HBnB

Introduction

This document presents the UML diagrams for the **HBnB** project, a clone of Airbnb designed to facilitate property listings and bookings. The diagrams illustrate the system's architecture, interactions, and data flow, providing a comprehensive overview of its structure and functionalities.

The included diagrams cover:

- **High-Level Package Diagram**: Showcasing the separation between the presentation layer, business logic layer, and data persistence layer.

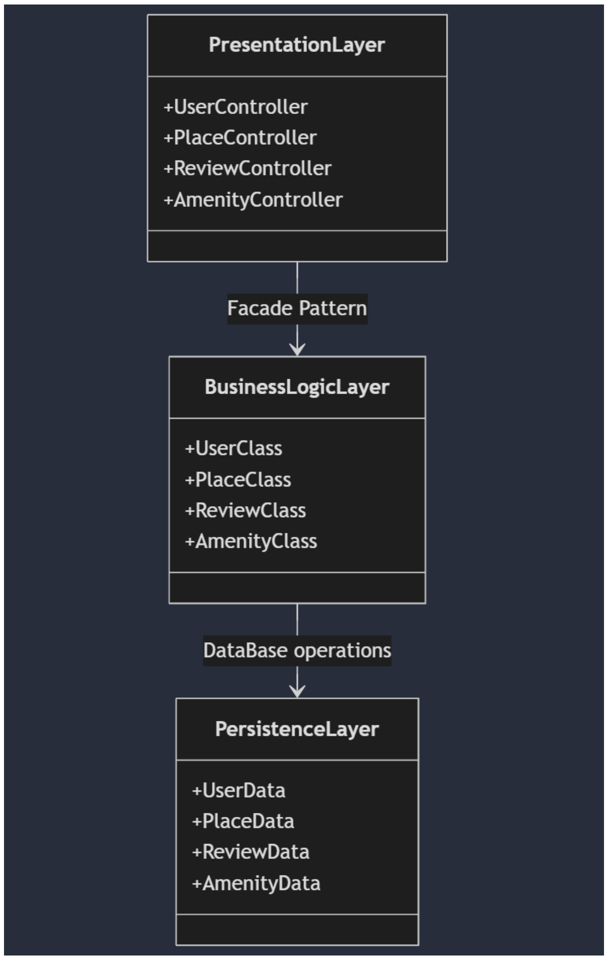
- **Class Diagram**: Defining the core components of the system, including users, places, reviews, and amenities, along with their relationships and attributes.

- **Sequence Diagrams**: Representing key API interactions, such as user registration, place creation, fetching places, and submitting reviews.

Mathieu ZUCALLI
Sebastien GEORGESCU

<u>High-Level Package Diagram:</u>

- **Presentation Layer:** This layer serves as the entry point for user interactions, handling API requests and responses. It includes controllers such as `UserController`, `PlaceController`, `ReviewController`, and `AmenityController`, which manage different aspects of the system. The primary role of this layer is to receive user inputs, forward them to the business logic layer, and return appropriate responses. It ensures that the system remains accessible and user-friendly.



- **Business Logic Layer:** This layer is responsible for processing requests and enforcing business rules. It contains core classes (`UserClass`, `PlaceClass`, `ReviewClass`, and `AmenityClass`) that validate data, apply logic, and ensure that operations comply with system requirements. This layer acts as an intermediary between the presentation and persistence layers, ensuring that only valid data reaches the database while maintaining the integrity of the application's functionalities.

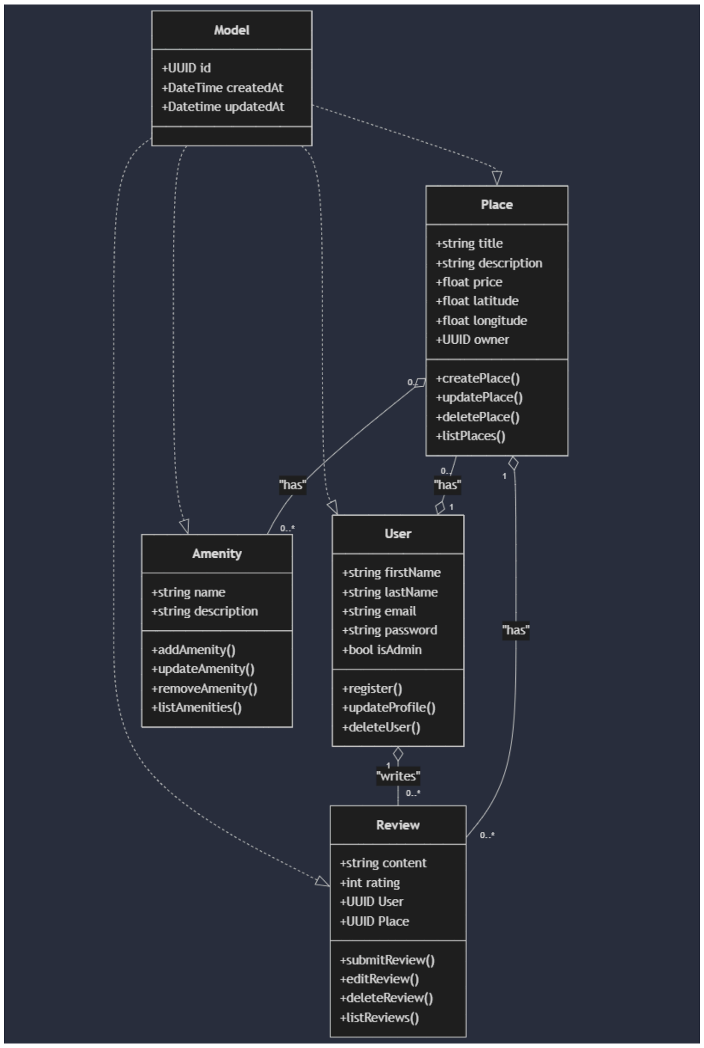
- **Persistence Layer**: This layer is in charge of data storage and retrieval. It consists of database management classes (`UserData`, `PlaceData`, `ReviewData`, and `AmenityData`) that handle interactions with the database. The persistence layer ensures efficient data access, maintains consistency, and separates database operations from the rest of the system, improving maintainability and scalability.

  <u>*This three-layer architecture enhances modularity, making the system more maintainable, scalable, and easier to extend in the future.*</u>

Mathieu ZUCALLI
Sebastien GEORGESCU

Class diagram:

- **Model:** This is a base class that provides common attributes for all entities in the system. It includes a unique identifier (`id`), a creation timestamp (`createdAt`), and an update timestamp (`updatedAt`). This ensures that every object in the system maintains a consistent structure for tracking changes over time.



- **User:** This class represents a system user. It stores personal information such as `firstName`, `lastName`, `email`, and `password`. The `isAdmin` attribute determines whether a user has administrative privileges. Users can perform actions such as registering (`register()`), updating their profile (`updateProfile()`), and deleting their account (`deleteUser()`).

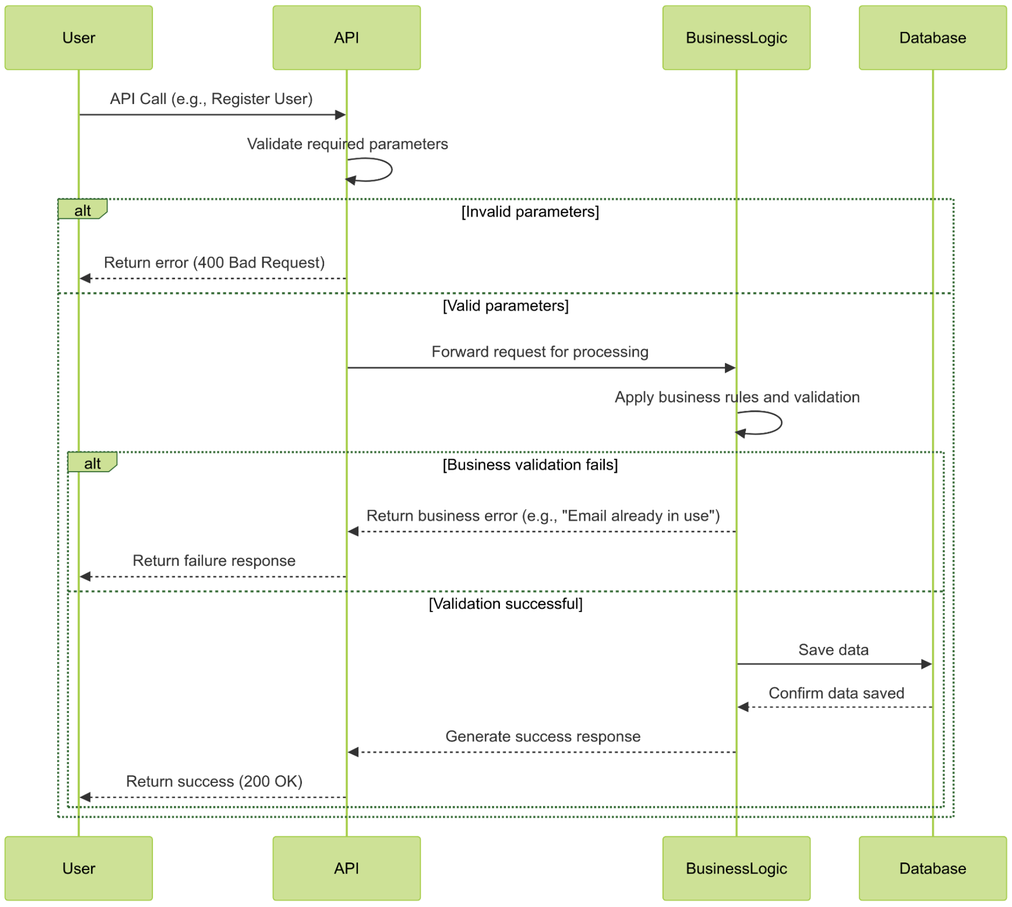
- **Place:** This class defines properties that users can list on the platform. Each place has attributes such as `title`, `description`, `price`, `latitude`, and `longitude`, along with an `owner` identifier linking it to a user. Methods include `createPlace()`, `updatePlace()`, `deletePlace()`, and `listPlaces()`, allowing users to manage their listings.

- **Review:** This class represents user feedback on places. Each review has `content`, a `rating`, and references to both the `User` who wrote it and the `Place` being reviewed. Users can submit (`submitReview()`), edit (`editReview()`), delete (`deleteReview()`), and retrieve (`listReviews()`) reviews.

- **Amenity:** This class represents additional features available in a place, such as Wi-Fi, parking, or a swimming pool. It includes `name` and `description` attributes, along with methods to add (`addAmenity()`), update (`updateAmenity()`), remove (`removeAmenity()`), and list (`listAmenities()`) amenities.

Sequence Diagrams:

User Registration Sequence Diagram:


User
API
BusinessLogic
Database
API Call (e.g., Register User)
Validate required parameters
alt
[Invalid parameters]
Return error (400 Bad Request)
[Valid parameters]
Forward request for processing
Apply business rules and validation
alt
[Business validation fails]
Return business error (e.g., "Email already in use")
Return failure response
[Validation successful]
Save data
Confirm data saved
Generate success response
Return success (200 OK)
User
API
BusinessLogic
Database

This sequence diagram illustrates the user registration process in the Hbnb system, detailing the interactions between the User, API, Business Logic, and Database. The process follows these steps:

1. **User initiates registration:**
   - The User sends an API request to register, including necessary details such as email, password, and personal information.

2. **API validation:**
   - The API first checks if all required parameters are provided.
   - If any required parameter is missing or invalid, the API returns a 400 Bad Request error to the user.

Mathieu ZUCALLI
Sebastien GEORGESCU

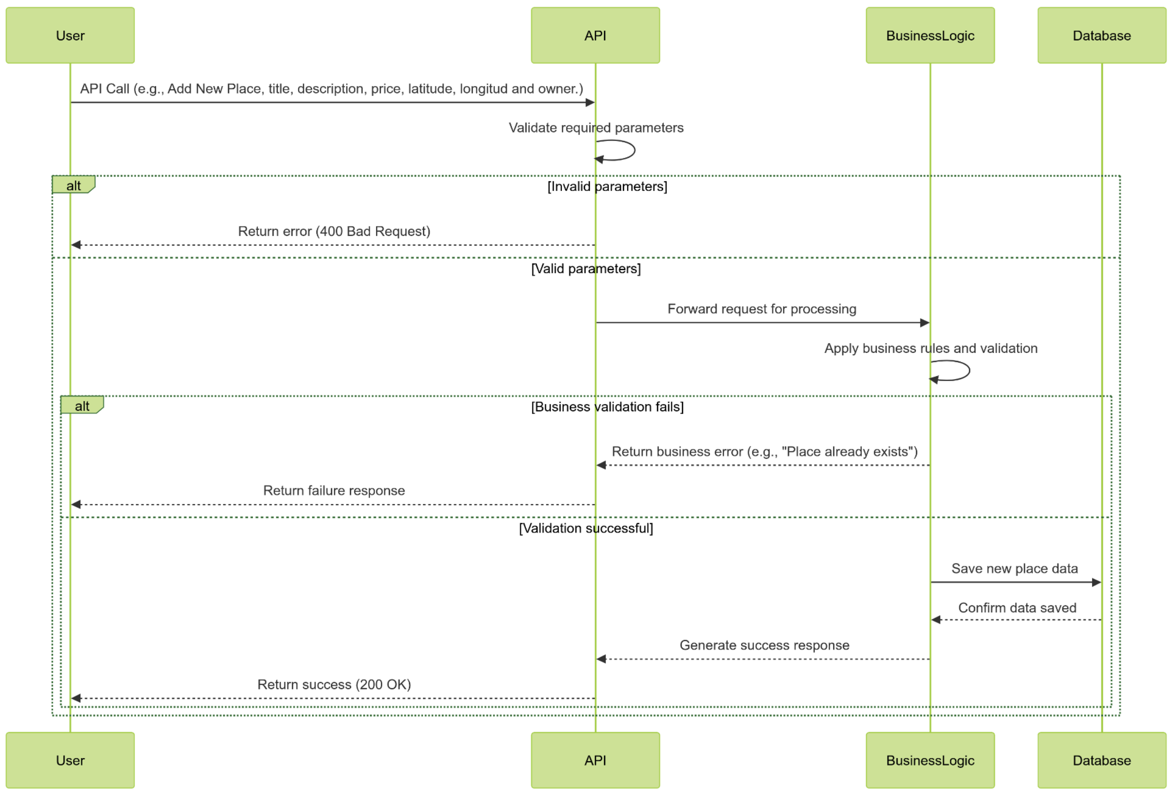
3. **Forwarding to Business Logic:**
    - If the input is valid, the API forwards the request to the Business Logic layer for further processing.

4. **Business logic validation:**
    - The Business Logic applies rules and validation (e.g., checking if the email is already in use).
    - If validation fails, an error (e.g., "Email already in use") is returned to the API, which then informs the user.

5. **Saving user data:**
    - If validation succeeds, the Business Logic sends the data to the Database for storage.
    - The Database confirms that the data has been saved successfully.

6. **Returning a success response:**
    - The Business Logic generates a success response.
    - The API sends a 200 OK response back to the user, confirming successful registration.

Place Creation Sequence Diagram:


User
API
BusinessLogic
Database
API Call (e.g., Add New Place, title, description, price, latitude, longitud and owner.)
Validate required parameters
alt
[Invalid parameters]
Return error (400 Bad Request)
[Valid parameters]
Forward request for processing
Apply business rules and validation
alt
[Business validation fails]
Return business error (e.g., "Place already exists")
Return failure response
[Validation successful]
Save new place data
Confirm data saved
Generate success response
Return success (200 OK)
User
API
BusinessLogic
Database

This sequence diagram illustrates the process of adding a new place in the Hbnb system. It outlines the interactions between the User, API, Business Logic, and Database during the place creation workflow.

1. **User initiates place creation:**
   - The User sends an API request to add a new place, providing details such as `title`, `description`, `price`, `latitude`, `longitude`, and `owner`.

2. **API validation:**
   - The API verifies whether all required parameters are present.
   - If any parameter is missing or invalid, the API returns a 400 Bad Request error to the user.

Mathieu ZUCALLI
Sebastien GEORGESCU

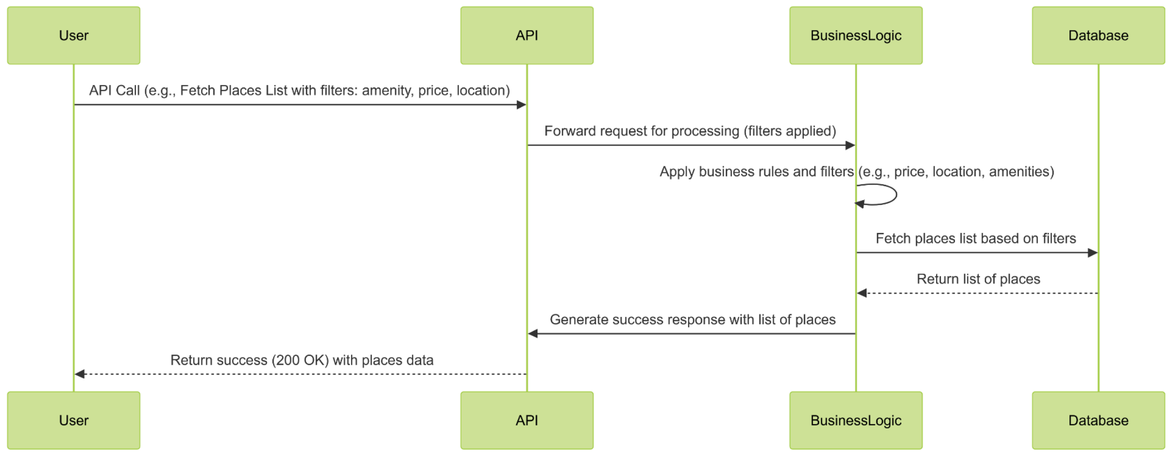
3. **Forwarding to Business Logic:**
    - If validation passes, the API forwards the request to the Business Logic layer for processing.

4. **Business logic validation:**
    - The Business Logic applies necessary rules and constraints, such as checking for duplicate places or ensuring valid pricing.
    - If a business rule is violated (e.g., "Place already exists"), an error response is sent back to the API, which then informs the user.

5. **Saving place data:**
    - If all validations succeed, the Business Logic saves the new place details in the Database.
    - The Database confirms that the data has been successfully stored.

6. **Returning a success response:**
    - The Business Logic generates a success message.
    - The API sends a 200 OK response back to the user, confirming that the place has been successfully added.

Fetching Places Sequence Diagram:


User
API
BusinessLogic
Database
API Call (e.g., Fetch Places List with filters: amenity, price, location)
Forward request for processing (filters applied)
Apply business rules and filters (e.g., price, location, amenities)
Fetch places list based on filters
Return list of places
Generate success response with list of places
Return success (200 OK) with places data
User
API
BusinessLogic
Database

This sequence diagram illustrates the process of fetching a list of places in the Hbnb system. It details how the User, API, Business Logic, and Database interact to retrieve places based on specific filters.

1. **User initiates a request for places:**
   - The User sends an API request to fetch a list of places.
   - The request may include filters such as `amenity`, `price range`, and `location`.

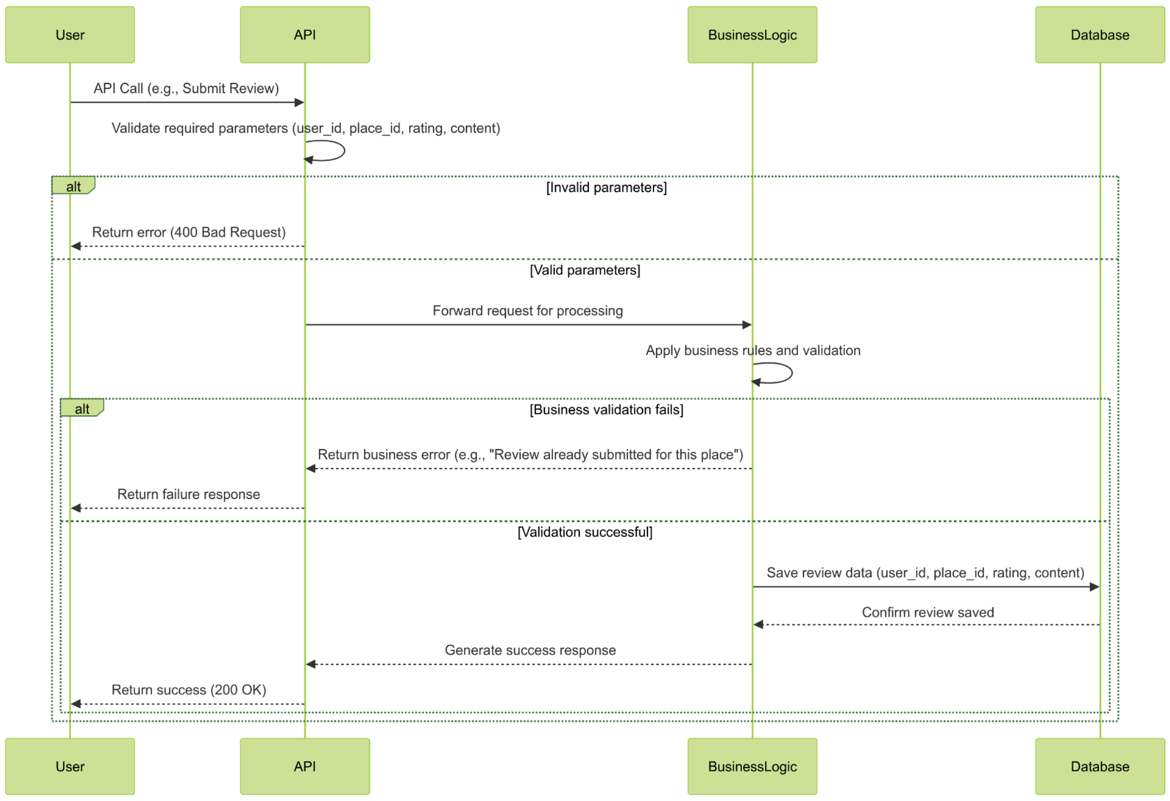
2. **API forwards the request to Business Logic:**
   - The API receives the request and forwards it to the Business Logic layer while applying the provided filters.

3. **Business logic applies filtering rules:**
   - The Business Logic processes the request by applying the given filters (e.g., filtering by price, location, and amenities).

4. **Database query execution:**
    - The Business Logic requests the Database to retrieve places that match the applied filters.
    - The Database executes the query and returns the filtered list of places.

5. **Generating the success response:**
    - The Business Logic formats the retrieved place data into a structured response.
    - The API sends a 200 OK response back to the User, including the list of matching places.

Review Submission Sequence Diagram:


User
API
BusinessLogic
Database
API Call (e.g., Submit Review)
Validate required parameters (user_id, place_id, rating, content)
alt
[Invalid parameters]
Return error (400 Bad Request)
[Valid parameters]
Forward request for processing
Apply business rules and validation
alt
[Business validation fails]
Return business error (e.g., "Review already submitted for this place")
Return failure response
[Validation successful]
Save review data (user_id, place_id, rating, content)
Confirm review saved
Generate success response
Return success (200 OK)
User
API
BusinessLogic
Database

This sequence diagram outlines the process of submitting a review in the Hbnb system. It demonstrates the interactions between the User, API, Business Logic, and Database to ensure proper validation and storage of user reviews.

1. **User initiates a review submission:**
   - The User sends an API request to submit a review for a specific place.
   - The request includes `user_id`, `place_id`, `rating`, and `content`.

2. **API validation:**
   - The API checks if all required parameters are present and valid.
   - If any parameter is missing or incorrect, the API returns a 400 Bad Request error to the user.

Mathieu ZUCALLI
Sebastien GEORGESCU

3. **Forwarding to Business Logic:**
   - If the validation passes, the API forwards the request to the Business Logic layer for processing.

4. **Business logic validation:**
   - The Business Logic applies specific rules, such as verifying whether the user has already submitted a review for the given place.
   - If a business rule is violated (e.g., "Review already submitted for this place"), an error response is sent back to the API, which then informs the user.

5. **Saving review data:**
   - If all validations succeed, the Business Logic stores the review in the Database.
   - The Database confirms that the review has been successfully saved.

6. **Returning a success response:**
   - The Business Logic generates a success response.
   - The API sends a 200 OK response back to the user, confirming that the review has been successfully submitted.