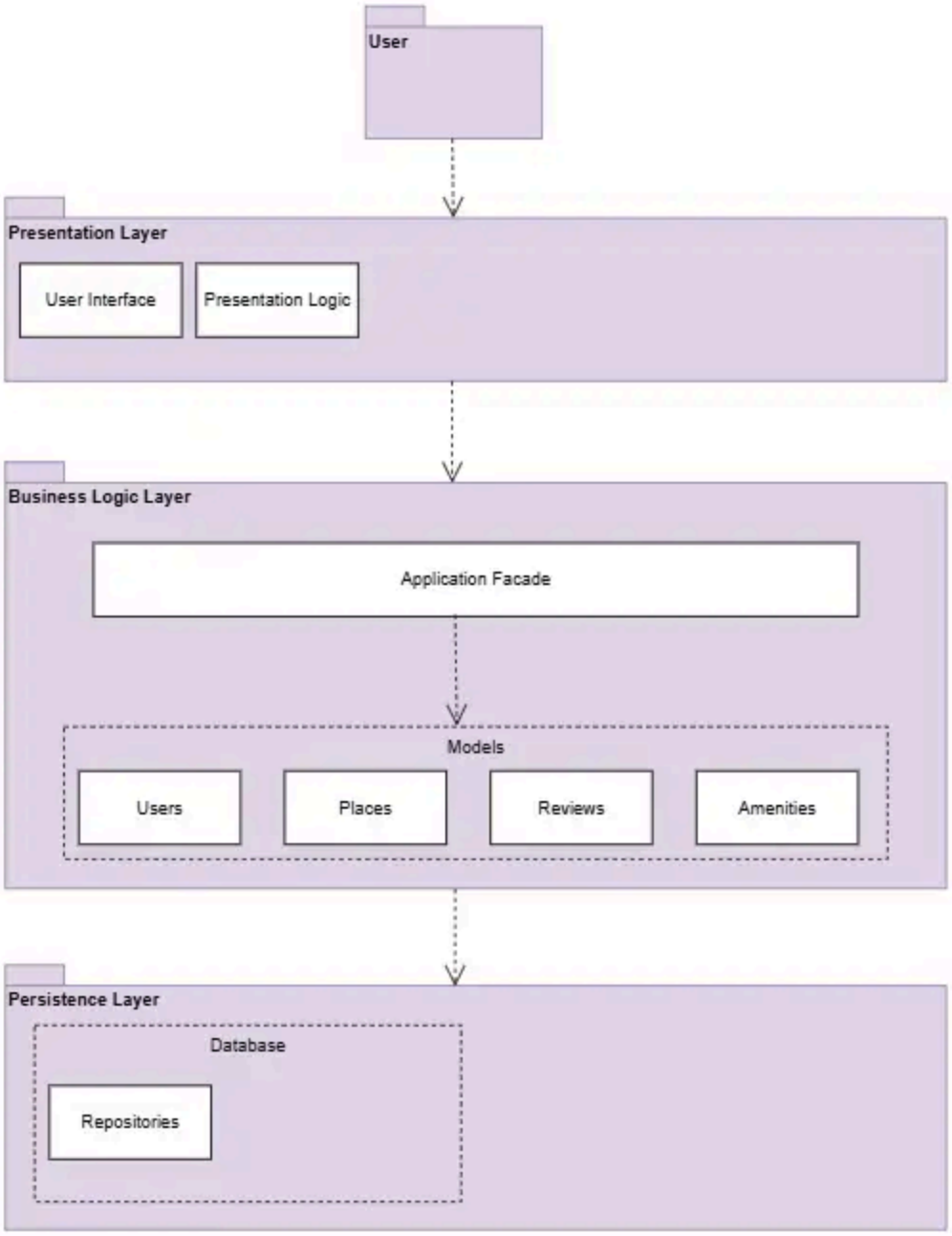
HBnB Project - UML

Authors: Toni Mathieson, Ashleigh Henna, Anna Halaapiapi, Madison Fleming

Overview

Comprehensive technical documentation that serves as the foundation for the development of the HBnB Evolution application. This part of the project includes a high-level Package Diagram, detailed Class Diagram for the Business Logic layer and Sequence Diagrams for API calls.

Task 0. High-Level Package Diagram



Layer Descriptions and Responsibilities

1. **Presentation Layer**
   This is the outermost layer and the main entry point of the application. Its primary responsibility is to handle all direct communication with the client (the user). It defines the API endpoints (e.g., `/users`, `/places`), processes incoming HTTP requests, parses data like JSON payloads, and formats the final HTTP responses.
2. **Business Logic Layer**
   This is the core or "brain" of the application. It contains all the business rules, logic, and data models (`User`, `Place`, etc.) that define how the application functions. Its responsibilities include validating data according to specific rules (e.g., a review rating must be between 1 and 5), orchestrating complex operations, and managing the relationships between different data entities.
3. **Persistence Layer**
   This layer is responsible for all data storage and retrieval. It implements the CRUD (Create, Read, Update, Delete) operations by communicating with the database.

How the Facade Pattern Facilitates Communication

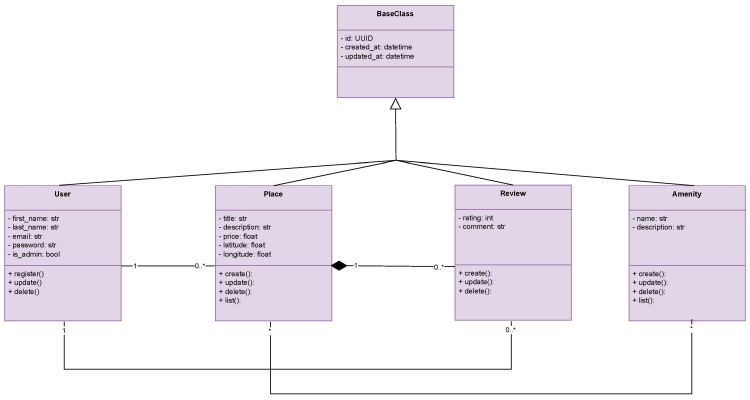
The Facade pattern is used to provide a single, simplified interface to a complex system. In this architecture, it acts as a "front door" to the Business Logic Layer.

Here's how it facilitates communication:

1. **Simplifies Interaction:** Instead of the Presentation Layer needing to know about all the different classes and methods within the Business Logic Layer, it only makes calls to the `Application Facade`. For example, to create a new place, the API endpoint makes one simple call, like `facade.create_place(data)`.
2. **Decouples Layers:** The Facade hides the internal complexity of the Business Logic Layer from the Presentation Layer. This means you can change how the business logic works internally (e.g., add more validation steps, call different services) without ever needing to change the code in the Presentation Layer.
3. **Centralises Control:** It provides a single point of entry, making the communication flow easy to manage, debug, and understand. The Facade orchestrates all the necessary steps behind the scenes, such as validating data, creating model objects, and calling the Persistence Layer to save them.

Task 1. Detailed Class Diagram for Business Logic Layer

Class Diagram for Business Logic Layer



Key

\+ denotes public attributes or methods
\- denotes private attributes or methods
△ inheritance relationship
— association relationship
◆— composition relationship

Multiplicity

1: exactly one instance
0..* zero to many instances
* many instances

Explanatory Notes

BaseClass
The BaseClass acts as the parent class for all entities in the HBnB system.
The purpose of this class is to define the attributes that are required to be shared across all subclasses.

Attributes:
• id (UUID4): uniquely identifies each object.
• created_at, updated_at: tracks the creation and update datetime for all entities.

User
The User class defines the individuals that will interact with the HBnB system.
Users can be regular users or administrators.

Attributes:
• first_name, last_name, email, password: these values will be captured to build a profile for each user.
• is_admin: a boolean flag which will indicate if the user is a regular user or an administrator.

Methods:
• register(), update(), delete(): these methods will handle user account registration, update and deletion.

Place
The Place class defines a property.

Attributes:
• title, description, price, latitude and longitude: these values will be captured to build a profile for each property.

Methods:
• create(), update(), delete(), list(): these methods will handle property creation, update, deletion and listing.

Review
The Review class defines user feedback for each Place.

Attributes:
• rating, comment: these values will be captured to build a user review of a property.

Methods:
• create(), update(), delete(): these methods will handle the creation, update and deletion of a user review.

Amenity
The Amenity class defines the features and services of each Place.

Attributes:
• name, description: these values will capture the name and description of each amenity that a property provides.

Methods:
• create(), update(), delete(), list(): these methods will handle the creation, update, deletion and listing of each amenity.

Entity Relationships

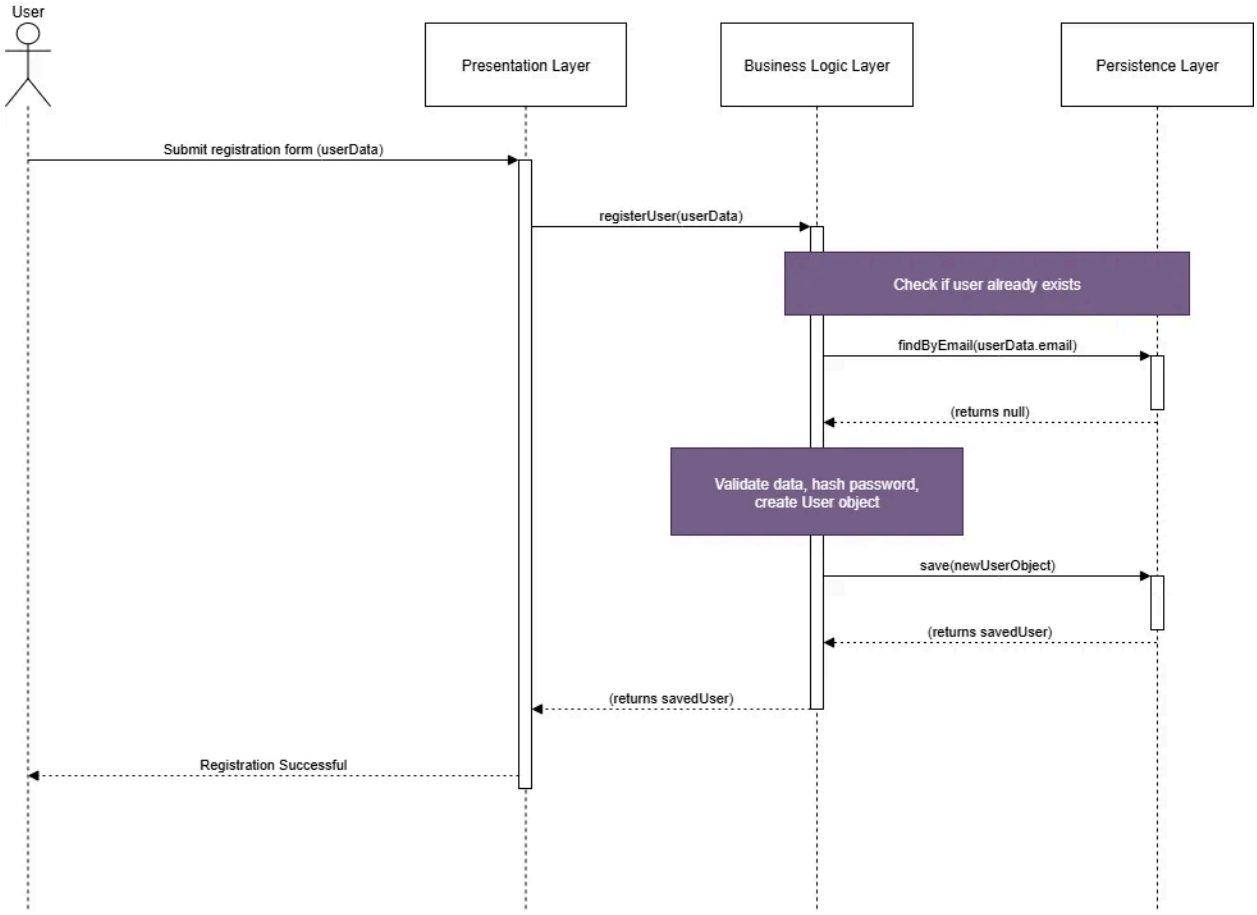
Inheritance (or Generalization):
All entities (User, Place, Review & Amenity) inherit from BaseClass. This allows for consistent identification and timestamping of each entity in the HBnB system, in order to comply with auditing requirements.

Associations (with Multiplicity):
Association is used between the following entities to indicate that instances of one entity are connected to instances of another entity.
• User (1) ↔ (0..*) Place: One user can own zero to many places. Each place belongs to one user.
• User (1) ↔ (0..*) Review: One user can write zero to many reviews. Each review is written by one user.
• Place (*) ↔ (*) Amenity: Each place can have many amenities. Each amenity can belong to many places.

Composition (with Multiplicity):
Composition is used between Place and Review to show that the part class (Review) cannot exist independently of the whole class (Place).
This is represented by a filled diamond on the side of the whole class (Place).
• Place (1) ↔ (0..*) Review: One place can have zero to many reviews. Reviews cannot exist without a place.

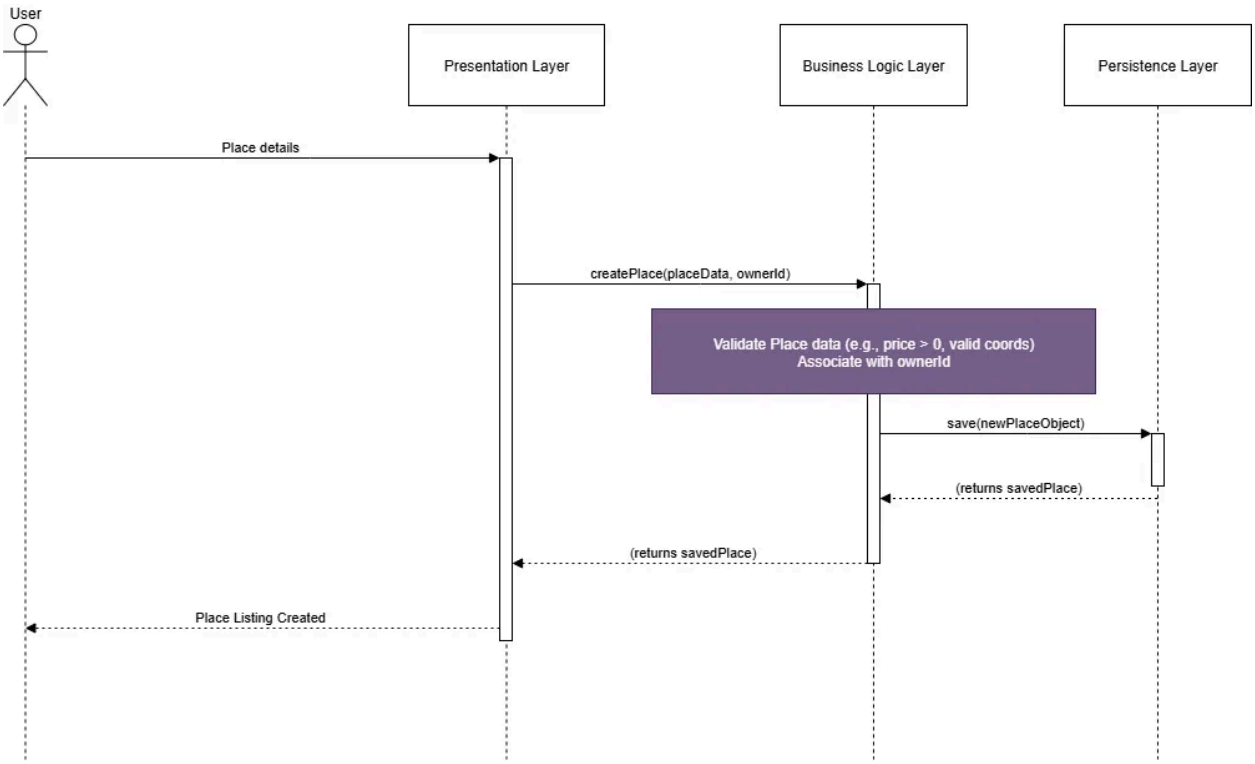
Task 2. Sequence Diagrams for API Calls

User Registration: A user signs up for a new account.



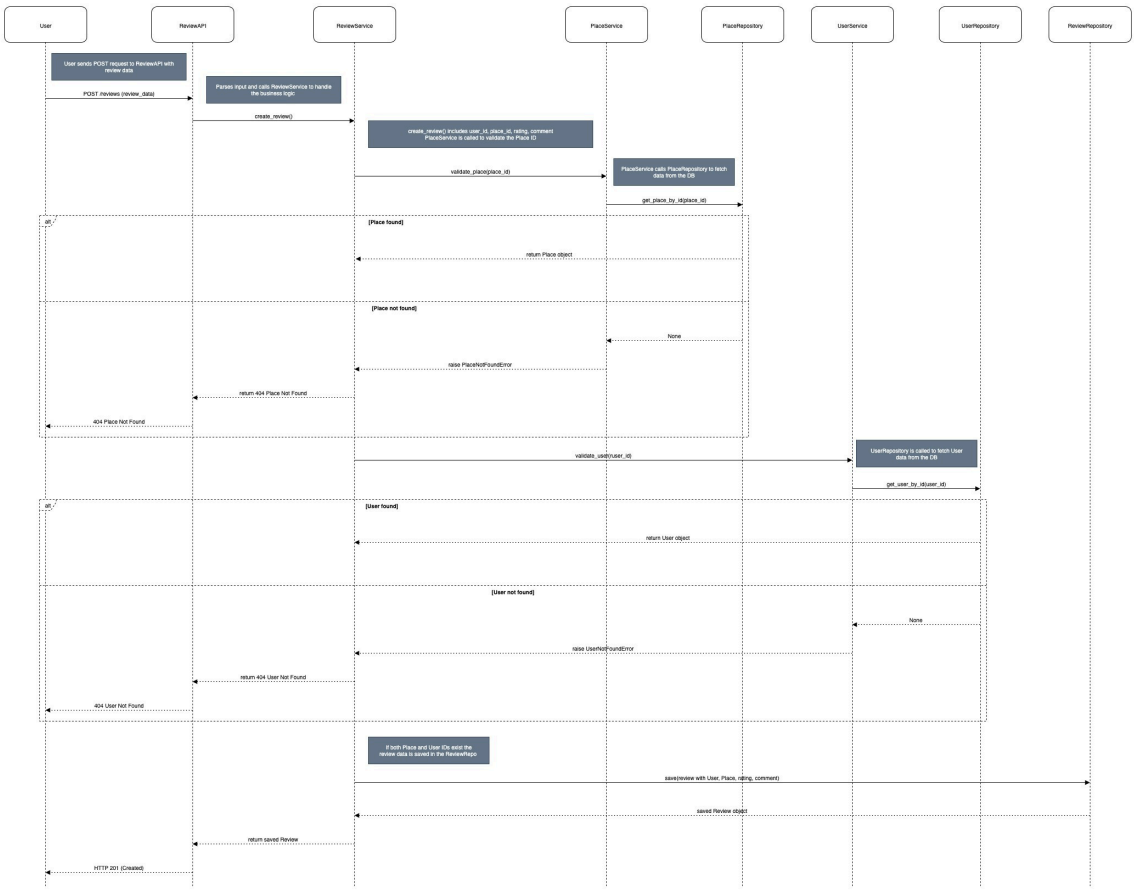
- **Description:** This diagram shows how a new user signs up. It outlines the steps required to securely validate and create a new user account.
- **Interaction Flow:**
  - **Presentation Layer:** Receives the user's data and calls the business logic layer.
  - **Business Logic Layer:** Validates that the email is unique, securely hashes the password, and instructs the persistence layer to save the user.
  - **Persistence Layer:** Executes the database commands to check for the email and save the new user record.

Place Creation: A user creates a new place listing.



- **Description:** This diagram shows how a logged-in user lists a new property.
- **Interaction Flow:**
  - **Presentation Layer:** Authenticates the user's token to get their ID, then passes the property data and `ownerId` to the business logic layer.
  - **Business Logic Layer:** Validates the property data (e.g., price, location) and creates a `Place` object associated with the `ownerId`.
  - **Persistence Layer:** Saves the new `Place` record to the database, linking it to the correct user.

Review Submission: A user submits a review for a place.



This documentation corresponds with the Review Sequence Diagram, which illustrates the process of creating a review in the HBnB application. It shows how the various components interact with each other - to validate data and save a review.

Overview

When a user submits a review via the API, the application validates the provided `user_id` and `place_id` before saving the review. This ensures that the review is associated with valid and existing entities in the database. The Sequence Diagram outlines the interaction between system components, including how validation and error handling is managed.

Components

Actor	Layer	Role
User	Client	Initiates the request via web interface or mobile app

Actor	Layer	Role
ReviewAPI	Presentation Layer	Receives and handles POST requests
ReviewService	Business Logic Layer	Validates input and calls to UserService, PlaceService, ReviewRepository to create/update reviews
PlaceService	Business Logic Layer	Verifies if a place exists and is valid for reviewing. E.g. User had booked it
UserService	Business Logic Layer	Verifies that the User ID exists and is valid for the review operation
PlaceRepository	Persistence Layer	Fetches place data from the database
UserRepository	Persistence Layer	Fetches user data from database
ReviewRepository	Persistence Layer	Saves review to database

Flow of Events

1. User sends a `POST` request to the `ReviewAPI` with review data - including `user_id` and `place_id`, rating and comment.
2. The `ReviewAPI` parses and forwards the request to the `ReviewService` to handle the business logic.
3. The `ReviewService` first calls the `PlaceService` to validate the `place_id`
4. The `PlaceService` fetches the place using the `PlaceRepository`. If the place is found, then the object is returned. Otherwise, an error is raised and a `404` response is returned to the user.
5. Similarly, the `ReviewService` calls the `UserService` to validate the `user_id`.
6. The `UserService` then queries the `UserRepository` where if the user is found, the object is returned. Otherwise, a `404` response is sent to the user.
7. Once both the `User` and `Place` are validated, the `ReviewService` creates a new review object.
8. The review is then saved to the `ReviewRepository`
9. The saved review is returned to the `ReviewAPI`, which responds with `201 Created` status to the user.

Alternatives / Error Handling

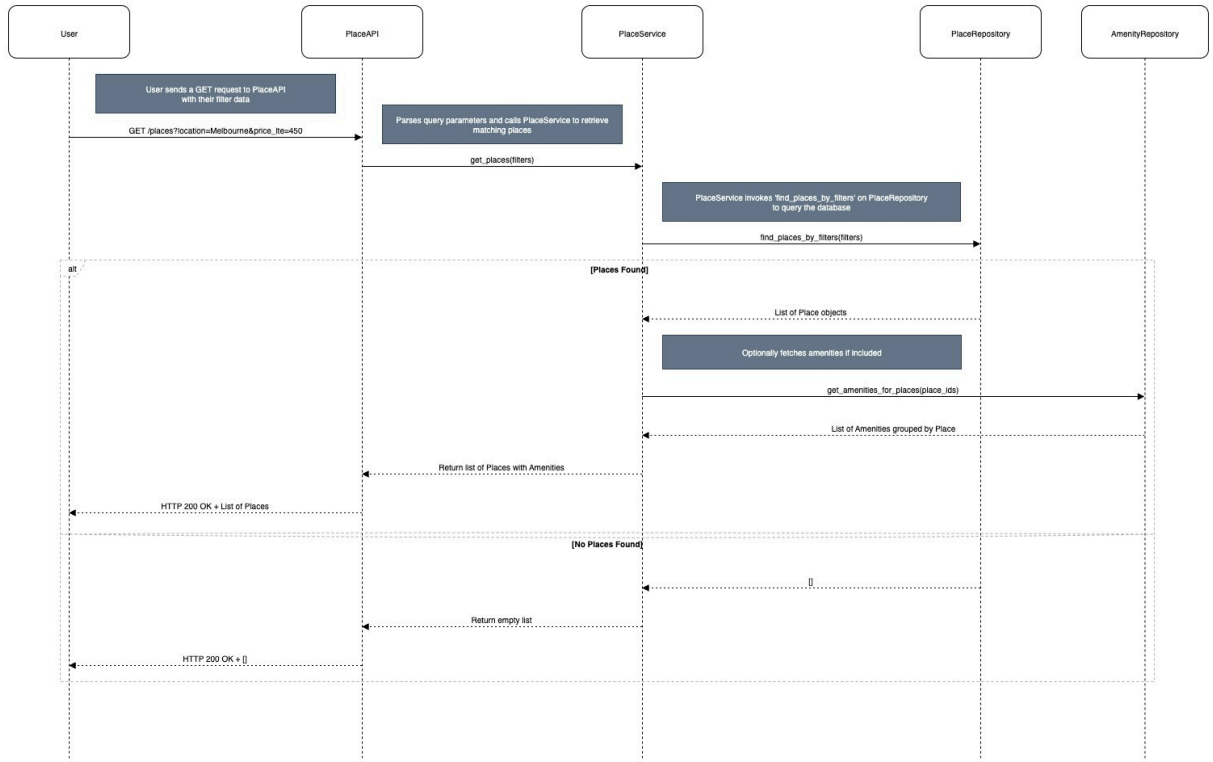
The diagram includes `alt` fragments to represent the conditional flows:

- If the Place or User does not exist, then the system exits the normal flow and returns a `404` Error
- These `alt/else` blocks model the control logic, similar to `if/else` statements in the code.

Design Considerations

- Returning the `User` and `Place` objects during validation prevents repeated database lookups and keeps the service layer efficient.
- This also ensures that the review is created with valid object references - as opposed to just raw IDs

Fetching a List of Places: A user requests a list of places based on certain criteria.



Purpose

This sequence diagram describes the steps involved when a user requests a list of places in the HBnB Evolution system. It illustrates the flow of information across the Presentation, Business Logic and Persistence layers.

Components

Actor	Layer	Role
User	Client	Initiates the request via web/browser
PlaceAPI	Presentation Layer	Receives and handles the HTTP GET request
PlaceService	Business Logic Layer	Coordinates logic and filter processing
PlaceRepository	Persistence Layer	Fetches place data from the database
AmenityRepository	Persistence Layer (optional)	(If included) Fetches amenities for places

Flow of Events

1. User sends a `GET /places?location=melbourne&price=450` request to the API
2. `PlaceAPI` parses the query parameters (converting request into Python format).
3. `PlaceAPI` calls `get_places(filters)` from `PlaceService` to retrieve matching places.
4. `PlaceService` enforces application rules (e.g. checks if location is valid, if the price is an integer and > 0 etc) and maps user filters to DB fields.
5. `PlaceService` invokes `find_places_by_filters(filters)` on `PlaceRepository` to query the database
6. `PlaceRepository` executes a database query and returns a list of `Place` objects.
7. (Optional) `PlaceService` calls `get_amenities_for_places(place_ids)` on `AmenityRepository` to enrich results
8. `PlaceService` returns a list of places (with or without amenities) to `PlaceAPI`
9. `PlaceAPI` serialises the result into JSON and returns a `200 OK` HTTP response to the user.

Alternatives

- If no places are found, the service returns an empty list:

    - API still responds with a `200 OK` status (along with an empty list [])
- If an error occurs (invalid filter) then the API returns a `400 Bad Request` response along with an error message.

Design Considerations

- This call adheres to RESTful principles using the `GET` request
- The the **Facade Pattern** where `PlaceService` acts as the facade between the API and data.
- The system follows a clean separation between presentation, business logic and data layers where:
    - API handles request/response
    - Service handles logic
    - Repository handles data access