HBnB Project Technical Documentation

Introduction

This document offers an extensive technical overview of the HBnB project, detailing its architecture, design principles, and system interactions. The primary goal is to act as a reference for developers, guiding the implementation process while ensuring consistency and clarity throughout the project.

The HBnB application adopts a three-tiered architecture, segmenting the system into:

- **Presentation Layer**: Manages user interactions through services and APIs.
- **Business Logic Layer**: Holds the core application logic and models (e.g., User, Place, Review, Amenity).
- **Persistence Layer**: Oversees data storage and retrieval from the database.

A crucial design pattern employed in this architecture is the **Facade Pattern**, which streamlines communication between layers by offering a unified interface. This method enhances modularity, maintainability, and scalability.

Document Structure

- **High-Level Architecture**: Summary of the three-tiered architecture with a package diagram.
- **Business Logic Layer**: Comprehensive class diagram explaining core entities and their relationships.
- **API Interaction Flow**: Sequence diagrams depicting key API calls and interactions.

Each section includes explanatory notes that justify design choices and describe how components interact. This document aims to promote a structured and efficient development process for the HBnB project.

High-Level Architecture

The HBnB application employs a three-tiered structure, ensuring a distinct separation of responsibilities, modularity, and ease of maintenance. The three tiers—Presentation Layer, Business Logic Layer, and Persistence Layer—interact through a structured workflow utilizing the Facade Pattern to streamline communications.

Layer Breakdown

1. **Presentation Layer (User Interface & API Services)**

This tier is accountable for managing user interactions and API communications. It encompasses:

- **Handling Interface**: Oversees user input and UI rendering.
- **Service API**: Enables communication between the frontend and backend.
- **Key Methods**:
  - `renderUI()`: Displays the user interface.
  - `connectAPI(endpoint)`: Connects to API endpoints to retrieve or transmit data.

2. **Business Logic Layer (Model & Core Logic)**

This tier houses the core application logic and entity models, ensuring data processing and validation. It includes:

- **Models**: Represent fundamental entities (User, Place, Review, Amenity).

- **Data Processing & Validation**: Ensures accuracy before interacting with the database.
- **Key Methods**:
  - `processData(data)`: Handles incoming data and returns a success status.
  - `validateUser(userId)`: Confirms user identity and returns the corresponding user object.

3. **Persistence Layer (Database Access)**

This tier manages data storage and retrieval. It communicates directly with the database to execute CRUD operations.

- **Database Access**: Oversees all database interactions.
- **Key Methods**:
  - `saveData(data)`: Stores processed data in the database.
  - `retrieveData(query)`: Fetches data based on queries.

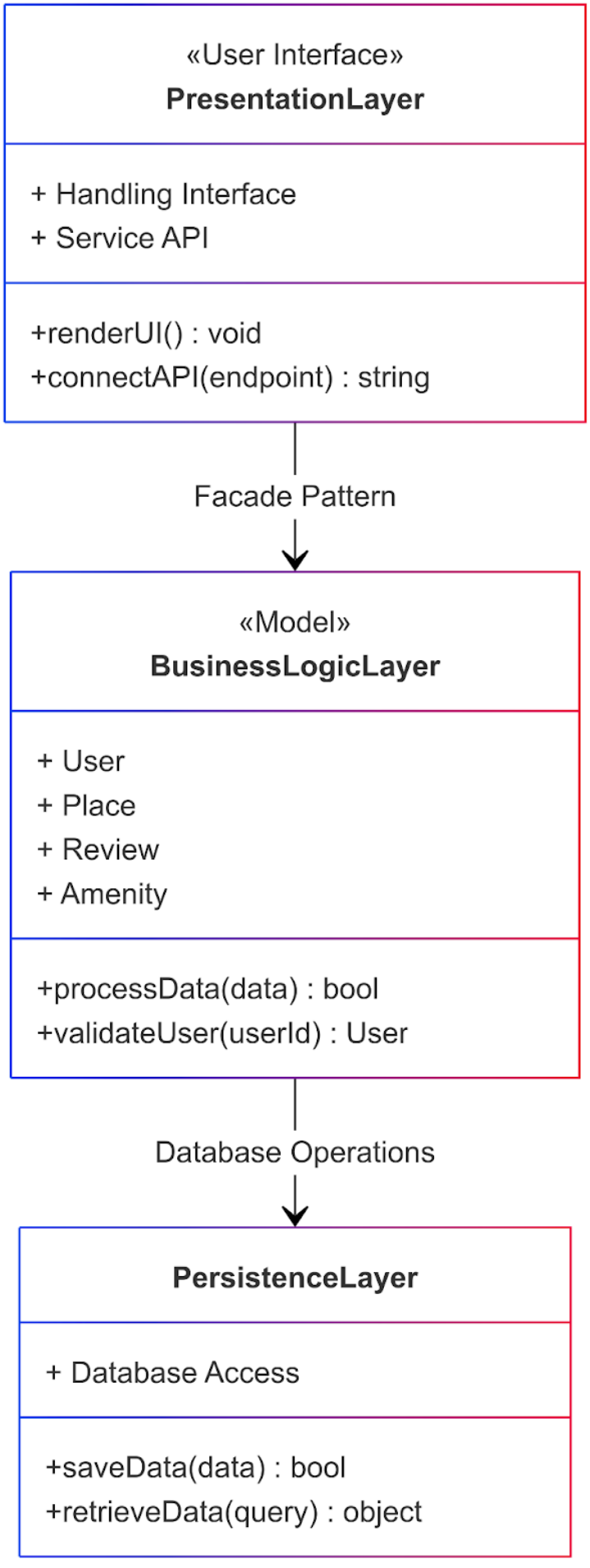
Communication Flow via Facade Pattern

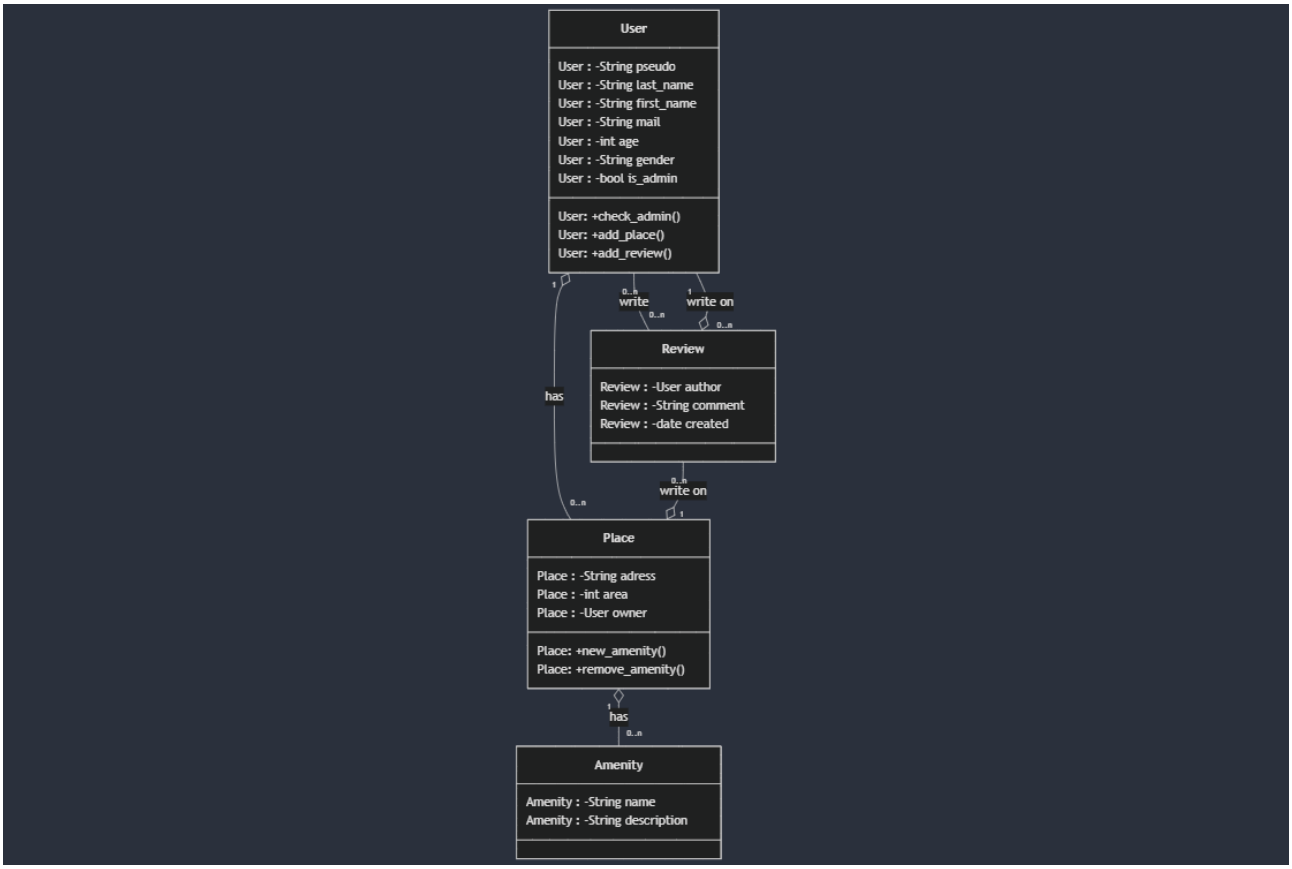
To uphold separation and modularity, the tiers interact through the Facade Pattern, which offers a simplified interface for inter-tier communication:

- The Presentation Layer communicates with the Business Logic Layer using service APIs.
- The Business Logic Layer processes data and interacts with the Persistence Layer for database operations.
- The Persistence Layer handles all database interactions and returns the requested data to the Business Logic Layer, which then processes it and sends it back to the Presentation Layer for display.

Diagram Representation

Below is a visual representation of the package structure using Mermaid.js:


«User Interface»
PresentationLayer
+ Handling Interface
+ Service API
+renderUI() : void
+connectAPI(endpoint) : string
Facade Pattern
«Model»
BusinessLogicLayer
+ User
+ Place
+ Review
+ Amenity
+processData(data) : bool
+validateUser(userId) : User
Database Operations
PersistenceLayer
+ Database Access
+saveData(data) : bool
+retrieveData(query) : object



3.1 Class description and relationships

3.1.1 User class

The user represent a person interacting with the HBNB platform. A user can add place, write reviews and it's possible to check if is an admin.

Attributes

- pseudo (String) – The user's username.
- last_name (String) – The user's last name.
- first_name (String) – The user's first name.
- mail (String) – The user's email.
- age (int) – The user's age.
- gender (String) – The user's gender.
- is_admin (bool) – Defines whether the user has administrative privileges.

Methods

- check_admin() – Verifies if the user has admin rights.
- add_place() – Allows the user to add a new place.
- add_review() – Enables the user to write a review.

Relationships

- **Has** (1 → 0..n) Review – A user can write multiple reviews.

- **Has** (1 → 0..n) Place – A user can own multiple places.

3.1.2 Review class
The review class represent a user feedback.

Attributes

- author (User) – The user who wrote the review.
- comment (String) – The review content.
- created (date) – The date the review was created.

Relationships

- **Written by** (1 → 0..n) User – A user can write multiple reviews.
- **Written on** (0..n → 1) Place – A review is linked to a single place.

3.3.3 Place class
The place class represent a rental location listed on HBNB.

Attributes

- address (String) – The physical address of the place.
- area (int) – The surface area in square meters.
- owner (User) – The user who owns the place.

Methods

- new_amenity() – Adds a new amenity to the place.
- remove_amenity() – Removes an existing amenity from the place.

Relationships

- **Has** (1 → 0..n) Review – A place can have multiple reviews.
- **Has** (1 → 0..n) Amenity – A place can include multiple amenities.

3.3.4 Amenity class
The amenity class represent additional features available in a place.

Attributes

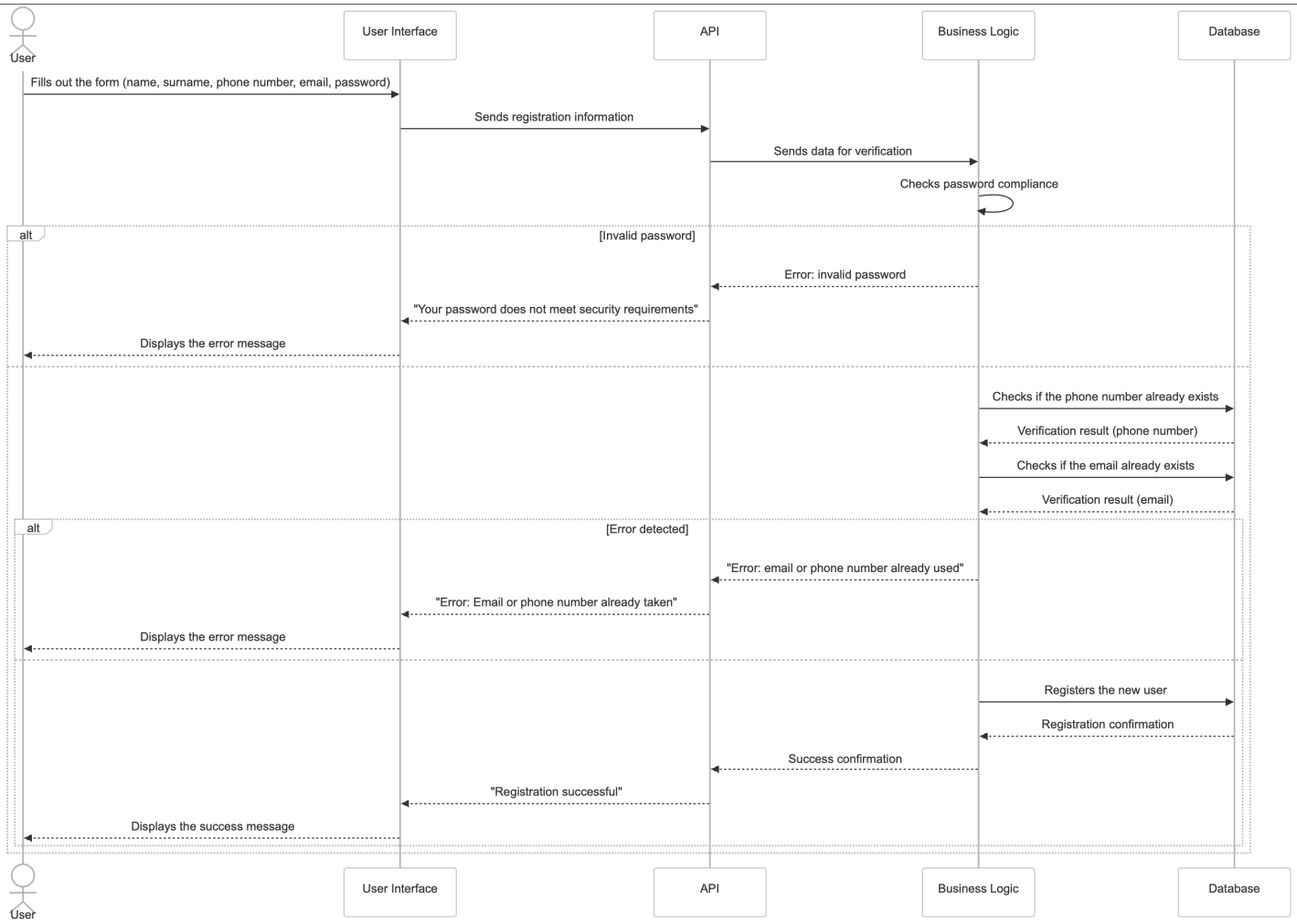
- name (String) – The name of the amenity.
- description (String) – A brief description of the amenity.

Relationships

- **Has** (0..n → 1) Place – An amenity belongs to a place.

UML sequence diagram:

User registration:



(https://www.mermaidchart.com/raw/546103ba-8c63-4450-985e-bc615a792955?theme=light&version=v0.1&format=svg)

This sequence UML diagram illustrates the process of registering a user in the HBnB Evolution application. It shows the sequence of interactions between the user, the interface, the API, the Business Logic layer and the database. The goal is to check and register a new user while managing possible errors.

Actors involved	*Description*
User	Completes the registration form and submits the request.
User Interface	Transmits user information to the API.
API	Processes the request and transmits the data to the Business Logic layer.
Business Logic	Checks the validity of the information entered (password, email, phone).
Database	Stores user information if it is valid and unique.

The process:

The user fills out the registration form with their first and last name, phone number, email, and password. Then the user interface sends this information to the API, which forwards the data to the Business Logic layer.
La couche Business Logic effectue plusieurs vérifications :

Checks if the password meets the security criteria. ✓
Checks if the phone number is already used in the database. ✓
Checks if the email is already used in the database. ✓

Two possible scenarios.

If all information is valid:

The user is registered in the database.

A confirmation of success is sent to the API, which informs the user interface.
The user receives a "Registration successful" message.

If an error is detected (invalid password, email or phone number already in use):

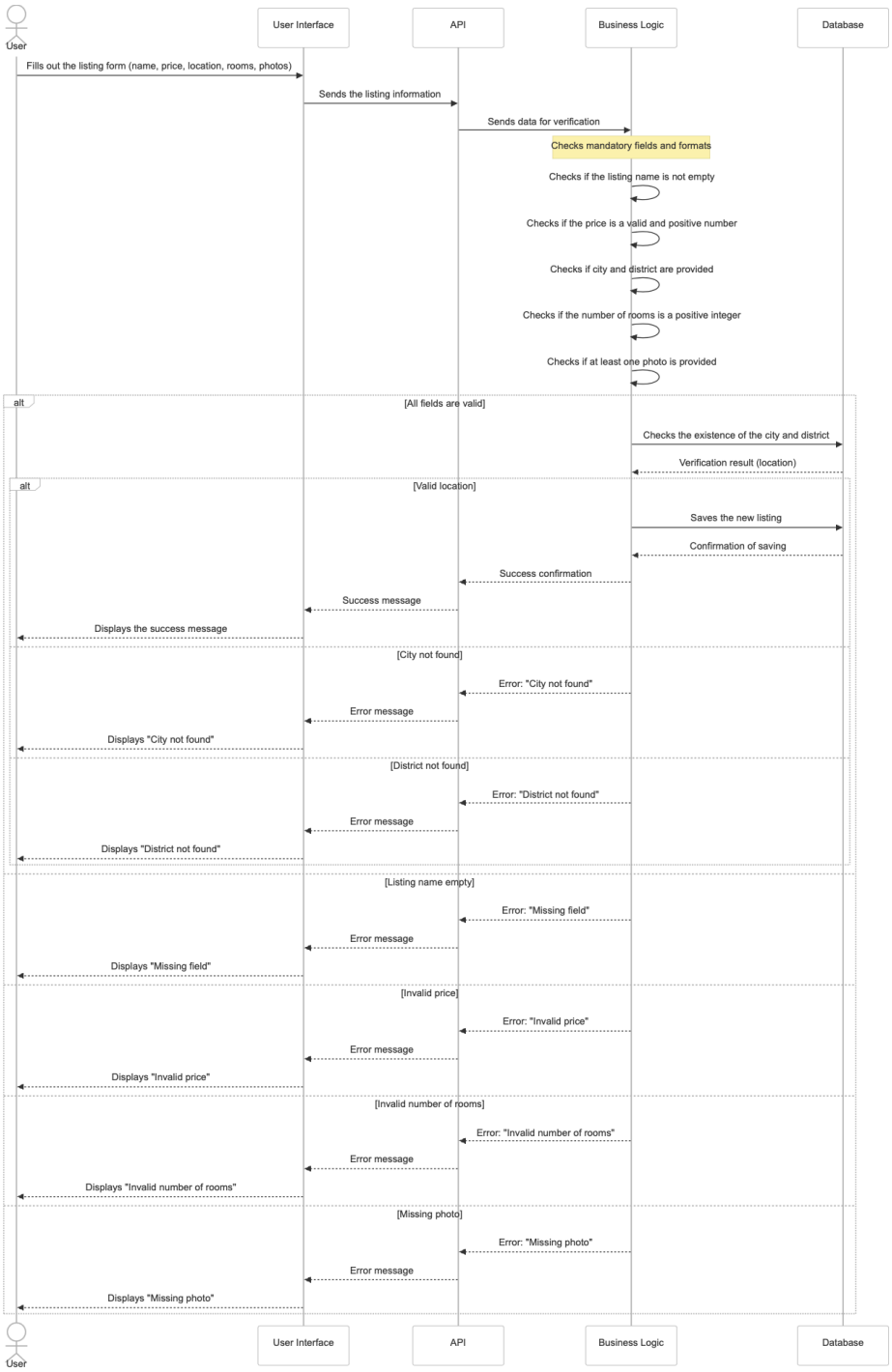
An error message is returned to the API, then to the user interface.
The user sees a message explaining the reason for the failure.

Error management:

- The diagram takes into account several types of possible errors during registration:

- Password does not comply *"Your password does not meet security requirements"*
- Email already in use "Email already taken"
- Phone number already in use *"Phone number already taken"*
- Technical error with the database *"Technical error, please try again later »*

Adding housing:



(https://www.mermaidchart.com/raw/d71ed22f-b4cf-4483-8a42-ad93eeef978f?theme=light&version=v0.1&format=svg)

This sequence UML diagram illustrates the process of adding housing in the HBnB Evolution application. It shows the interactions between the user, the interface, the API, the business layer and the database to validate and register a new housing ad.

Actors involved	Description
User	Completes the housing addition form and submits the request.
User Interface	Transmits the information from the form to the API.
API	Processes the request and transmits the data to the Business Logic layer.
Business Logic	Checks the information entered and applies the business rules.
Database	Stores housing information if it is valid and existing.

The process:

The user fills out the housing addition form with the following information :

- Name of the accommodation ✓
- Price ✓
- Location (city and neighborhood) ✓
- Number of rooms ✓
- Photos of the accommodation ✓

The user interface transmits this information to the API which sends the request to the Business Logic layer which will allow the checks to be made:

- Check if the name of the accommodation is not empty.
- Check if the price is valid and positive.
- Checks if the city and neighborhood are filled in.
- Checks if the number of pieces is a positive integer.
- Check if at least one photo is provided.

Two possible scenarios:

If all the information is valid ✓:

The Business Logic layer checks if the city and neighborhood exist in the database.
If the location is valid, the ad is saved in the database.
A confirmation of success is sent to the API, which informs the UI.
The user receives a message "Ad added successfully".

If an error is detected (missing required fields, invalid location):

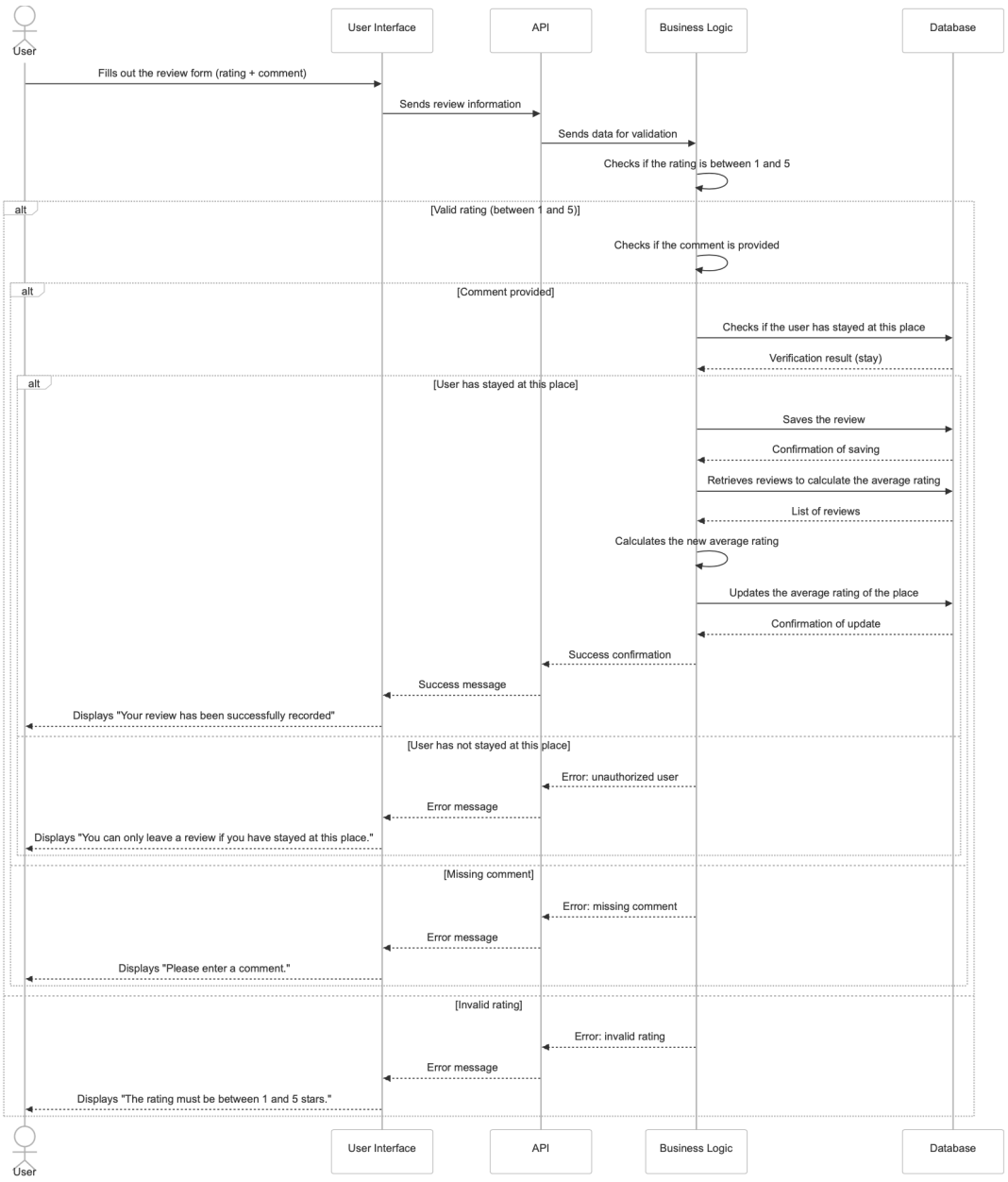
- An error message is returned to the API, then to the UI.
- The user sees a message explaining the reason for the failure.

Error Management :

The diagram takes into account several types of possible errors when adding a housing:

- Name of the empty dwelling *"Missing field"*
- Invalid price *"Invalid price"*
- Invalid number of rooms *"Invalid number of rooms"*
- Missing photo *"Missing photo"*
- City not found *"City not found"*
- District not found *"District not found »*

Submission of a notice



(https://www.mermaidchart.com/raw/ee3f86fe-caca-441b-ab32-e001f16d96eb?theme=light&version=v0.1&format=svg)

This sequence UML diagram illustrates the process of submitting a notice in the HBnB Evolution application. It describes the sequence of interactions between the user, the interface, the API, the business layer and the database when registering a notice on a housing. The objective is to validate the data entered, verify that the user has stayed in the accommodation, and then register the notice.

Actors involved	Description
User	Complete the notice form and submit the request.
User Interface	Reports the information to the API.
API	Processes the request and transmits the data to the Business Logic layer.
Business Logic	Checks the validity of the information and applies business rules.
Database	Stores the validated review and updates the average rating of the housing.

The process:

The user fills out the review form, entering a rating between 1 and 5 stars and a comment The user interface transmits this information to the API.
The API sends the request to the Business Logic layer, which performs several checks:

- Is the grade valid (between 1 and 5)?✓
- Is the comment filled in?✓
- Did the user stay in this accommodation?✓

Two possible scenarios:

If all the information is valid, the Business Logic layer checks in the database if the user has stayed in this accommodation. If it is confirmed, the notice is recorded in the database. Business Logic recalculates the average rating of the housing and updates the information. A confirmation of success is sent to the API, which informs the user interface. The user receives a message "*Your review has been successfully saved*".

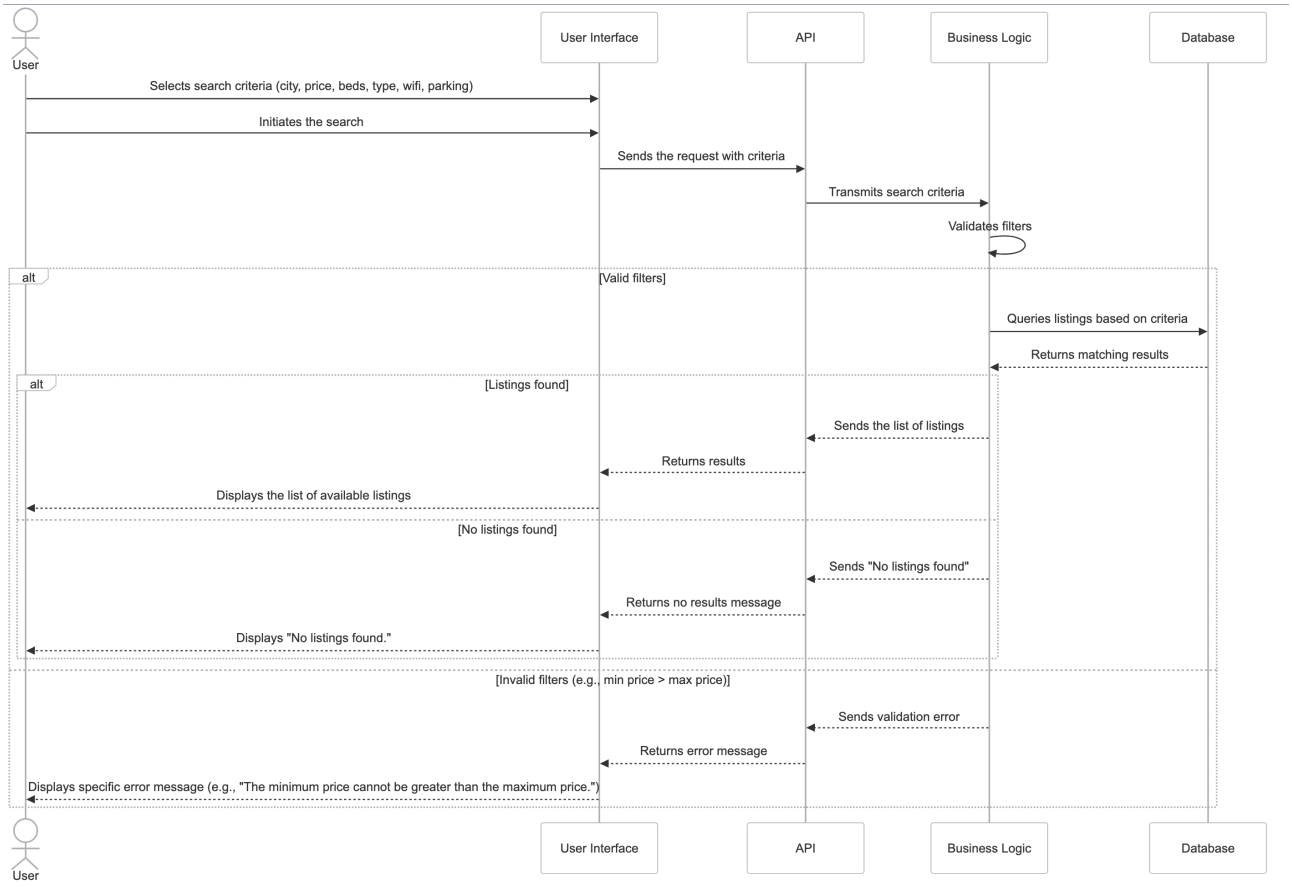
If an error is detected (invalid note, missing comment, unauthorized user):

- An error message is returned to the API, then to the user interface.
- The user sees a message explaining the reason for the failure.

Error Management :

- Invalid grade (less than 1 or greater than 5) *"The rating must be between 1 and 5 stars."*
- Missing comment *"Please enter a comment."*
- Unauthorized user (has never stayed in this accommodation) *"You can only leave a review if you have stayed at this place."*
- Technical problem with the database *"Technical error, please try again later."*

Housing recovery :



(https://www.mermaidchart.com/raw/6017b771-435b-4e5f-aecd-798ca5da2dac?theme=light&version=v0.1&format=svg)

This sequence UML diagram illustrates the process of retrieving available housing in the HBnB Evolution application. It shows the interactions between the user, the interface, the API, the business layer and the database when searching for housing with specific criteria. The goal is to filter the available housing according to the user's search criteria and return the relevant results.

Actors involved	Description
User	Selects search criteria and launches the query.
User Interface	Transmits the search criteria to the API and displays the results.

Actors involved	Description
API	Processes the request and transmits the criteria to the Business Logic layer.
Business Logic	Checks the validity of the filters and queries the database.
Database	Search for the corresponding housing and return the results.

The process:

The user selects his search criteria, such as:

- City
- Price (min / max)
- Number of beds
- Type of housing (fulle, shared, etc.)
- Availability of WiFi
- Presence of a parking lot

The user launches the search via the user interface then it sends the request with the criteria to the API. The API transmits these criteria to the Business Logic layer, which performs several checks:

- Are the filters valid? (a minimum price cannot be higher than the maximum price). ✓
- If the filters are valid, Business Logic sends a query to the database to retrieve the corresponding housings.✓

Two possible scenarios:

If housing is found, the list of housing is sent to the API which transmits the results to the user interface. The user sees the available housing displayed on his screen.

If no housing meets the criteria:

- A "*No listings found*" message is sent to the API, then to the user interface.
- The user sees a message "*No accommodation found.*"

Error Management :

The diagram takes into account several types of possible errors when recovering housing:

- No accommodation found "No listings found. »
- Invalid filters (ex: minimum price higher than the maximum price) "The minimum price cannot be greater than the maximum price. »
- Technical problem with the database "Technical error, please try again later. »

Conclusion of the UML documentation – HBnB Evolution

This technical documentation presents the design and architecture of the HBnB Evolution application, detailing its main layers, entities and interactions. The objective is to provide a clear and structured vision of how the system works before its implementation.

General Architecture (Package Diagram – Gabriel)

The architecture is divided into three main layers:
Presentation Layer (User Interface and API):
- This layer manages the interaction with the user and the communication via API services.

Business Logic Layer (Business Logic):
- Contains the essential business rules and ensures the link between the user interface and the database.
- Implements a Façade Pattern to simplify interactions.

Persistence Layer (Database):
- Manages the recording and retrieval of data via operations on the database.

This organization guarantees a clear separation of responsibilities and a better scalability of the system.

Business Model (Class Diagram – Damien)

The business model is based on four main entities:
User: Has an account, can add accommodations and leave reviews.
Place: Defined by an owner (User), an address and a surface area.
Review: Written by a user (User), it is attached to an accommodation (Place).
Amenity: Lists the equipment associated with an accommodation (Place).
The relationships between these entities respect the principles of composition and association in UML, ensuring clear and extensible modeling.

API Interactions (UML Sequence Diagrams – Brahim)

The main user interactions are represented through four UML sequence diagrams:
Registering a user:
- Checks the information entered (email, phone, password).
- Stores the user in the database if all the data is valid.

Adding an accommodation:
- Checks that all the required fields are filled in.
- Validates the existence of the city and neighborhood before saving.

Submitting a review:

- Verifies that the user has actually stayed in the accommodation before saving their comment.
- Updates the average rating of the accommodation after each new review.

Retrieving accommodations:

- Filters available accommodations according to the selected criteria (city, price, amenities, etc.).
- Returns relevant results or displays a message if no accommodation matches.

These scenarios ensure robust management of user interactions, with efficient error handling and a response adapted to each situation.