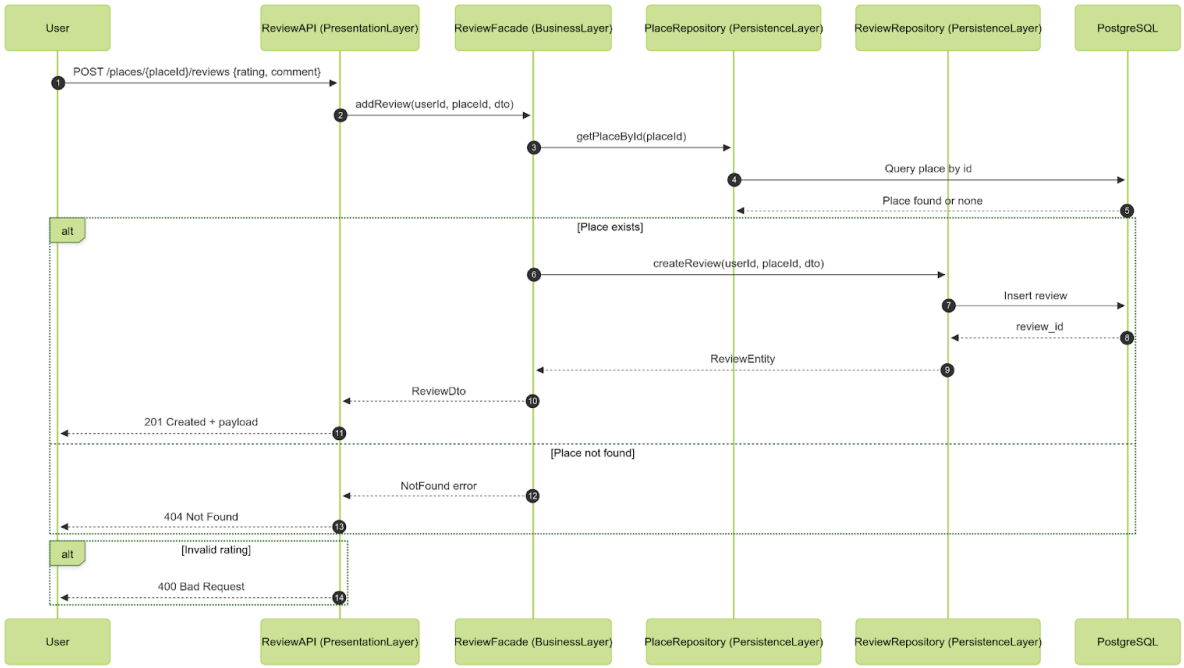
Key Components Involved

- **User**: Initiates the creation request by providing place details (title, price, coordinates, amenities).
- **PlaceAPI (Presentation Layer)**: Receives the HTTP request, validates input format, and forwards the data to the business logic layer.
- **PlaceFacade (Business Logic Layer)**: Orchestrates the workflow, ensuring that the owner is valid, the input is consistent, and repository calls are coordinated.
- **PlaceRepository (Persistence Layer)**: Handles database operations, including inserting the new place and linking it with amenities.
- **PostgreSQL (Database)**: Stores the place records and associated amenity mappings.

Error cases:

- **404** Not Found or 403 Forbidden if the owner is invalid or lacks permission.
- **400** Bad Requests if the payload is malformed or missing required attributes.

4.3 Add Review Sequence Diagram – POST /places/{id}/reviews



Purpose:

This Diagram illustrates a user to leave a review for a place, including rating and comment. It captures both the **successful review creation** and **error scenarios**. The diagram ensures clarity on how data flows across the layers of the application when a user submits a review.

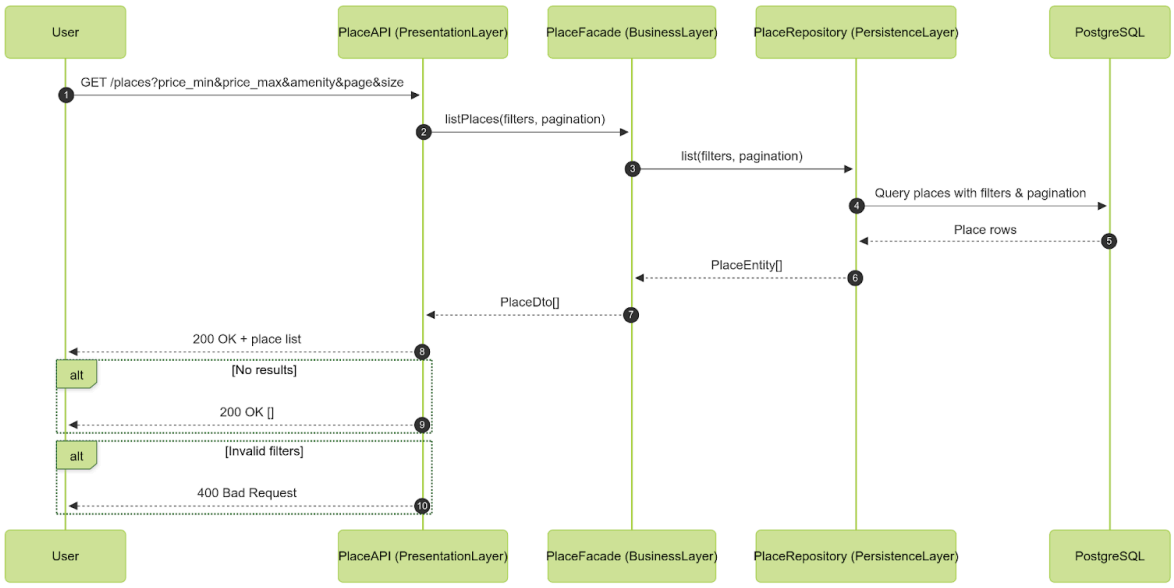
Key Components Involved:

1. **User**: Initiates the review submission by sending a POST request.
2. **ReviewAPI (Presentation Layer)**: Validates request format, parses input, and passes the request into the business logic.
3. **ReviewFacade (Business Logic Layer)**: Orchestrates the process. It checks that the target place exists and delegates persistence operations to the repositories.
4. **PlaceRepository (Persistence Layer)**: Verifies the existence of the target place by querying the database.
5. **ReviewRepository (Persistence Layer)**: Inserts the review record into the database if validation passes.
6. **PostgreSQL**: Stores the actual data (places and reviews).

Error Cases:

- **404** Not Found: Place does not exist.
- **400** Bad Request: Rating is outside allowed range (1–5) or payload invalid.

4.4 Fetch Places Sequence Diagram– GET /places?filters



Purpose:

This diagram returns a paginated list of places, optionally filtered by price or amenities. It shows the interaction between the different layers of the application when handling a GET /places request.

Key Components Involved:

1. **User**: request places with filters (price range, amenity, pagination).
2. **PlaceAPI (Presentation Layer)**: parses query params and calls Business Logic.
3. **PlaceFacade (Business Logic Layer)**: Validates and coordinates the filtering logic, delegating persistence operations to the repository.
4. **PlaceRepository (Persistence Layer)**: queries the DB and returns matching places.
5. **PostgreSQL (Database)**: Stores all place records and processes the SQL queries executed by the repository.

Error Cases:

- **200 OK []**: Valid query but no results match.
- **400 Bad Request**: Invalid filters (e.g. negative price, bad amenity value).

4.5 Design decision & Rationale

The design of the four sequence diagrams — ***Register User***, ***Create Place***, ***Add Review***, and ***Fetch Places*** — reflects a consistent set of architectural principles and patterns. Several key decisions were made to ensure the system remains robust, modular, and extensible.

- **Layered Architecture**

  All use cases follow a strict three-layer design:

  - **Presentation Layer (API)** for receiving HTTP requests, validating input, and formatting responses.
  - **Business Logic Layer (Facades)** for enforcing rules, coordinating workflows, and orchestrating persistence operations.
  - **Persistence Layer (Repositories)** for interacting with the database.

- **Facade Pattern for Simplified Interaction**
  - The Facade classes (**UserFacade, PlaceFacade, ReviewFacade**) act as a unified entry point for the Business Logic layer.
  - This pattern ensures that the Presentation Layer never directly handles internal logic or multiple repositories, making the system easier to maintain and extend.

- **Repository Pattern for Database Access**
  - Repositories (UserRepository, PlaceRepository, ReviewRepository) are responsible for CRUD operations.
  - This design abstracts raw SQL/ORM calls, allowing changes in the persistence mechanism without impacting the higher layers.
  - Returning **Entities** from repositories and mapping them to **DTOs** in the facade maintains a clean boundary between internal models and external responses.

- **Error Handling:**

  Each sequence diagram explicitly shows **failure scenarios**:

  - **400 Bad Request** → invalid input (e.g., rating out of range, missing fields, invalid filters).
  - **404 Not Found** → resource does not exist (e.g., place not found, user not found).

- **409 Conflict** → duplicate entry (e.g., registering an email that already exists).

❖ **Consistency Across Diagrams**

- All diagrams adopt the same **naming conventions** (XXXAPI, XXXFacade, XXXRepository).

- Arrows are consistently used: solid for requests **(->>)** and dashed for responses **(-->>).**

4.6. Role in overall Architecture:

Together, the four sequence diagrams illustrate the fundamental operations of the Hbnb system, showcasing how each use case integrates into the broader architecture.

- **User Registration** establishes the foundation for authentication and personalized interactions by securely creating user accounts.

- **Place Creation** represents the supply side of the marketplace, enabling hosts to contribute rental listings.

- **Review Submission** provides social proof and feedback mechanisms, which are essential for trust and quality control.

- **Place Fetching** supports the demand side, allowing users to browse and filter available listings efficiently.

The four sequence diagrams collectively demonstrate a coherent design strategy where every feature, from data entry to retrieval, reinforces the robustness and maintainability of the Hbnb application.