Holberton Ponce

HBnB Evolution - UML

Willie J Touset Castellar - Cohort 25

Github - https://github.com/neonbloo25

HBnB Evolution:

Introduction:

This project is an academic assignment aimed at educating the student on the functionality of a platform similar to AirBnB. The objective is to gain a comprehensive understanding of how a booking and review system operates, including the key features and components essential to such a platform. The project focuses on exploring the underlying business logic, user interactions, and system architecture necessary for building a fully functional service that allows users to discover, book, and review properties.

Overview:

The system is designed as a web platform that allows users to discover, book, and review properties, much like AirBnB. At a high level, the system architecture consists of three main layers: the front-end, back-end, and database.

- **Front-End**: The user interface provides a seamless experience for browsing properties, viewing details, submitting reviews, and managing bookings. It also allows users to register, create profiles, and interact with the platform's features through intuitive design and navigation.

- **Back-End**: The business logic layer handles user authentication, property listings, review management, and booking processes. It processes user requests, ensures secure transactions, and interacts with the database to manage and retrieve data. This layer ensures the smooth functioning of all critical operations, including place creation, review submission, and data validation.

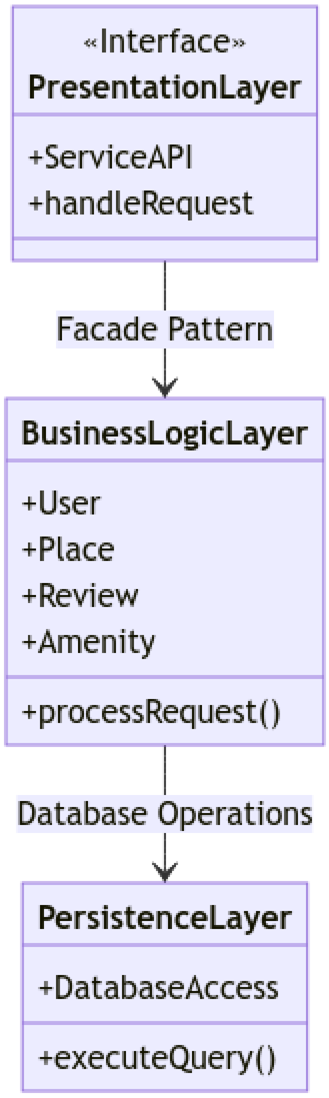
- **Database**: The database stores all relevant data, including user information, property listings, reviews, and booking records. It supports efficient querying and management of data to ensure scalability and responsiveness.

Core functionality includes:

- **User Registration and Profile Management**: Users can sign up, log in, and manage their personal information.
- **Property Listing and Search**: Property owners can list their properties, and users can search for available listings based on location, amenities, and price.
- **Review Submission**: Users can submit reviews for properties they have stayed in, helping others make informed decisions.
- **Amenity System**: Users can manage and list amenities available to users, and list them based on Place.

This architecture ensures that the system is scalable, maintainable, and capable of handling the core functionalities required for a platform like AirBnB.

Task 0 - High Level Package Diagram



Objective

Provide a visual representation of the system's architecture, illustrating how the front-end, back-end, and database interact, helping to clarify the relationships between components and the flow of data and processes.

Overview

Illustrate the system's key components: the front-end, back-end, and database, highlighting their interactions and data flow to support user registration, property listings, bookings, and reviews.

```
%%Diagram type

classDiagram

%%Nodes and Contents

class PresentationLayer {
   <<Interface>>
   +ServiceAPI
   +handleRequest
}
class BusinessLogicLayer {
   +User
   +Place
   +Review
   +Amenity
   +processRequest()
}
class PersistenceLayer {
   +DatabaseAccess
   +executeQuery()
}

%%Element Display order and line labels

PresentationLayer --> BusinessLogicLayer : Facade Pattern
BusinessLogicLayer --> PersistenceLayer : Database Operations
```

Diagram Explanation

The diagram represents a simplified view of the system's architecture, divided into three main layers: Presentation Layer, Business Logic Layer, and Persistence Layer.

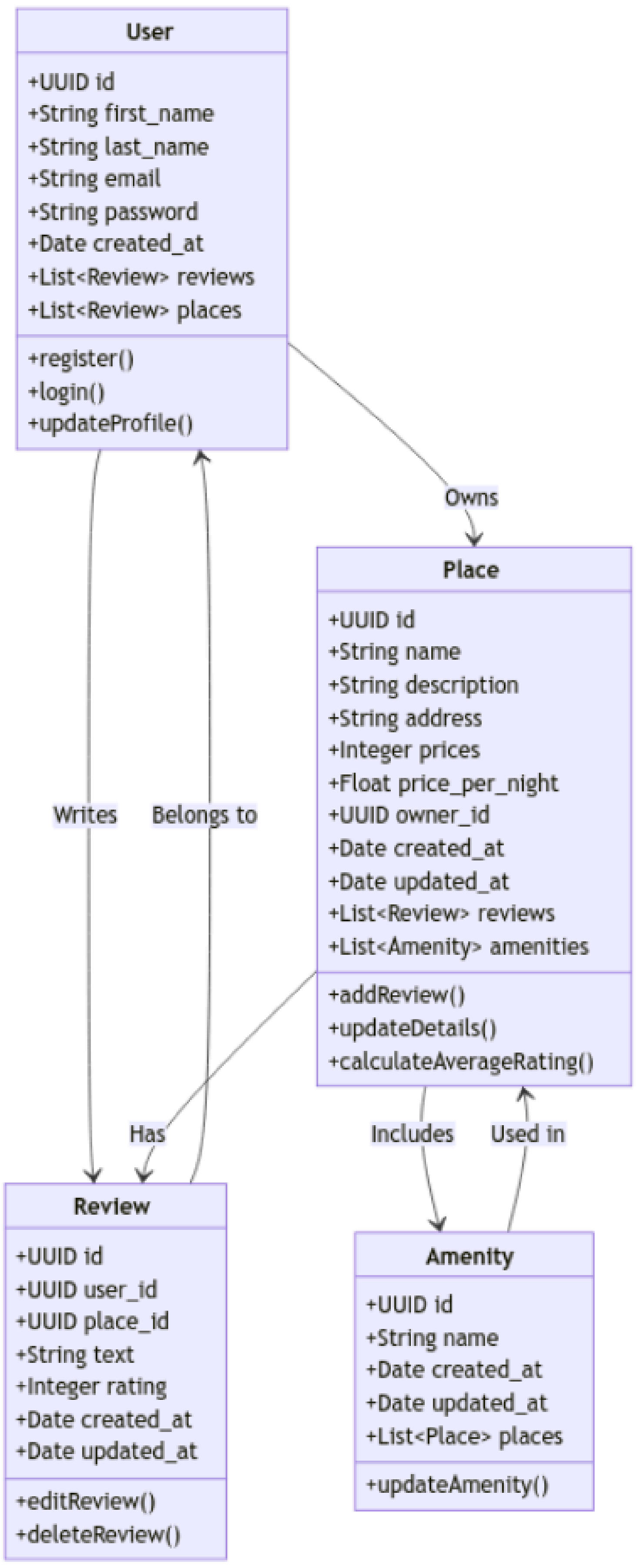
1. Presentation Layer: This layer acts as the interface between the user and the system. It exposes a ServiceAPI to interact with the system and has the function handleRequest() to process incoming requests from users. Its purpose is to receive user input and relay it to the appropriate business logic.

2. Business Logic Layer: The core of the system, this layer handles the application's primary operations. It includes entities such as User, Place, Review, and Amenity, which represent the core business objects in the platform. The processRequest() function processes incoming requests, applying the necessary business logic to interact with users, places, reviews, and amenities.

3. Persistence Layer: This layer is responsible for interacting with the database. It contains DatabaseAccess, which handles the communication with the database, and the executeQuery() function, which performs database operations like storing and retrieving data.

Interactions:

The Presentation Layer communicates with the Business Logic Layer using the Facade Pattern, simplifying the interface between the user and the core business logic. This pattern allows the presentation layer to interact with the complex logic in the business layer in a streamlined manner.
The Business Logic Layer then interacts with the Persistence Layer to perform necessary database operations, such as saving user data, managing property listings, and processing reviews.



Task 1 - Detailed Class Diagram for Business Logic Layer

Objective:

To represent the core business logic of the system. It illustrates how key entities such as **User, Place, Review,** and **Amenity** interact and how the system processes user requests. The diagram shows the structure and relationships between these entities, highlighting their roles in managing user interactions, property listings, and reviews within the platform.

Overview:

The diagram highlights the core classes within the Business Logic Layer: `User`, `Place`, `Review`, and `Amenity`. The `User` class represents platform users who can create profiles, book places, and submit reviews. The `Place` class stores information about properties listed on the platform. The `Review` class captures user feedback for places, while the `Amenity` class defines the features and facilities available at each place. These classes work together to manage the platform’s primary functionality, including user registration, place listings, and review management.

```
%%Diagram type

classDiagram

%%Nodes and Contents

class User {
   +UUID id0
.12
   +String first_name
   +String last_name
   +String email
   +String password
   +Date created_at
   +List~Review~ reviews
   +List~Review~ places
   +register()
   +login()
   +updateProfile()
}

class Place {
   +UUID id
   +String name
   +String description
   +String address
   +Integer prices
   +Float price_per_night
   +UUID owner_id
   +Date created_at
   +Date updated_at
   +List~Review~ reviews
   +List~Amenity~ amenities
   +addReview()
   +updateDetails()
   +calculateAverageRating()
}

class Review {
   +UUID id
   +UUID user_id
   +UUID place_id
   +String text
   +Integer rating
   +Date created_at
   +Date updated_at
   +editReview()
   +deleteReview()
}

class Amenity {
   +UUID id
   +String name
   +Date created_at
   +Date updated_at
   +List~Place~ places
   +updateAmenity()
}

%%Element Display order and line labels

User --> Place : Owns
User --> Review : Writes
Place --> Review : Has
Review --> User : Belongs to
Place --> Amenity : Includes
Amenity --> Place : Used in
```

Explanation of Each Entity:

1. User Class

The `User` class represents the platform's users. It contains attributes like `id`, `first_name`, `last_name`, `email`, and `password`. Key methods include `register()` for creating accounts, `login()` for authentication, and `updateProfile()` for modifying user details. Users can have multiple reviews and places associated with them, reflecting their activity on the platform.

2. Place Class

The `Place` class represents properties listed on the platform. It contains attributes such as `id`, `name`, `description`, `address`, `price_per_night`, and `owner_id`. Methods like `addReview()` allow users to submit reviews, while `updateDetails()` and `calculateAverageRating()` manage place information and ratings. Each place has associated reviews and amenities, linking it to user feedback and offered features.

3. Review Class

The `Review` class stores feedback submitted by users for places. It includes `id`, `user_id`, `place_id`, `text`, `rating`, and timestamps. The methods `editReview()` and `deleteReview()` allow users to modify or remove their reviews. Reviews are tied to both users and places, providing valuable insights on the quality of properties.

4. Amenity Class

The `Amenity` class represents features or services available at places, such as Wi-Fi or parking. It includes attributes like `id`, `name`, and timestamps. Amenities are linked to places through a list, with the `updateAmenity()` method enabling changes to amenity details.

Explanation of Relationships

1. User-Review

A `User` can write multiple `Review` instances, which reflect their experiences with places. Each review is tied to the user who created it, establishing a direct relationship between users and their feedback.

2. User-Place

A `User` can own or visit multiple `Place` instances. The relationship shows how users interact with places, either as owners (via `owner_id`) or as visitors who might leave reviews.

3. Place-Review

Each `Place` can have multiple `Review` instances. Reviews are associated with places, providing feedback on the property's quality, amenities, and overall experience.

4. Place-Amenity

A `Place` can include various `Amenity` instances, such as Wi-Fi, parking, or a pool. Amenities are linked to places, enhancing the property's features and attracting potential visitors.

5. Review-User & Place

Each `Review` is connected to both a `User` and a `Place`. A review belongs to a user (who wrote it) and is linked to the place it pertains to, providing valuable insights on user experiences at that location.

Task 2 - Sequence Diagram for API Calls

Objective

These illustrate how API calls are processed within the system. It shows the interactions between the client, API, business logic, and persistence layers, highlighting the flow of data and operations as the system responds to user requests.

Overview

This provides a high-level view of how the system handles API calls, demonstrating the sequence of interactions between various components. It illustrates how a user's request is received by the API, processed by the business logic, and stored or retrieved from the database, ensuring that the system handles requests in an orderly and efficient manner.

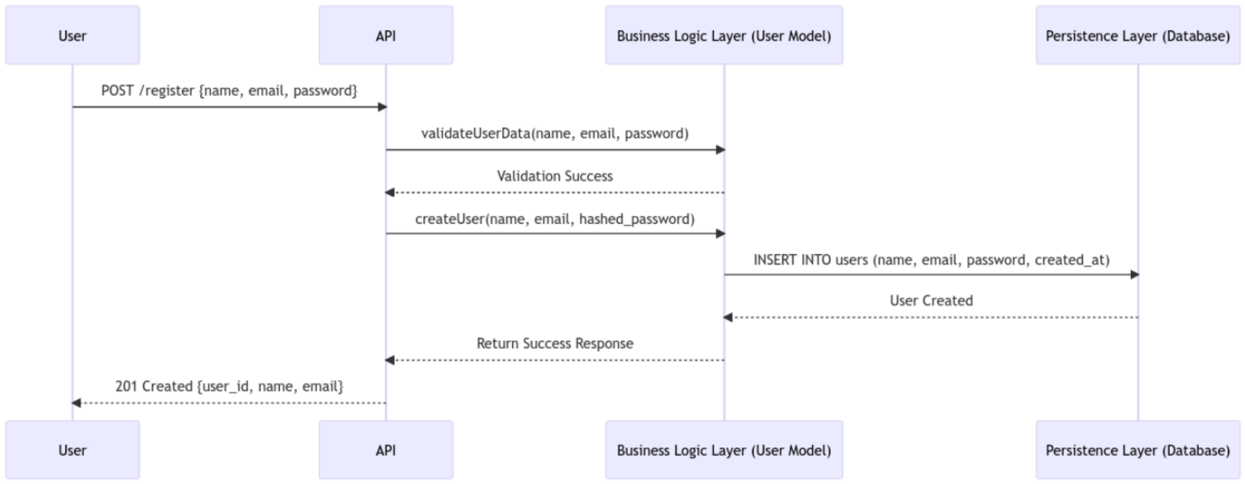
1. User Registration

Overview

The user registration process allows new users to create an account by providing necessary details such as their name, email, and password. The system validates the provided data, hashes the password for security, and stores the new user information in the database.

Description

The process begins when the user sends a POST request to the API with their registration details (`name`, `email`, and `password`). The API then forwards this data to the Business Logic Layer to validate the input. Upon successful validation, the API triggers the creation of a new user, which is handled by the Business Logic Layer. This layer interacts with the database to insert the user information into the `users` table. Once the user is created, the Business Logic Layer sends a success response back to the API, which in turn returns a `201 Created` status to the user along with their `user_id`, `name`, and `email`. This completes the user registration process.


User
API
Business Logic Layer (User Model)
Persistence Layer (Database)
POST /register {name, email, password}
validateUserData(name, email, password)
Validation Success
createUser(name, email, hashed_password)
INSERT INTO users (name, email, password, created_at)
User Created
Return Success Response
201 Created {user_id, name, email}
User
API
Business Logic Layer (User Model)
Persistence Layer (Database)

```
%%Diagram type

sequenceDiagram

   %%Nodes Layers

   participant User
   participant API
   participant BusinessLogic as Business Logic Layer (User Model)
   participant Database as Persistence Layer (Database)

   %%Layer Relations and Line Labels

   User->>API: POST /register {name, email, password}
   API->>BusinessLogic: validateUserData(name, email, password)
   BusinessLogic-->>API: Validation Success
   API->>BusinessLogic: createUser(name, email, hashed_password)
   BusinessLogic->>Database: INSERT INTO users (name, email, password, created_at)
   Database-->>BusinessLogic: User Created
   BusinessLogic-->>API: Return Success Response
   API-->>User: 201 Created {user_id, name, email}
```

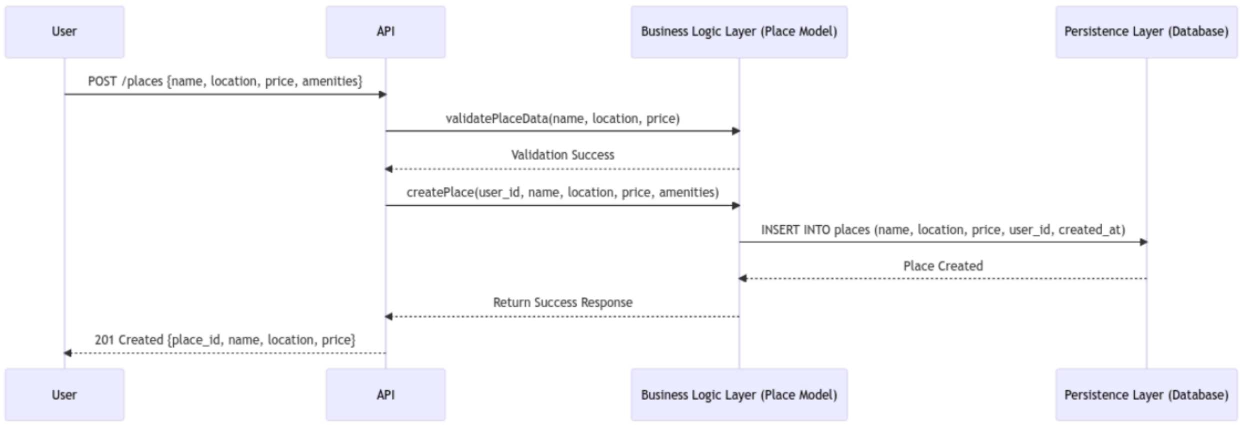
2. Place Creation

Overview

The place creation process allows a user to add a new property to the platform by providing necessary details like name, location, price, and amenities.

Description

The process begins when the user sends a POST request to the API with the place details. The API forwards this data to the Business Logic Layer for validation. After successful validation, the API sends a request to create the place, which the Business Logic Layer processes by inserting the details into the database. Once the place is successfully created, the Business Logic Layer returns a success response to the API, which in turn responds to the user with a `201 Created` status, including the place's ID, name, location, and price.


User
API
Business Logic Layer (Place Model)
Persistence Layer (Database)
POST /places {name, location, price, amenities}
validatePlaceData(name, location, price)
Validation Success
createPlace(user_id, name, location, price, amenities)
INSERT INTO places (name, location, price, user_id, created_at)
Place Created
Return Success Response
201 Created {place_id, name, location, price}
User
API
Business Logic Layer (Place Model)
Persistence Layer (Database)

```
%%Diagram Type

sequenceDiagram

   %%Node Layers

   participant User
   participant API
   participant BusinessLogic as Business Logic Layer (Place Model)
   participant Database as Persistence Layer (Database)

   %%Layer Relations and Link labels

   User->>API: POST /places {name, location, price, amenities}
   API->>BusinessLogic: validatePlaceData(name, location, price)
   BusinessLogic-->>API: Validation Success
   API->>BusinessLogic: createPlace(user_id, name, location, price, amenities)
   BusinessLogic->>Database: INSERT INTO places (name, location, price, user_id, created_at)
   Database-->>BusinessLogic: Place Created
   BusinessLogic-->>API: Return Success Response
   API-->>User: 201 Created {place_id, name, location, price}
```

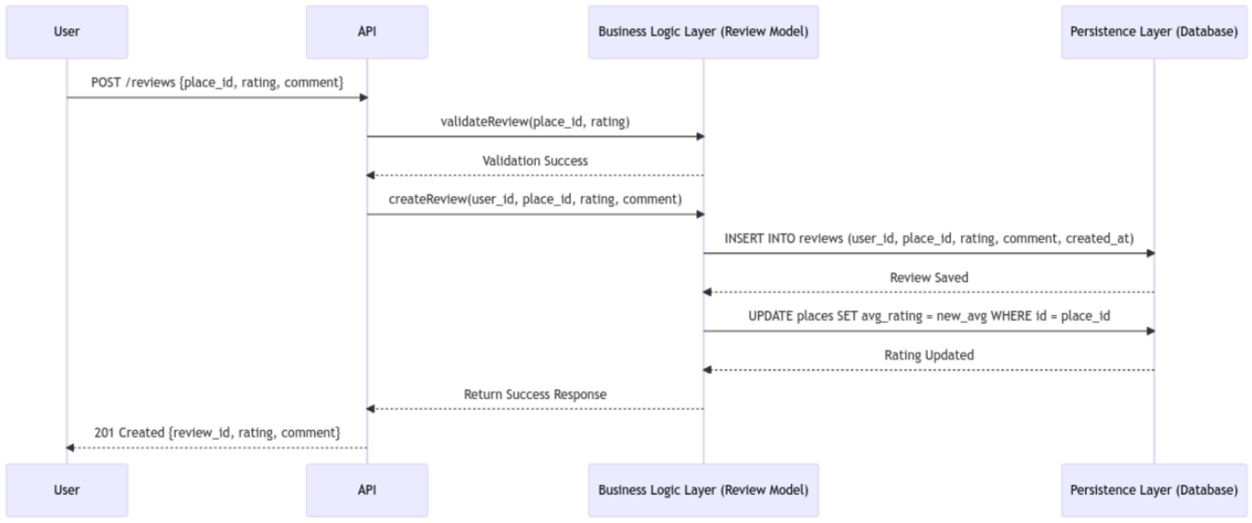
3. Review Submission

Overview

The review submission process allows users to provide feedback on places by submitting a rating and comment.

Description

The process begins when the user sends a POST request to the API with the review details (place_id, rating, and comment). The API forwards the data to the Business Logic Layer for validation. Once validated, the API sends a request to create the review, which is then stored in the database. After the review is saved, the Business Logic Layer updates the average rating for the place. A success response is returned to the user with the review details, including the review ID, rating, and comment.


User
API
Business Logic Layer (Review Model)
Persistence Layer (Database)
POST /reviews {place_id, rating, comment}
validateReview(place_id, rating)
Validation Success
createReview(user_id, place_id, rating, comment)
INSERT INTO reviews (user_id, place_id, rating, comment, created_at)
Review Saved
UPDATE places SET avg_rating = new_avg WHERE id = place_id
Rating Updated
Return Success Response
201 Created {review_id, rating, comment}

```
%%Diagram Type

sequenceDiagram

   %%Node Layers
   participant User
   participant API
   participant BusinessLogic as Business Logic Layer (Review Model)
   participant Database as Persistence Layer (Database)

   %%Layer Dynamics
   User->>API: POST /reviews {place_id, rating, comment}
   API->>BusinessLogic: validateReview(place_id, rating)
   BusinessLogic-->>API: Validation Success
   API->>BusinessLogic: createReview(user_id, place_id, rating, comment)
   BusinessLogic->>Database: INSERT INTO reviews (user_id, place_id, rating, comment,
created_at)
   Database-->>BusinessLogic: Review Saved
   BusinessLogic->>Database: UPDATE places SET avg_rating = new_avg WHERE id = place_id
   Database-->>BusinessLogic: Rating Updated
   BusinessLogic-->>API: Return Success Response
   API-->>User: 201 Created {review_id, rating, comment}
```

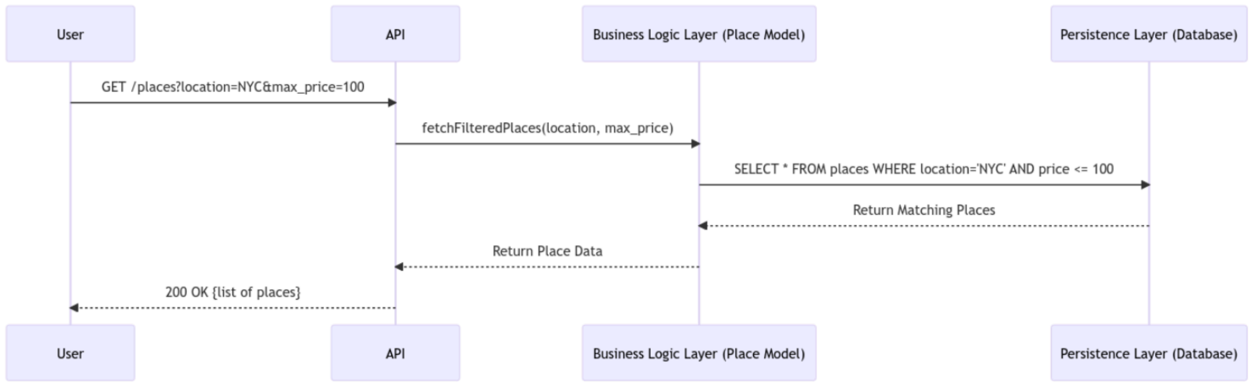
4. Fetching & Listing Places

Overview

The process of fetching and listing places allows users to search for properties based on location and price criteria.

Description

The process starts when the user sends a GET request to the API with the desired location and maximum price. The API forwards these parameters to the Business Logic Layer, which requests matching places from the database. The database returns the filtered results based on the specified criteria. The Business Logic Layer then sends the place data back to the API, which responds to the user with a list of places that meet the search requirements.


User
API
Business Logic Layer (Place Model)
Persistence Layer (Database)
GET /places?location=NYC&max_price=100
fetchFilteredPlaces(location, max_price)
SELECT * FROM places WHERE location='NYC' AND price <= 100
Return Matching Places
Return Place Data
200 OK {list of places}
User
API
Business Logic Layer (Place Model)
Persistence Layer (Database)

```
%%Diagram Type

sequenceDiagram

   %%Node Layers

   participant User
   participant API
```

```
    participant BusinessLogic as Business Logic Layer (Place Model)
    participant Database as Persistence Layer (Database)

    %%Node Relations

    User->>API: GET /places?location=NYC&max_price=100
    API->>BusinessLogic: fetchFilteredPlaces(location, max_price)
    BusinessLogic->>Database: SELECT * FROM places WHERE location='NYC' AND price <= 100
    Database-->>BusinessLogic: Return Matching Places
    BusinessLogic-->>API: Return Place Data
    API-->>User: 200 OK {list of places}
```

Explanations & Key Interactions

1. User Registration

During user registration, the user submits their details through an API request. The API forwards this data to the business logic layer for validation. After validation, the business logic layer creates the user in the database and returns a success response to the API, which then sends a confirmation to the user.

2. Place Creation

In the place creation process, the user submits place details via an API request. The API sends this data to the business logic layer for validation. After successful validation, the business logic layer creates the place in the database and returns a success response to the API, which confirms the place creation to the user.

3. Review Submission

For review submission, the user sends a review via an API request. The API validates the review data with the business logic layer, and once validated, the business logic layer stores the review and updates the place's rating. The API then sends a confirmation response back to the user.

4. Fetching Places

When fetching places, the user sends a request with specific search criteria. The API forwards these parameters to the business logic layer, which queries the database for matching places. The results are then sent back to the API, which responds to the user with the relevant list of places.

Conclusion

This project provides a comprehensive exploration of a simplified version of an AirBnB-like platform, focusing on core functionalities like user registration, place creation, review submission, and place searching. Developing the system maps out key interactions between users, places, and reviews, clarifying the flow of data and business logic.

One challenge encountered is ensuring smooth integration across different system components, but this is overcome by focusing on clear, modular interactions.

Looking forward, enhancing the platform by adding advanced features, improving performance, and refining user experience is recommended. Future steps involve optimizing system scalability and introducing new functionalities, such as better filtering options or payment integrations.