Holberton’s Bed and Breakfast

Unified Modeling Language

Contents

Introduction

Purpose and Scope

This technical document serves as a comprehensive blueprint for the development of HBnB (Holberton’s Bed and Breakfast). Its primary purpose is to provide a clear and detailed reference for the system's architecture and design, guiding the implementation phases of the project.

Project Overview

HBnB is a platform designed to connect property owners with potential guests, facilitating short-term rentals and unique accommodation experiences. The application encompasses several key functionalities:

1. User Management: Allowing users to register, update their profiles, and operate as either guests or hosts.
2. Place Management: Enabling hosts to list their properties with details such as description, price, and location.
3. Review System: Facilitating guest feedback through ratings and comments on their stays.
4. Amenity Management: Associating various amenities with listed properties to enhance guest experiences.

Technical Architecture

The HBnB application is built on a three-layered architecture, ensuring modularity, scalability, and maintainability:

1. Presentation Layer: Handles user interactions through services and API endpoints, managing input/output and user interface elements.
2. Business Logic Layer: Contains the core models (User, Place, Review, Amenity) and implements the rules governing the platform's functionality.
3. Persistence Layer: Manages data storage and retrieval, interfacing with the database to perform CRUD operations.

Document Structure

This technical document is organized into several key sections:

1. High-Level Architecture: Presents a package diagram illustrating the layered structure and interactions between components.

2. Business Logic Layer: Details the core entities through a comprehensive class diagram, explaining their attributes, methods, and relationships.
3. API Interaction Flow: Provides sequence diagrams for critical API calls, demonstrating the flow of information across the system's layers.

This detailed technical documentation aims to ensure a clear understanding of the HBnB system's design and functionality, facilitating efficient development and future maintenance of the application.

High-Level Architecture

High-Level Architecture is the structured framework used to design, describe and communicate the components and their interactions within a system. This helps people in a team understand how each part of the system fits together and interacts with each other. The design should be simple and easy enough for not only technical people but also non-technical people to comprehend as well.

Package Diagram

A package diagram is a type of Unified Modeling Language (UML) diagram mainly used to represent the organization and the structure of a system in the form of packages. In this case, the package diagram represents the three-layered architecture of the HBnB project. Each layer plays a distinct role in separating concerns and ensuring modularity, scalability, and maintainability.

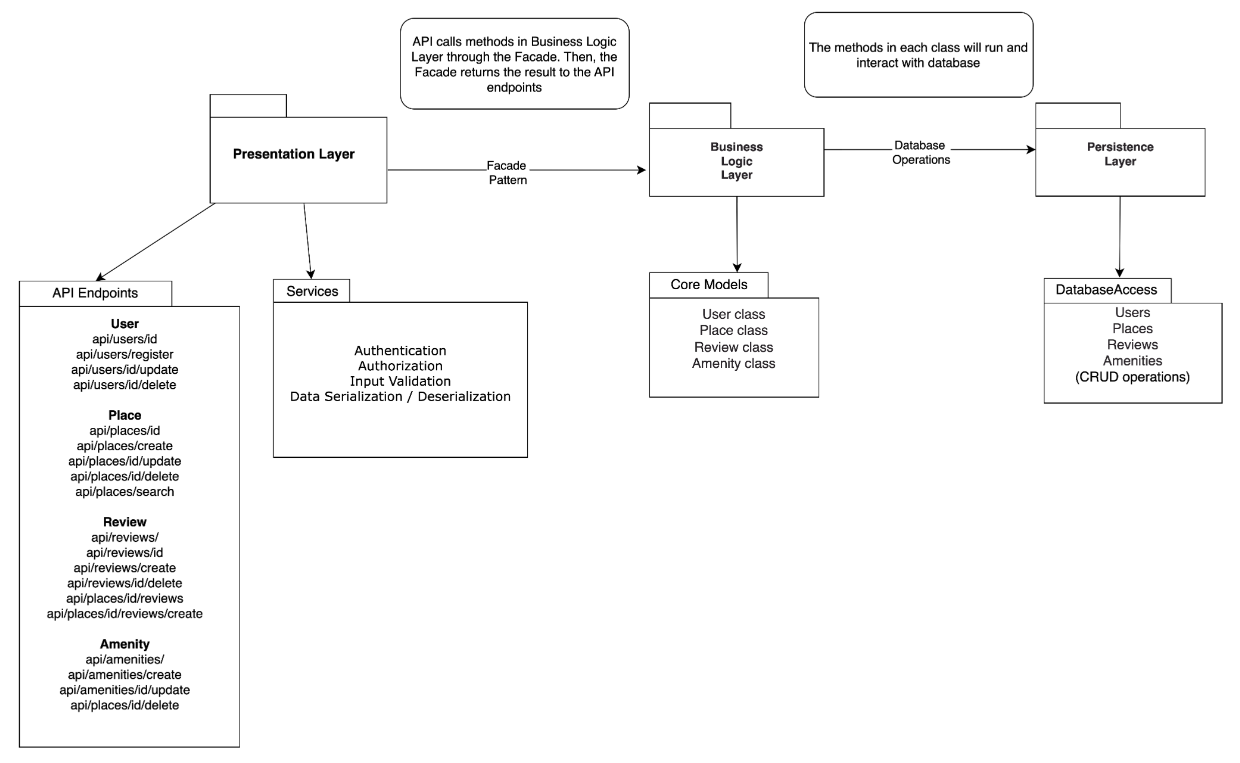

Figure 1. High-Level Package Diagram

Presentation Layer

The presentation layer is responsible for handling user input and displaying the user interface for mobile and website. It includes all the services and APIs that are exposed to the users. The HBnB app allows users to search for accommodations using filters, make reservations, leave reviews or contact the hosts.

API Endpoints

API Endpoints is a digital location where an API receives requests about a specific resource on its server. It enables clients to interact with user-related functionalities such as creating, updating, fetching, and deleting user profiles.

Example of API endpoints:

1. Endpoint: /api/users/id
   Description: This endpoint is used to retrieve the details of a specific user by their unique identifier (ID). It fetches the details of a user by their unique ID.
2. Endpoint: /api/users/register
   Description: This endpoint is used to register a new user in the system. It accepts user

details such as name, email, and password, and creates a new user profile in the database.

3. Endpoint: /api/users/id/update
   Description: This endpoint is used to edit or update the details of an existing user by their unique identifier (ID). The client sends the updated data for the user, such as their name, email, or password, and the server updates the user's profile in the system.
4. Endpoint: /api/users/id/delete
   Description: This endpoint is used to delete a user from the system by their unique identifier (ID). When a request is made to this endpoint, the specified user is removed from the system's database.

Services

Services are components or functionalities that interact with the user interface or external clients. These services enable the front-end to communicate with the back-end business logic, ensuring that the user can interact with the application effectively. The services are typically responsible for managing requests, processing them, and returning the appropriate responses to the user interface.

1. Authentication and Authorization Services: These services manage the user authentication (login, sign up) and ensure that only authorized users can access specific functionalities
2. Input Validation Services: These services ensure the security and integrity of data that flows through the system. By validating and sanitizing user inputs early on, this service prevents malicious data from entering the application, reduces errors, and improves user experience by providing clear feedback on invalid inputs.
3. Data Serialization/Deserialization Services: These services transform the data returned by the back-end into formats that the front-end can use (e.g., converting database records into JSON objects). They also handle requests from the front-end, parsing the incoming data into formats that the back-end can process.

Facade Pattern

Facade Pattern is a structural design pattern that provides a simplified interface to a set of interfaces in a subsystem (a library or a framework), making it easier to use. This pattern hides the complexities of the subsystem and provides a simpler interface for the client code.

How the Facade Pattern Simplifies Interactions in HBnB project:

1. **Unified Interface**: Instead of a client needing to call various methods from UserRepository, PlaceRepository, and ReviewRepository separately, they can use a single HBnBFacade class that abstracts these calls.
2. **Reduced Complexity**: A client wants to book a place. Instead of understanding how to validate the user, check place availability, create a booking, and handle payments, they can call a single method like facade.bookPlace(userId, placeId, startDate, endDate). The facade takes care of all the underlying complexities.

3. **Improved Maintainability**: If a new payment method is integrated, changes can be made in the facade without affecting the user interface or any client code that uses the facade.

Business Logic Layer

Business Logic Layer contains the application's business models and logic that represent the entities in the system (e.g., User, Place, Review, Amenity). These classes interact to represent the core entities of the system and implement the rules of the HBnB platform. Also, they encapsulate key data and behaviors, managing user interactions, property listings, reviews, and amenities. More importantly, they are the backbone of the platform functionality, ensuring data integrity, and coordinating various operations.

User Class

The User class represents a person using the HBnB platform, either as a guest or a host. It encapsulates all the logic related to user actions and attributes.

Example Use Case:

> Guest: A user signs up, signs in, browses places, and books a stay. The User class manages the user profile, authenticates the user, and tracks their bookings or writes reviews.
>
> Host: A host lists their properties, updates details including amenities, and interacts with potential guests. The User class handles managing property listings, editing details, and responding to booking requests.

Place Class

The Place class represents an accommodation listed on the HBnB platform. This class contains all the attributes of a place and the logic for managing listings, reservations, and reviews.

Example Use Case:

> A host lists a house for rent on the platform. The Place class manages the property details like price, location, and amenities. Guests can view the place's details, book it for specific dates, and leave reviews.

Review Class

The Review class represents feedback given by a guest about a place or a host. It allows users to rate their experience and leave comments.

Example Use Case:

After staying at a place, a guest leaves a review describing their experience. The Review class manages the logic of storing the review, associating it with the guest and the place, and calculating the overall rating of the property based on multiple reviews.

Amenity Class

The Amenity class represents the features or facilities available at a specific place. Each property can have multiple amenities, which are used to describe the conveniences offered to guests.

Example Use Case:

A user is booking a hotel room online and wants to filter the search results by specific amenities such as Wi-Fi, parking, or a swimming pool.

Persistence Layer

Persistence Layer: is responsible for managing data storage, retrieval, and communication with databases. This layer interacts with databases (or other storage systems) to perform CRUD (Create, Read, Update, Delete) operations on the entities defined in the Business Logic Layer. It ensures that data is stored efficiently and securely while providing an abstraction layer between the application and the database.

The Repositories

The repositories serve as a layer between the Business Logic Layer and the Persistence Layer, providing an interface for performing CRUD (Create, Read, Update, Delete) operations on the various entities. By isolating data access logic into repositories, the application maintains a clean separation of concerns, which enhances maintainability, testability, and scalability.

User Repository

The User Repository manages the interactions with the database for user-related operations. It encapsulates the logic for creating, reading, updating, and deleting user records.

Key Responsibilities:

- Create: Adds new users to the database.
- Read: Fetches user details based on various criteria (ID, email).
- Update: Modifies existing user information.
- Delete: Removes users from the database

Place Repository

The Place Repository handles interactions with the database regarding property listings. It provides methods to manage places that users can make a reservation or list.

Key Responsibilities:

- Create: Adds new property listings to the database.
- Read: Retrieves information about specific places or all places listed by a user.

- Update: Modifies details of existing property listings.
- Delete: Removes property listings from the database.

Review Repository

The Review Repository manages the interaction with the database for user reviews related to properties. It encapsulates the logic for creating and retrieving reviews.

Key Responsibilities:
- Create: Adds new reviews to the database.
- Read: Retrieves reviews based on specific criteria.
- Delete: Removes reviews from the database.

Amenity Repository

The Amenity Repository handles interactions related to amenities associated with property listings. It allows managing the features available at each place.

Key Responsibilities:
- Create: Adds new amenities to the database.
- Read: Retrieves amenities associated with specific properties.
- Update: Modifies existing amenities.
- Delete: Removes amenities from the database.

The Business Logic Layer

The Business Logic Layer in the HBnB program manages the fundamental entities that represent use places, reviews, and amenities. This layer serves as an intermediate point between the presentation (API) and persistence (database) layers, including business rules and procedures. The class diagram represents the structure of these entities, including their attributes, methods, and relationships, as well as how they interact to carry out the application's primary functions.

The class diagram focuses on four primary entities within the business logic:
- User
- Place
- Review
- Amenity

These entities serve as the foundation for managing the system's fundamental functions, such as user registrations, property listings, reviews, and property-specific features. They collectively handle the interactions and processes that ensure the HBnB application's basic functionality. The following class diagram shows the relationships between these entities, as well as their properties and methods.

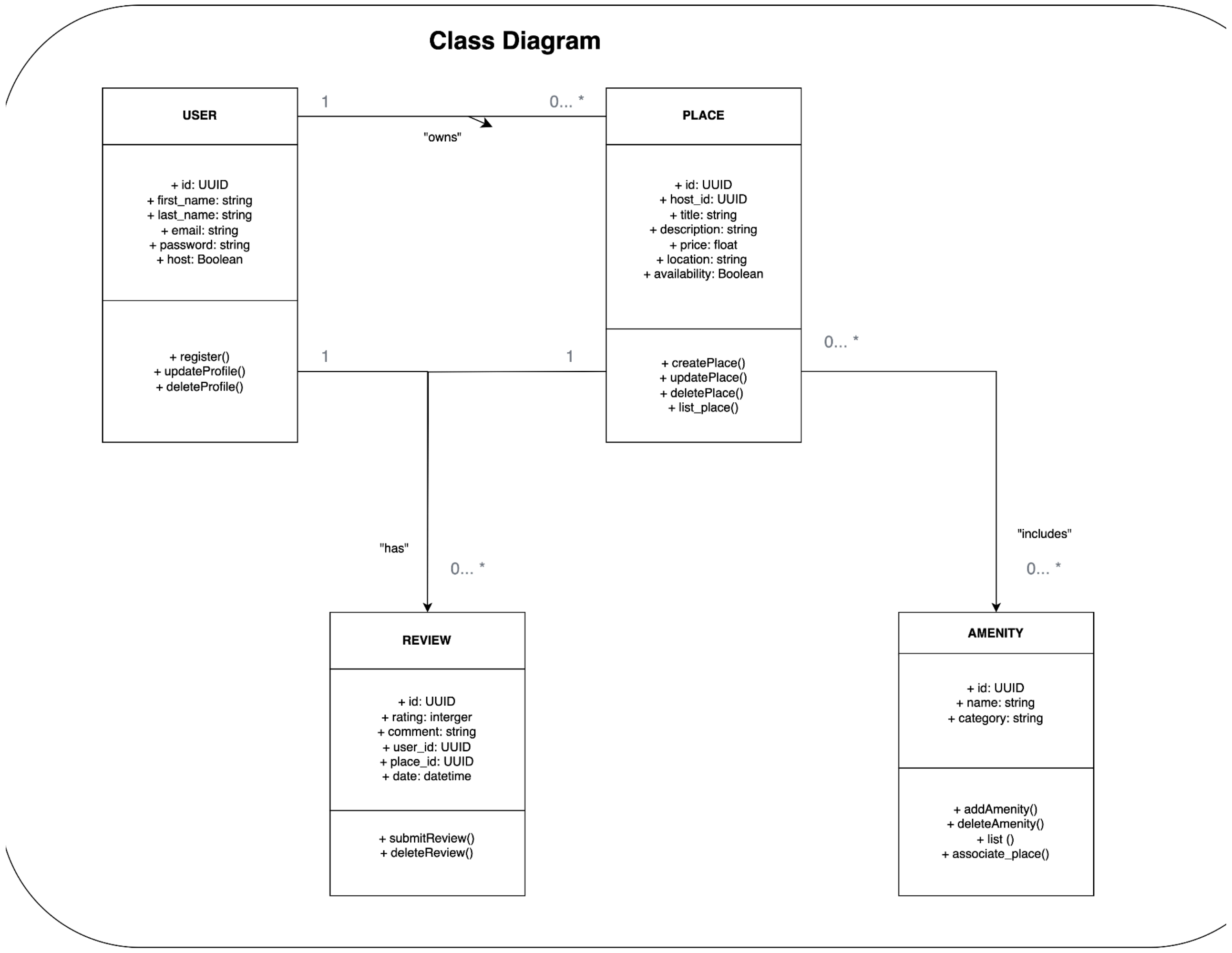

Figure 2. Class Diagram

Entity Descriptions

User

The User entity represents an individual within the HBnB application, managing user-specific attributes and methods related to registration, authentication, and profile management.

Attributes:

- id(UUID): A unique identifier for each user.
- first_name(string): The user’s first name.
- last-name(string): The user’s last name.
- email(string): The user’s email address, used for communication and login.
- password(string): The user's password for authentication.
- is_Owner(boolean): A flag indicating whether the user is a property owner.

Methods:

- register(): Handles new user registration by storing their details.
- updateProfile(): Allows the user to update their profile information.
- deleteProfile(): Allows the user to delete their profile from the system.

Places

The Place entity represents a property listed on the HBnB platform, managed by users who act as hosts. It includes attributes that define the place and methods for managing it.

Attributes:

- id(UUID): A unique identifier for each place.
- host_id(UUID): A reference to the user who hosts the place.
- title(string): The name or title of the property.
- description(string): A brief description of the property.
- price(float): The rental price of the location.
- location(string): The physical address or general location of the property.
- availability(boolean): A flag indicating whether the place is available for booking.

Methods:

- createPlace(): Creates a new place and stores its information.
- updatePlace(): Allows the user to update the place's details.
- deletePlace(): Allows the user to delete the place from the platform.
- list_place(): Publishes a new place on the platform for users to book.

Reviews

The Review entity captures user feedback for a place listed on the HBnB platform. It contains attributes related to the content and context of the review, as well as methods for managing reviews.

Attributes:

- id(UUID): A unique identifier for each review.
- rating(integer): A numeric rating (1 to 5) representing the user's evaluation of the place.
- comment(string): The text content of the review.
- user_id(UUID): A reference to the user who wrote the review.

- place_id(UUID): A reference to the reviewed place.
- date(datetime): The timestamp when the review was submitted.

Methods:

- submitReview(): Handles the submission of a new review by the user.
- deleteReview(): Handles the deletion of an existing review.

Amenities

The Amenity entity represents additional features or services associated with a place on the HBnB platform. Each amenity enhances the value or comfort of a place and can be linked to specific properties.

Attributes:

- id(UUID): A unique identifier for each amenity.
- name(string): The name or type of the amenity (e.g Wifi, Pool).
- category(string): The category under which the amenity is classified.

Methods:

- addAmenity(): Handles the addition of a new amenity to the system.
- deleteAmenity(): Handles the deletion of an existing amenity.
- list(): Retrieves a list of available amenities
- associate_place(): Links an amenity to a specific place.

Relationships Between Entities

Users can host several Places. This relationship is portrayed as a one-to-many association, which means that a user can have several property listings on the platform. The host_id in the Place object connects each location to the user who established it.

Places can have several reviews, each provided by a distinct user. This is a one-to-many relationship, with each review having a User (reviewer) and a Place (reviewed property).

A user can leave multiple reviews for different places. This relationship is also one-to-many, with each user able to provide feedback on several properties.

The link between place and amenity is many-to-many. A single location may offer several facilities (e.g., WiFi, Pool, Parking), and a single amenity can be associated with multiple places. This relationship is crucial for detailing the facilities a place offers.

Business Logic Fit

The class diagram represents the Business Logic Layer by encapsulating the system's key business entities and their interactions. Here's how these classes fit in the business logic:

1. **User Management**: The User class manages registration, property creation, and review submission, which are the primary operations that users are able to perform.
2. **Property Listings**: The Place class handles rental listings, allowing users to create, modify, and associate properties with a variety of amenities. This class is required for managing all property-related operations.
3. **Reviews System**: The Review class allows users to provide comments on properties, promoting confidence and transparency within the platform. Reviews have a huge impact on property popularity and user decision-making.
4. **Amenities Association**: The Amenity class enables places to be customized with features, allowing users to sort listings according to their preferences.

These entities work together to define the main functions of the application, developing a smooth and coherent experience for users while maintaining business rules such as user property ownership, the ability to post reviews, and the choice to add amenities.

API Interaction Flow

The following section contains sequence diagrams for major API calls in the HBnB application, as well as descriptions of the relationships between the various layers (Presentation, Business Logic, and Persistence). These graphics show how several components work together to accomplish each API request.

Each graphic contains an overview of the entities involved, a step-by-step analysis of the interactions, and design considerations taken to ensure system robustness and clarity.

User Registration API Call

The user registration API call handles the process of creating a new user account on the HBnB platform. It involves validation of user input, saving the new user's data to the database, and returning a confirmation to the client. These are the key components involved:

- **User**: The client initiating the registration process.
- **Presentation layer(API)**: Receives and validates the registration request.
- **Business Logic**: Processes the data and enforces business rules.
- **Persistence Layer (Database)**: Saves the new user information.

Interaction Flow

1. The **User** initiates the process by attempting to create a new account.
2. The **API** receives the initial request and responds by sending back the **Sign Up Form** to the user.
3. The **User** fills out the form and submits all required information (e.g., Name, Email, Phone Number, etc.).
4. The **API** passes the submitted data to the **Business Logic Layer (User Model)** to check for any missing or invalid data.
5. If the data is invalid or incomplete, the **API** returns a **400 Bad Request** to the user and sends the form again, prompting the user to correct the errors.
6. If the data is valid, the **API** passes the validated data to the **Business Logic Layer** for final validation and processing.
7. Once validated, the **Business Logic** sends a request to the **Database** to save the new user's information.
8. The **Database** confirms the successful registration of the user.
9. The **Business Logic** returns a success message to the **API**.
10. The **API** responds to the User with a registration success message.

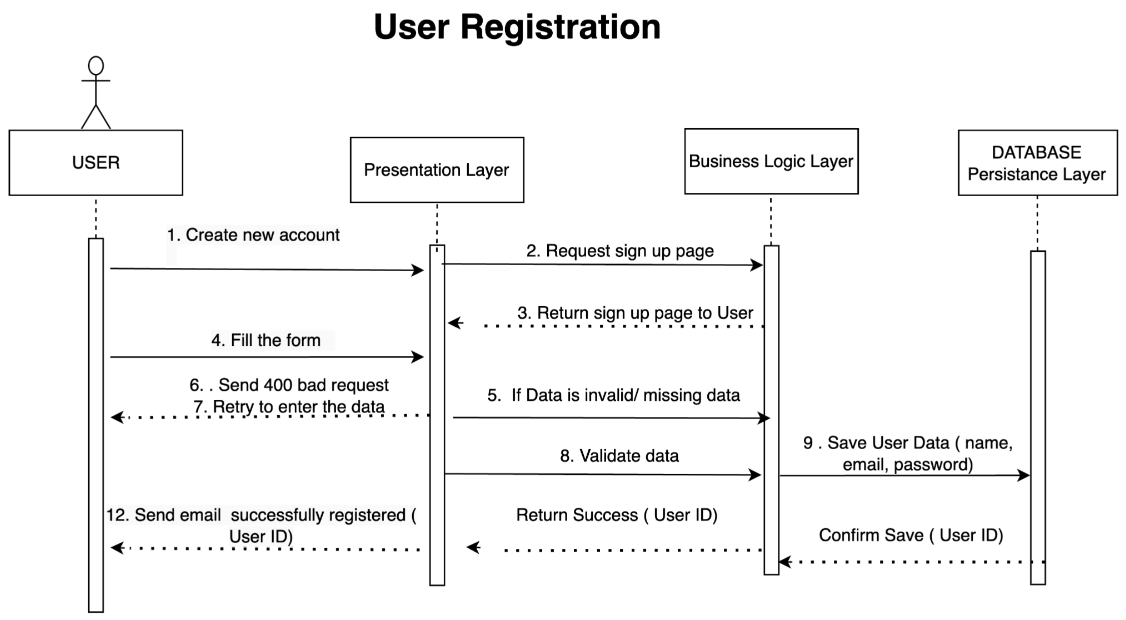
User Registration
USER
Presentation Layer
Business Logic Layer
DATABASE Persistance Layer
1. Create new account
2. Request sign up page
3. Return sign up page to User
4. Fill the form
6. . Send 400 bad request
7. Retry to enter the data
5. If Data is invalid/ missing data
8. Validate data
9 . Save User Data (name, email, password)
12. Send email successfully registered (User ID)
Return Success (User ID)
Confirm Save (User ID)

Figure 3. User registration sequence diagram

Place Creation API Call

This API enables users to generate new property listings on the site. This API validates property facts and stores listing information in the database. Key components involved:

- **User (Host)**: The property owner is creating a new location.
- **Presentation layer(API)**: Receives and validates place creation requests.
- **Business Logic (Place Model)**: Ensures that the location details follow business rules.
- **The Persistence Layer (Database)** stores the location listing information.

Interaction Flow

1. The **Host User** initiates the process by attempting to create a new place.
2. The **API** receives the initial request and responds by sending back the **Place Creation Form** to the user.
3. The **User** fills out the form and submits all required information (e.g., Location, Price, Description, etc.).
4. The **API** passes the submitted data to the **Business Logic Layer (Place Model)** to check for any missing or invalid data.
5. If the data is invalid or incomplete, the **API** returns a **400 Bad Request** to the user and resends the form, prompting the user to correct any errors.

6. If the data is valid, the **API** passes the validated data to the **Business Logic Layer** for final processing.
7. The **Business Logic Layer** forwards the place details to the **Database** for storage.
8. The **Database** confirms the successful creation of the place.
9. The **Business Logic Layer** returns a success message to the **API**.
10. The **API** responds to the **Host User** with a confirmation message indicating that the place has been successfully created.

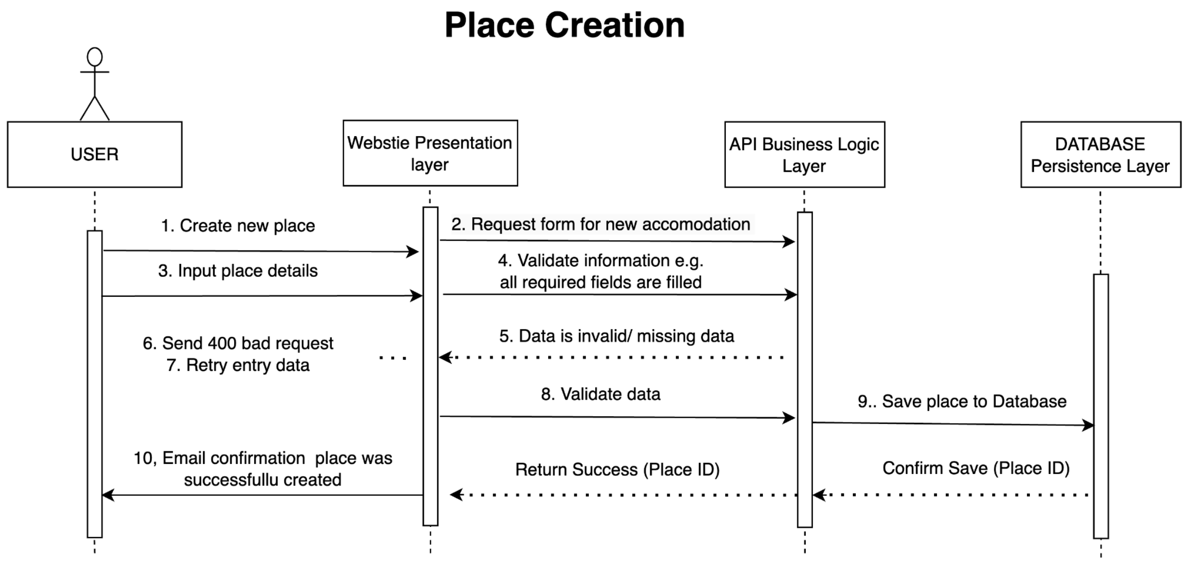
Place Creation
USER
Webstie Presentation layer
API Business Logic Layer
DATABASE Persistence Layer
1. Create new place
2. Request form for new accomodation
3. Input place details
4. Validate information e.g. all required fields are filled
5. Data is invalid/ missing data
6. Send 400 bad request
7. Retry entry data
8. Validate data
9.. Save place to Database
10, Email confirmation place was successfullu created
Return Success (Place ID)
Confirm Save (Place ID)

Figure 4. Place creation sequence diagram

Review Submission API Call

The review submission API call handles the process of submitting a new review for the accommodation on the HBnB platform. It involves signing in, user and password validation, retrieving user's data from the database, user filling the form and the data for review repository being updated in the database. These are the key components involved:

- **User:** The client signing into the system and submitting the review.
- **Presentation layer(API)**: Validates the review input and sends the request.
- **Business Logic**: Processes the data and enforces business rules.
- **Persistence Layer (Database)**: Saves the new review in the database

Interaction Flow

1. The **User** clicks the “sign-in” button on the HBnB homepage.
2. The **API** sends a request for a sign-in page to the server.
3. The **API** receives the sign-in page.
4. The **API** passes the page to the Presentation Layer for user to interact
5. The **User inputs** username and password.
6. The **API** validates the username and password in the **Business Logic Layer**. If they are valid, a request for data access will be sent to the **Database**.
7. The **Database** sends back user’s data to web API
8. Operations in **The Business Logic Layer** return the user's profile page to the **Presentation Layer(interface)**.
9. The **User** chooses a property to review.
10. The **API** sends a request for the review form to the server.
11. The **API** receives the page and returns to the Presentation Layer.
12. The **User** writes a review and submit it to the chosen property page.
13. The **Business Logic Laye**r sends data(a review) to the database.
14. The **Database** receives the data and saves it.
15. The data has been updated in the **Persistence Layer.**
16. The **API** returns the updated page to the **Presentation Layer.**
17. The review is added into the property’s page.

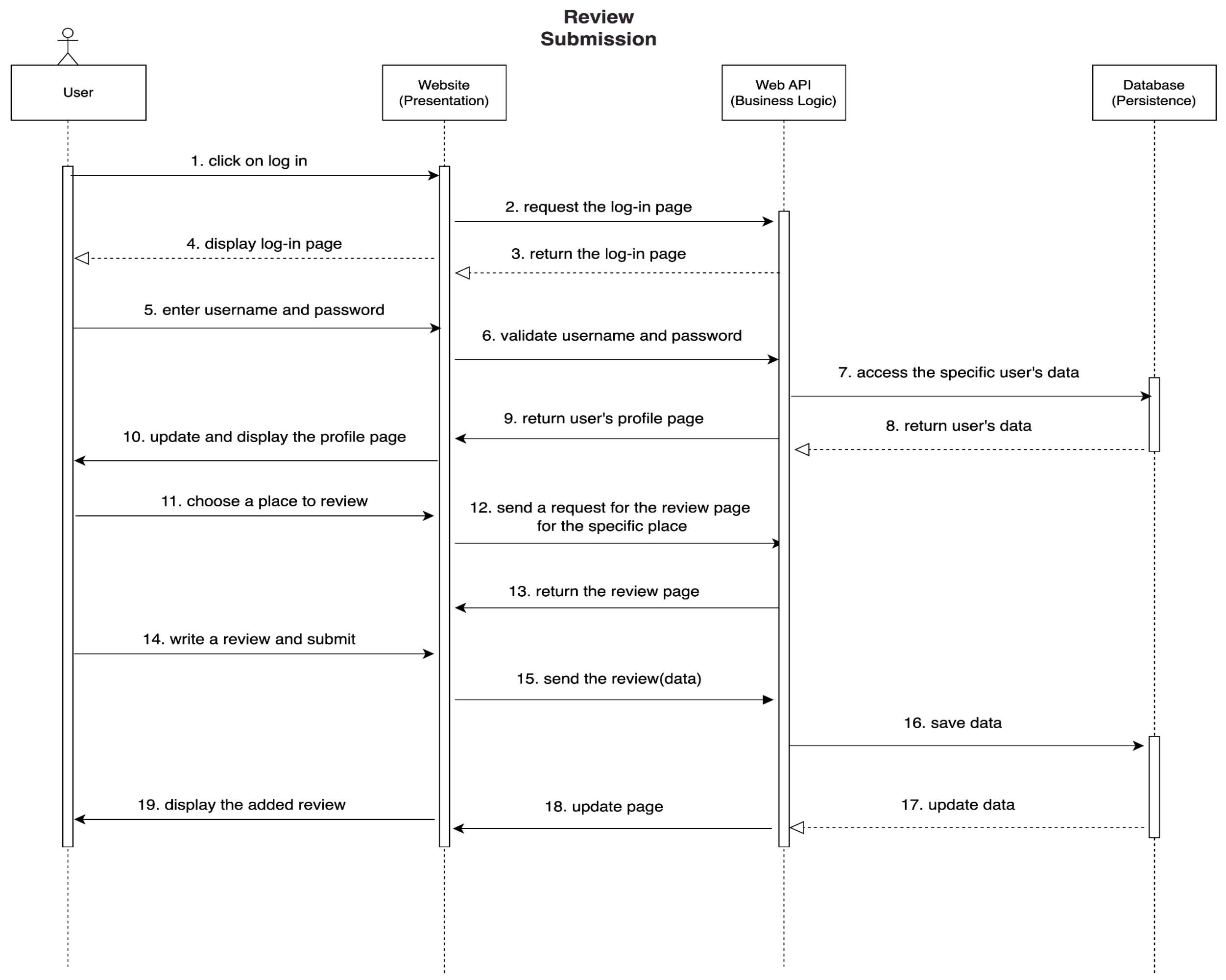
Review Submission
User
Website (Presentation)
Web API (Business Logic)
Database (Persistence)
1. click on log in
2. request the log-in page
3. return the log-in page
4. display log-in page
5. enter username and password
6. validate username and password
7. access the specific user's data
8. return user's data
9. return user's profile page
10. update and display the profile page
11. choose a place to review
12. send a request for the review page for the specific place
13. return the review page
14. write a review and submit
15. send the review(data)
16. save data
17. update data
18. update page
19. display the added review

Figure 5. Review submission sequence diagram

Fetching a List of Places API Call

The user fetching a list of places API Call handles the process of getting a list of properties on the HBnB platform. It involves searching for properties, and filtering the result. These are the key components involved:

- **User**: The client types in search details and use filter option
- **Presentation layer(API):** Receives and validates the input.
- **Business Logic**: Processes the data and enforces business rules.
- **Persistence Layer (Database)**: Sends back the data requested.

Interaction Flow

1. The **User** inputs their search details in the search bar and uses the filtering function to scope down the search result.
2. The **API** requests the page that displays the list of properties based on the filters
3. The **Business Logic Layer** validates the input before sending a request to database
4. The **Persistence Layer** receives the request and fetches the data based on the filters. Then, the database sends the data back to the **Business Logic Layer.**
5. The **API** receives and processes the data and returns the response to the **Presentation Layer.**
6. The requested page is displayed in the **Presentation Layer.**

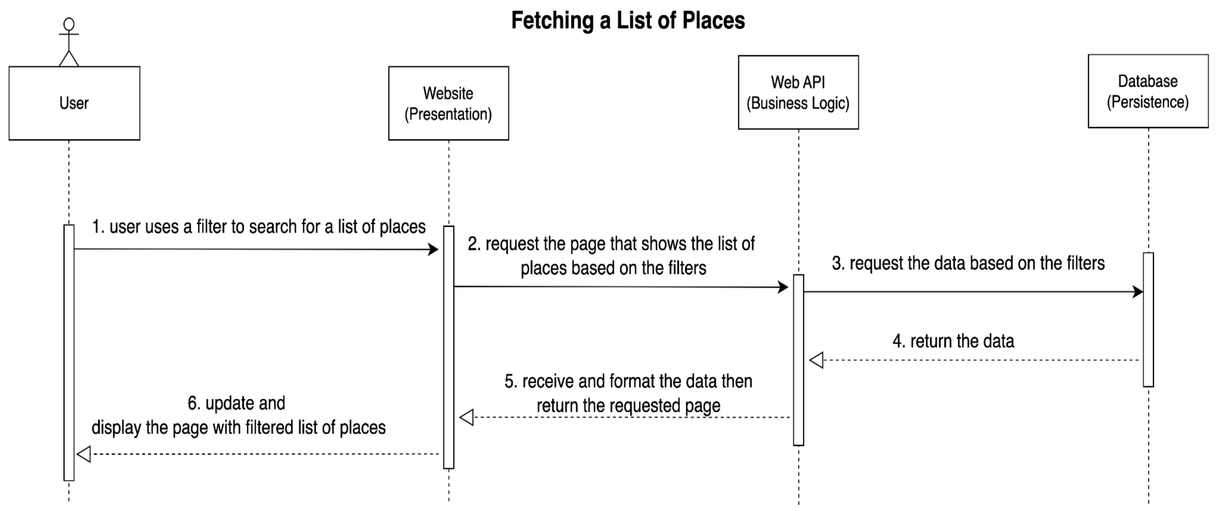
Fetching a List of Places
User
Website (Presentation)
Web API (Business Logic)
Database (Persistence)
1. user uses a filter to search for a list of places
2. request the page that shows the list of places based on the filters
3. request the data based on the filters
4. return the data
5. receive and format the data then return the requested page
6. update and display the page with filtered list of places

Figure 6. Review submission sequence diagram

Design Decisions

Layered Architecture: The interaction flows are separated into three layers (Presentation, Business Logic, and Persistence) to ensure clear separation of concerns, maintainability, and scalability.

Validation: In each stage, input data is thoroughly checked in the API and Business Logic Layers to ensure data integrity and security.

Consistency: UUIDs are employed across entities to provide unique identification for Users, Places, Reviews, and Amenities, ensuring data consistency across the system.

The sequence diagrams show the HBnB application's overall architecture, with the Presentation Layer handling external interactions (API calls), the Business Logic Layer applying the system's rules and logic, and the Persistence Layer managing long-term data storage. These diagrams provide a complete perspective of how API requests are handled, ensuring transparency and simple understanding for developers and users.

Conclusion

In summary, the HBnB system is designed on a strong, layered architecture that encourages scalability, maintainability, and modularity. By separating the Presentation, Business Logic, and Persistence layers, the program ensures that user interactions, business rules, and data management are clearly separated. The adoption of design patterns such as the Facade reduces interactions, reducing complexity for end users while allowing for future development.

The system supports core functionality including user registration, place construction, and review submissions with well-defined API interaction flows. These flows provide a clear picture of how data transfers between layers, ensuring that the platform operates smoothly and consistently.

Overall, this document provides a thorough guide for designing, developing, and maintaining the HBnB platform, providing vital insights into the system's architecture and assuring its success as a scalable, user-friendly property rental solution.