HBnB Evolution: Development guide

Start May 26, 2025 | End Aug 1, 2025

Authors:

Delphine COUTOULY-LABORDA | github.com/Delphes1980
Xavier LAFORGUE | github.com/XavierLaforgue

Project guidelines

For the development of a simplified AirBnB-like application

01

Introduction

Overview of the project

As part of the second trimester's main deliverable in the **Web Development track** of the **Holberton School formative program** (Holberton School), this document presents the technical architecture and design for **HBnB Evolution**, a simplified web application inspired by AirBnB. The application aims to replicate the original's core functionalities such as user management, property listings, amenities, and reviews.

This month-long project is divided into four phases:

1. **Documentation and Design** (current phase): Production of this technical document, including architecture, class models, and interaction diagrams.
2. **Business Logic and API Implementation**: Development of the application's core logic and RESTful API.
3. **Authentication and Database Setup**: Integration of secure user authentication and persistence mechanisms.
4. **Client Interface Delivery**: Implementation and deployment of a functional web client.

Purpose

The purpose of this document is to provide a clear, structured, technical blueprint that will guide the development and implementation of the application while ensuring consistency and maintainability throughout all phases of the project. The document aims to provide a comprehensive and detailed view of the system's structure, the interactions between its components, and the logic governing its core features.

Scope

HBnB Evolution is designed to support the core functionalities of a property rental platform (except booking and payment). These include:

- **User Management**: User registration, profile updates (including deletion), and distinction between regular users and administrators.
- **Place Management**: Registration, update, deletion, and listing of properties associated with a user.
- **Review Management**: Submission, modification, deletion, and listing of reviews for an existing property.
- **Amenity Management**: Specification, modification, deletion, and listing of amenities tied to an existing property.

Objectives of the Architecture

The system adopts a **layered architecture** to enforce a clean separation of concerns, increase maintainability, and support scalability. It leverages the **Facade design pattern** to streamline communication between layers, hiding the internal complexities of the business logic behind a unified interface.

Document structure

This document is organized as follows:

1. **High-Level Architecture**
    - Describes the system's overall structure using a three-layered approach: Presentation Layer, Business Logic Layer, and Persistence Layer.
    - Includes a package diagram illustrating component boundaries and inter-layer communication via the facade pattern.
2. **Business Logic Layer Design**
    - Provides a detailed class diagram capturing the core domain entities, their relationships, and business rules.
3. **API Interaction Flow**
    - Presents sequence diagrams for key user operations (user registration, place creation, review submission, and place listing).
    - Illustrates the flow of data and control between system layers for selected use cases.

Intended audience

This is a technical document and, as such, it is intended for an audience with a certain degree of proficiency in software development and system architecture. It assumes a working knowledge of UML (Unified Modelling Language), software design principles, and web application architecture.

02

High-Level Architecture

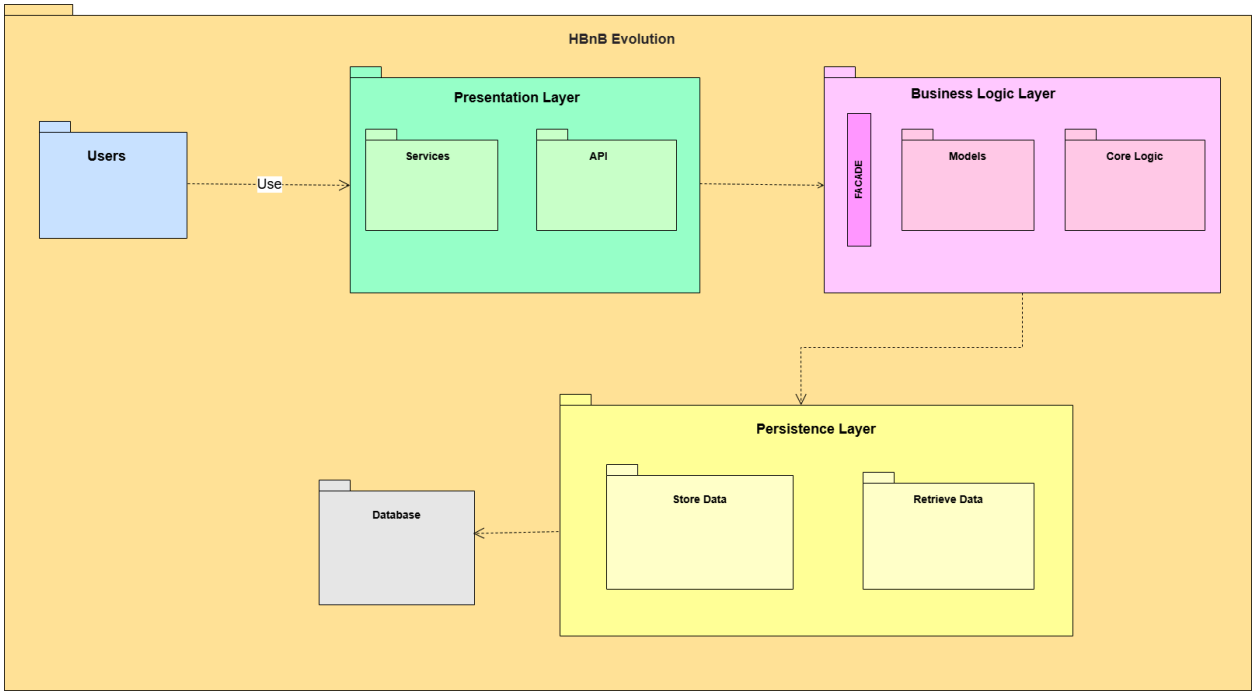

Fig. 1: HBnB Evolution high-level diagram.

This diagram provides a **fundamental overview of the HBnB Evolution application's overall architecture**. It clearly illustrates the system's core layered structure, the distinct responsibilities of each layer, and their essential communication pathways. It serves as a foundational blueprint for understanding the application's top-level design.

Key components

- **Presentation Layer:** The outermost layer, responsible for direct user interaction, API handling, initial validation, and data serialization/deserialization for display
- **Business Logic Layer (BLL):** The core of the application, containing and applying all business rules, performing complex validations, orchestrating operations, and defining the structure of business objects
- **Persistence Layer:** Exclusively responsible for data retention and retrieval, interacting with the database while abstracting its technical details from higher layers

Design Decisions & Rationale

- **Layered Architecture:** This fundamental design choice promotes clear **separation of concerns**, enhancing modularity, maintainability, scalability, and testability by assigning specific responsibilities to each layer
- **Facade Pattern (in BLL):** Employed in the BLL, the Facade pattern provides a **simplified, unified entry point** for the Presentation Layer to access complex business logic and data operations, effectively hiding internal workings and facilitating service execution
- **Strict Communication Flow:** Requests flow downwards (Presentation -> BLL -> Persistence), and responses travel upwards, ensuring a controlled and predictable interaction model between layers

Architectural Fit

This High-Level Diagram is the **foundational architectural overview** of the HBnB Evolution application. It establishes the system's core structure and the primary communication model that all other detailed diagrams (like Class and Sequence Diagrams) adhere to. It provides the essential context for understanding how the application's components are organized and interact at the broadest level, reinforcing the chosen N-tier design.

03

Business Logic Layer

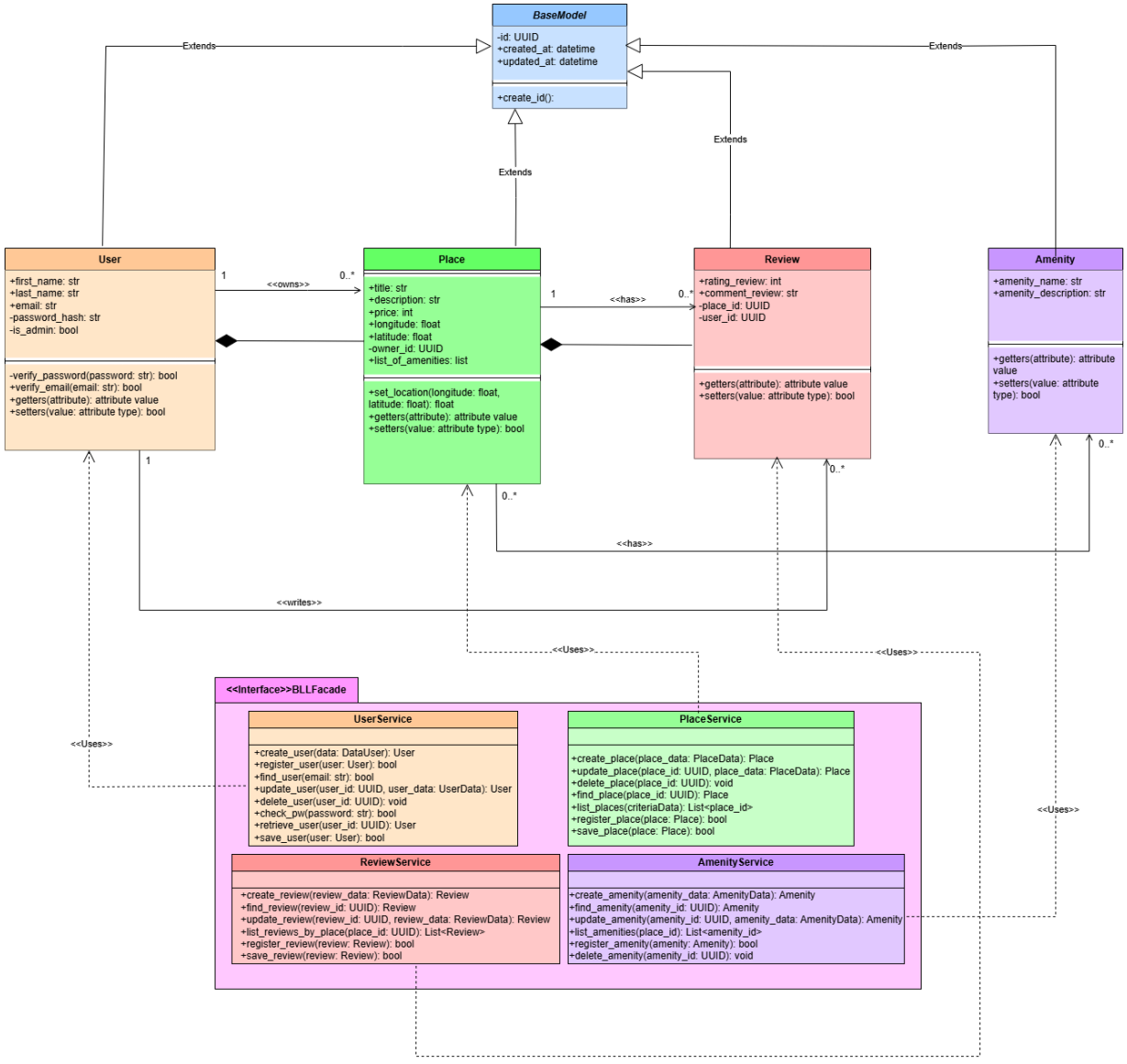

Fig. 2: HBnB Evolution class diagram. The parenthesis next to the attributes state the data type of that attribute. The parenthesis next to the methods state its arguments and their type following the format 'argument: argument type'. The return type of the methods is stated as: '-> return type'.

The Class Diagram illustrates the **static structure** of the HBnB Evolution application. It defines the core entities (User, Place, Amenity, Review) as blueprints, detailing their attributes, methods, and the relationships that govern how they interact within the system's business logic.

Key components

- **User:** Represents a registered application user with profile and authentication details
- **Place:** Represents a property listing, detailing its characteristics, owner, and associated amenities
- **Amenity:** Defines specific features or services available at a place
- **Review:** Represents user-generated comments and ratings for a place
- **BaseModel:** A foundational class providing common attributes (id, created_at, updated_at) and basic functionalities for all core entities, ensuring consistency and reusability
- **BLLFacade (UserService, PlaceService, ReviewService, AmenityService):** Represents the Business Logic Layer's entry point, encapsulating and orchestrating complex business operations for each entity

Design Decisions & Rationale

- **Inheritance from BaseModel:** All main entities inherit from BaseModel to centralize common attributes and methods (like unique ID generation and timestamp tracking), promoting DRY (Don't Repeat Yourself) principles and system-wide consistency for auditing
- **Clear Entity Relationships:** The diagram clearly defines one-to-many, many-to-many, and one-to-one relationships (e.g., User can own multiple Places, Place can have multiple Amenities, Review linked to one User and one Place), ensuring data integrity and correct object associations
- **BLL Facade for Business Logic:** The BLLFacade implements the Facade pattern, simplifying the interface to the complex business logic for the Presentation Layer and delegating data operations to the Persistence Layer, thus maintaining a clean separation of concerns
- **Attribute and Method Typing:** Explicitly stating data types for attributes and arguments/return types for methods enhances clarity, facilitates development, and enables better type checking
- **Password Hashing (password_hash):** Storing password hashes instead of plain text is a fundamental security decision to protect user credentials

Architectural Fit

This Class Diagram forms the **backbone of the Business Logic Layer**, defining the "what" of the system's data and operations. It provides the static context for the dynamic interactions illustrated in the Sequence Diagrams, showing the objects that are created, manipulated, and persisted. It directly underpins the layered architecture by defining the entities that move between layers and the services (within BLLFacade) that implement the core business logic, preparing the schema for the Persistence Layer.

04

API Interaction Flow

Description

This section presents a series of **sequence diagrams** that illustrate the flow of interactions between the components of the HBnB Evolution application in response to key API calls. These diagrams are designed to show how the system's three main layers—**Presentation**, **Business Logic**, and **Persistence**—collaborate to fulfill specific user requests.

The purpose of these diagrams is to:

- Make the flow of control and data across the layers explicit.
- Clarify how each layer contributes to handling API requests.
- Serve as a guide for implementing the corresponding endpoints and services in later development phases.

Four sequence diagrams will be presented next. Each one corresponds to a specific use case and demonstrates the dynamic behavior of the application, unlike the previously used diagrams (package diagram and class diagram) which represent the static structure of the application.

The four chosen API calls are:

- **User Registration** – A new user signs up for an account.
- **Place Creation** – A registered user creates a new place listing.
- **Review Submission** – A user submits a review for a place they visited.
- **Fetching a List of Places** – A user queries the system for available places based on specified criteria.

In each diagram, interactions begin with a call to an endpoint in the **Presentation Layer**, pass through the **Business Logic Layer**, and ultimately involve data operations in the **Persistence Layer**. The use of the **Facade pattern** is emphasized to illustrate the encapsulation of business logic and the simplification of layer-to-layer communication.

These diagrams not only help visualize runtime behavior but also reinforce architectural consistency and clarify responsibilities across components.

04.1

User Registration

The flow of information corresponding to a user registration request is presented in the following figure.

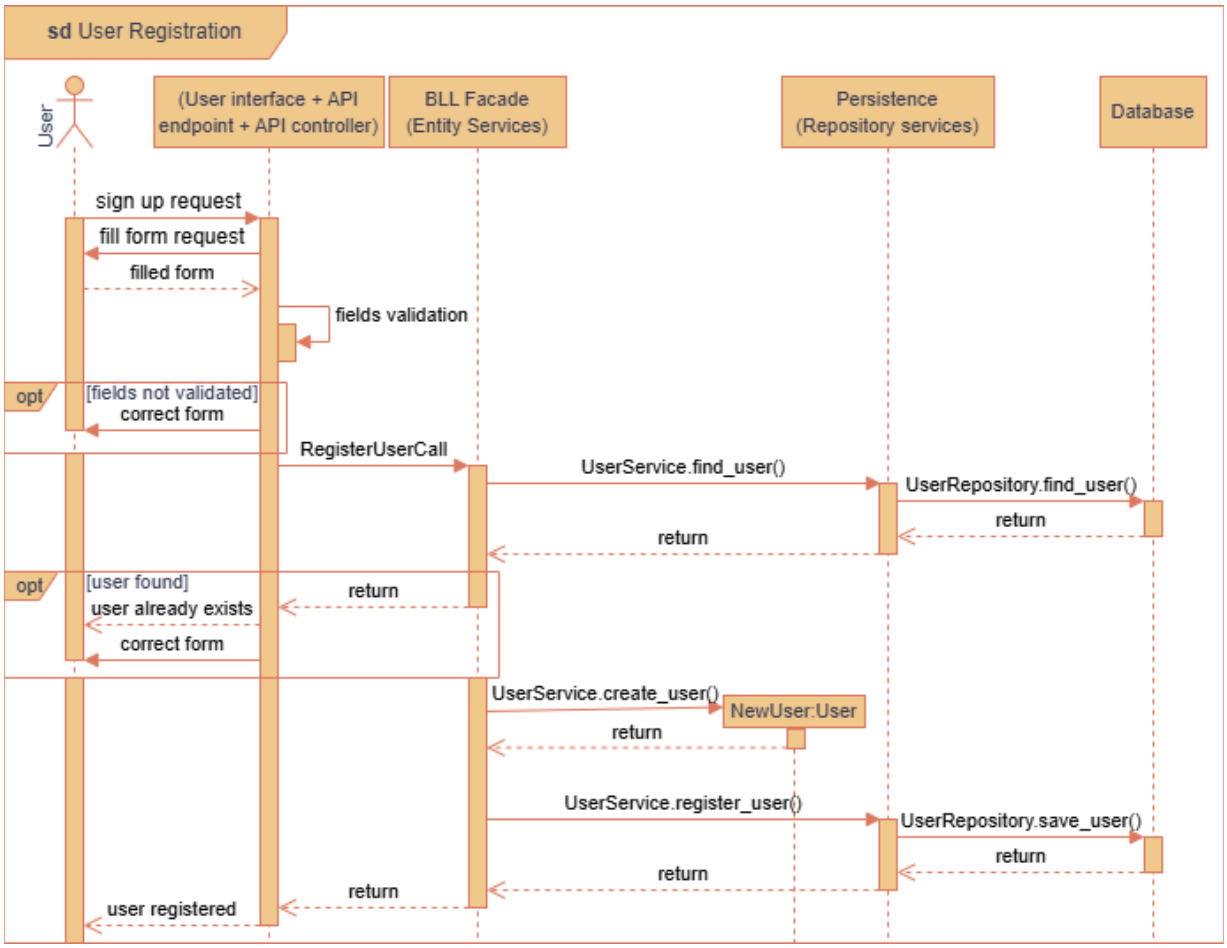

Fig. 3: user registration sequence diagram.

This diagram illustrates the dynamic flow for new user account registration, including data input, multi-level validation, checks for existing accounts, and secure user data persistence. It highlights successful registration as well as error handling for invalid input or duplicate accounts.

Key Components

- **User (Actor):** Initiates registration
- **User Interface + API Controller (Presentation Layer):** Handles user input, initial validation, and orchestrates requests to the BLL
- **BLL Facade (Business Logic Layer):** Orchestrates business rules (e.g., user existence check, user creation), ensures data consistency, and delegates to persistence

- **MyUser: User (Object):** Represents the in-memory user data being processed
- **Persistence Layer (Repository Services):** Manages direct database interactions for user search and saving
- **Database:** Stores user account information

Design Decisions & Rationale

- **Layered Architecture:** Promotes modularity and separation of concerns (Presentation, BLL, Persistence) for maintainability and scalability
- **Facade & Repository Patterns:** BLL Facade simplifies business logic access. Repository Pattern abstracts database interactions, enhancing testability and flexibility
- **Multi-Stage Validation:** Input validation (Presentation Layer) and business validation (BLL - e.g., user existence check) are explicitly modeled to provide early feedback and prevent invalid data from reaching the database
- **BLL Object Creation:** User object instantiation within the BLL (UserService.create_user()) ensures business rules govern object creation before persistence

Architectural Fit

This diagram is a core component of the application's user management. It defines the foundational process for onboarding new users, which is a prerequisite for subsequent application functionalities (e.g., login, place creation, review submission). It clearly demonstrates the division of responsibilities across the application's layered architecture, reflecting a robust N-tier design.

04.2

User Login

Besides user registration, all other use cases require being logged-in as a user. To simplify the diagrams of such other use cases we refer to the following user login diagram.

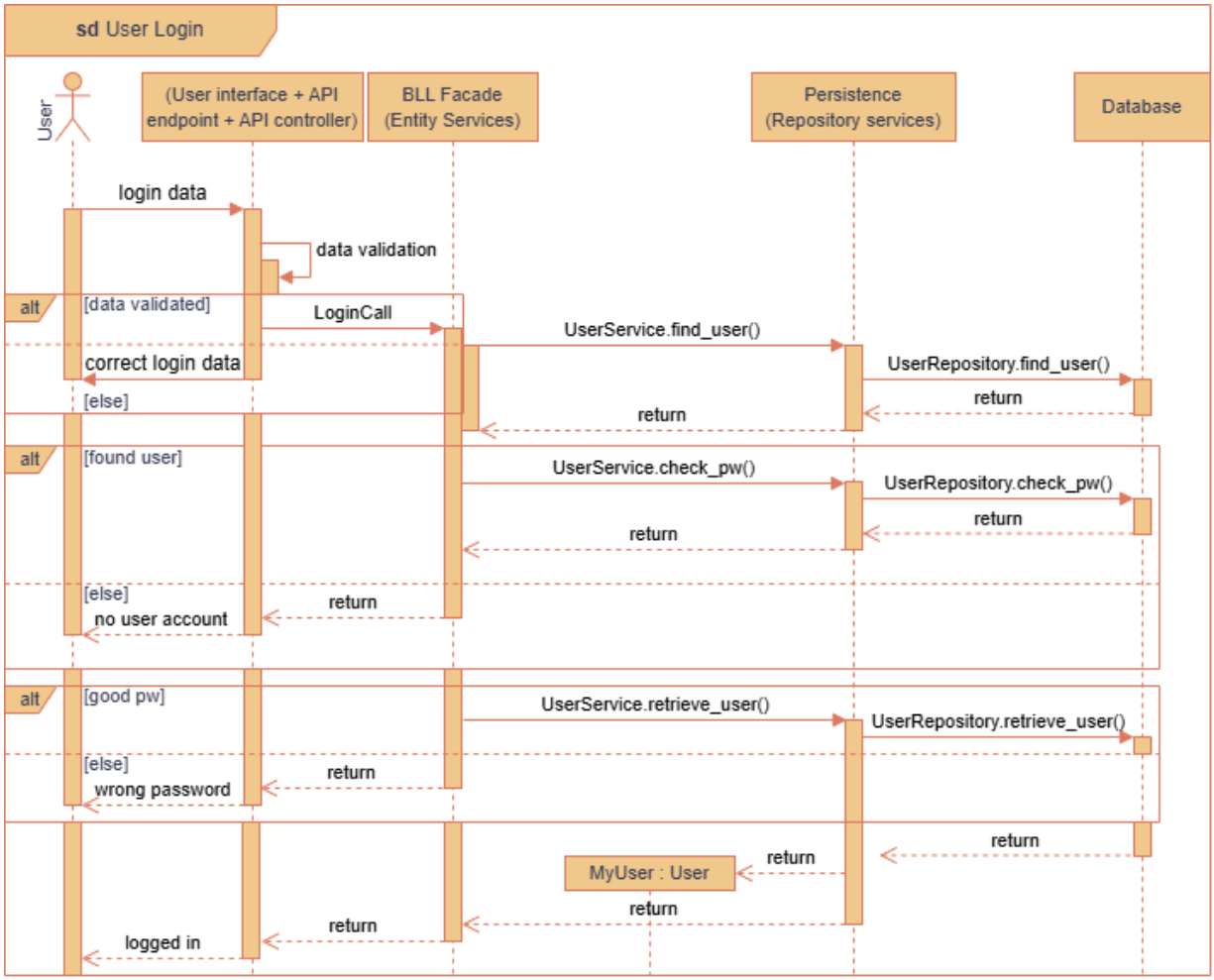

Fig. 4: user login sequence diagram.

This diagram illustrates the **dynamic flow of user authentication** within the HBnB Evolution application. It details the process of a user submitting credentials, the system verifying user existence and password validity, and the handling of both successful login and authentication failures (user not found, incorrect password).

Key components

- **User (Actor):** Initiates the login process.
- **User Interface + API Controller (Presentation Layer):** Handles user input (credentials), performs basic validation, and orchestrates requests to the BLL.
- **BLL Facade (Business Logic Layer):** Orchestrates the authentication logic, manages user verification, and delegates data retrieval to persistence.
- **MyUser : User (Object):** Represents the user instance retrieved upon successful authentication.
- **Persistence Layer (Repository Services):** Manages direct database interactions for finding user accounts.
- **Database:** Stores user account credentials and data.

Design Decisions & Rationale

- **Layered Architecture:** Promotes clear **separation of concerns** (Presentation, BLL, Persistence), enhancing modularity, maintainability, and scalability.
- **Facade & Repository Patterns:** The BLL Facade simplifies access to complex authentication logic, while the Repository Pattern abstracts database interactions, improving testability and flexibility.
- **Multi-Stage Authentication/Validation:** The diagram explicitly models two distinct validation steps (alt fragments): first, checking for user existence based on email, and second, verifying the provided password. This granular approach provides precise error feedback and enhances security.
- **Precondition for Actions:** The Ref UserLogin highlights that a successful execution of this flow is a prerequisite for subsequent user-specific operations (e.g., Place Creation, Review Submission).

Architectural Fit

This diagram is a **critical security and access control component** of the application. It defines the foundational authentication process that gates access to most protected functionalities. It clearly demonstrates how user credentials flow through the layered architecture to be validated against stored data, establishing the user's authenticated state necessary for other system interactions.

04.3

Place Creation

The process of creation of a Place performed by a User is depicted in the following diagram.

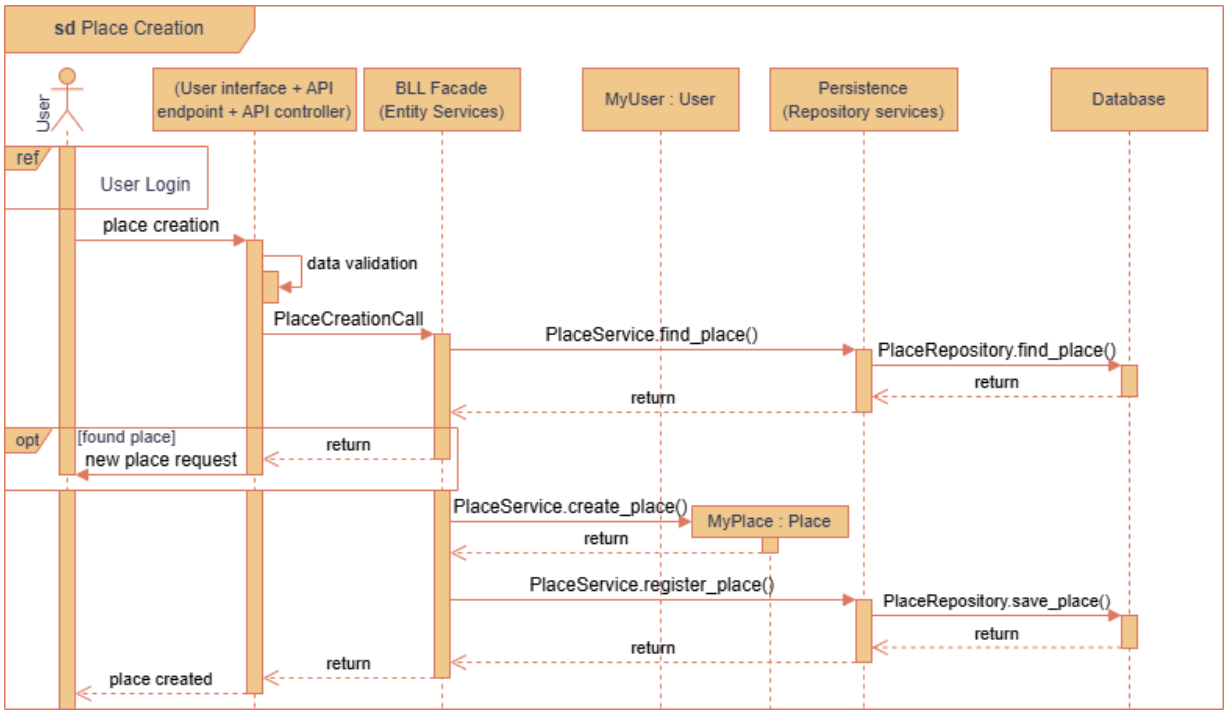

Fig. 5: place creation sequence diagram.

This diagram illustrates the dynamic flow for a registered and logged-in user to create a new place. It details the steps involving initial data submission, validation checks to ensure the place doesn't already exist, and the final persistence of the new place data.

Key Components

- **User (Actor):** The external entity initiating the place creation
- **User Interface + API Controller (Presentation Layer):** Handles user input, initial validation, and orchestrates requests to the BLL
- **BLL Facade (Business Logic Layer):** Orchestrates business rules (e.g., place existence check, place creation), ensures data consistency, and delegates to persistence
- **MyUser : User (Object):** Represents the logged-in user instance, implicitly indicating the user's authenticated status but not directly involved in the place creation messages
- **MyPlace : Place (Object):** Represents the in-memory place data being processed and eventually saved
- **Persistence Layer (Repository Services):** Manages direct database interactions for finding existing places and saving new ones
- **Database:** Stores place information

Design Decisions & Rationale

- **Layered Architecture:** Maintains clear separation of concerns (Presentation, BLL, Persistence) for modularity, maintainability, and scalability
- **Facade & Repository Patterns:** The BLL Facade simplifies business logic access, while the Repository Pattern abstracts database interactions, enhancing testability and flexibility
- **Pre-creation Existence Check (opt [found place]):** An explicit optional fragment is used to check if a place with similar details already exists before attempting creation. This prevents duplicates and provides immediate user feedback, stopping the process if a match is found
- **Object Instantiation within BLL:** The MyPlace object is instantiated within the BLL (PlaceService.create_place()). This places the responsibility of creating the core domain object within the business logic, where creation rules are best managed, before it's passed to the persistence layer for storage

Architectural Fit

This diagram is a crucial component within the application's core functionality of managing locations. It builds upon the Ref User Login process, highlighting that place creation requires an authenticated user. It demonstrates the effective division of responsibilities across the application's layered architecture, showing how user requests are processed through the Presentation, Business Logic, and Persistence layers to achieve data manipulation in the database.

04.4

Review Submission

The dynamic process of a User submitting a review for a place is illustrated in the following diagram.

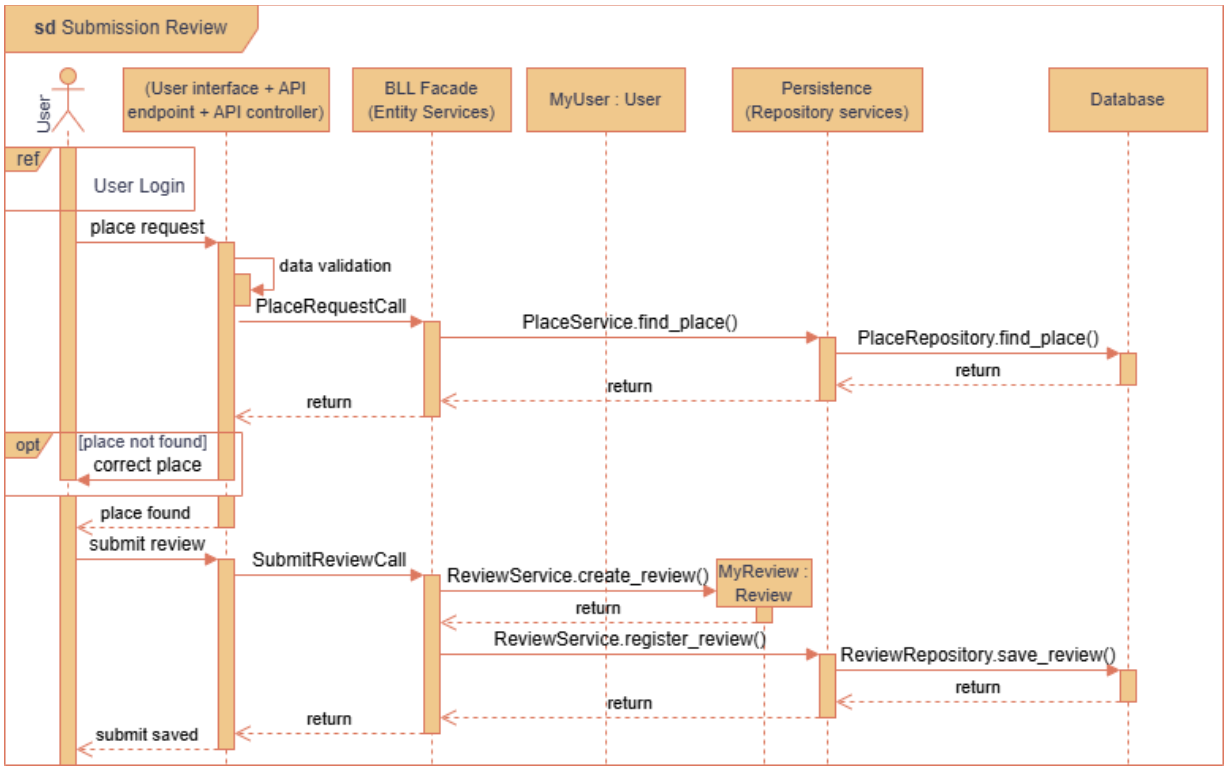

Fig. 6: review submission sequence diagram.

This diagram illustrates the dynamic flow for a registered and logged-in user to submit a review for a specific place. It covers the process of verifying the target place's existence, submitting review data, and ensuring the review's successful creation and persistence.

Key components

- **User (Actor):** Initiates the review submission
- **User Interface + API Controller (Presentation Layer):** Handles user input (place request, review data), initial validation, and forwards requests to the BLL
- **BLL Facade (Business Logic Layer):** Orchestrates business rules (e.g., finding the place, creating the review object), ensures data consistency, and delegates to persistence
- **MyUser : User (Object):** Represents the logged-in user instance, its ID is used to associate the review
- **MyReview : Review (Object):** Represents the in-memory review data being processed and eventually saved
- **Persistence Layer (Repository Services):** Manages direct database interactions for finding places and saving reviews

- **Database:** Stores place and review information

Design Decisions & Rationale

- **Layered Architecture:** Maintains clear separation of concerns (Presentation, BLL, Persistence) for modularity, maintainability, and scalability
- **Facade & Repository Patterns:** The BLL Facade simplifies business logic access. The Repository Pattern abstracts database interactions, enhancing testability and flexibility
- **Pre-submission Place Verification (opt [place not found]):** An explicit optional fragment checks if the place (to which the review is linked) exists. This prevents orphaned reviews and guides the user if the place is invalid
- **BLL Object Creation:** MyReview object instantiation within the BLL (ReviewService.create_review()) ensures business rules govern object creation before persistence, including associating it with a place_id and MyUser.id

Architectural Fit

This diagram is a core component within the application's place management and user interaction features. It builds upon Ref User Login, demonstrating that only authenticated users can submit reviews. It showcases the full cycle of interaction across Presentation, Business Logic, and Persistence layers, ensuring the integrity and proper storage of user-generated content associated with specific places.

04.5

Fetching a List of Places

The dynamic process of a User requesting and retrieving a list of places based on specific criteria is depicted in the following diagram.

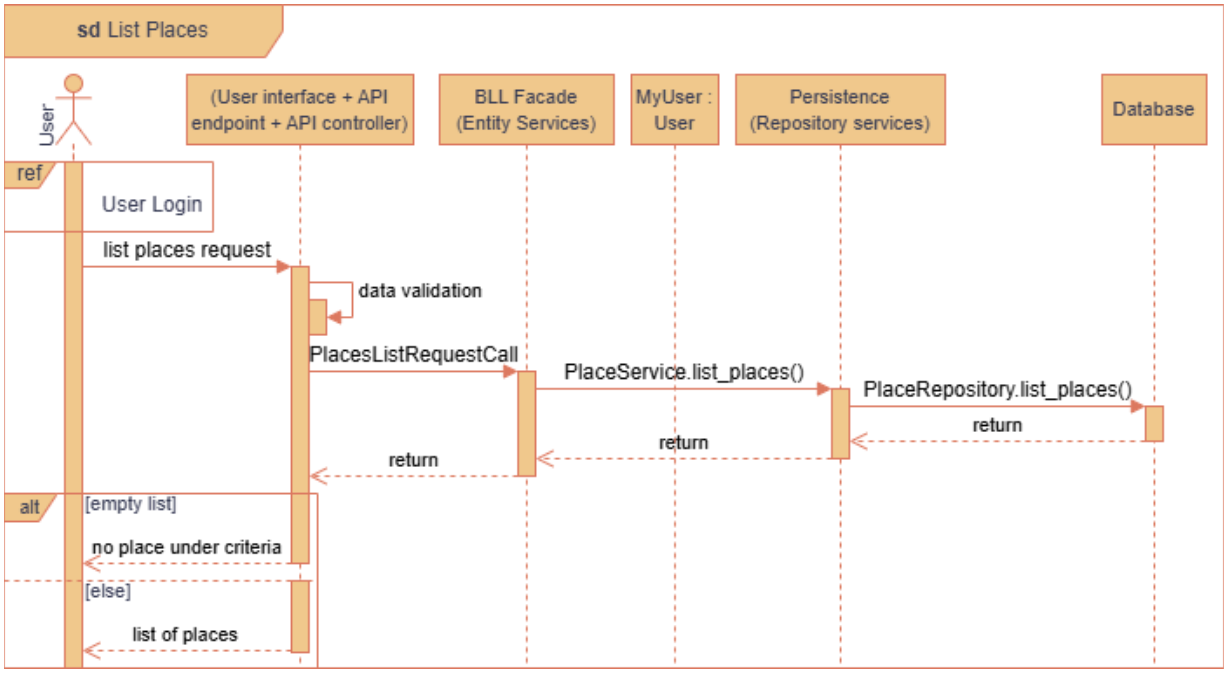

Fig. 7: list of places sequence diagram.

This diagram illustrates the dynamic flow for a registered and logged-in user to request and display a list of places based on specific criteria. It details the process of fetching, validating, and presenting the list, including handling cases where no places match the criteria.

Key components

- **User (Actor):** Initiates the request for a list of places
- **User Interface + API Controller (Presentation Layer):** Handles user input (criteria), initial validation, and forwards requests to the BLL
- **BLL Facade (Business Logic Layer):** Orchestrates the process of retrieving places, applies business logic related to criteria, and delegates to persistence
- **MyUser : User (Object):** Represents the logged-in user, implicitly indicating the user's authenticated status.
- **Persistence Layer (Repository Services):** Manages direct database interactions for finding places based on criteria.
- **Database:** Stores place information

Design Decisions & Rationale

- **Layered Architecture:** Maintains clear separation of concerns (Presentation, BLL, Persistence) for modularity, maintainability, and scalability
- **Facade & Repository Patterns:** The BLL Facade simplifies business logic access for place retrieval. The Repository Pattern abstracts database interactions, enhancing testability and flexibility by decoupling the BLL from specific DB access details
- **Criteria-Based Retrieval:** The system supports fetching places based on user-defined Criteria Data, allowing for flexible search functionalities
- **Alternative Fragment for Results (alt):** The explicit alt fragment for [empty list] vs. [else] ensures proper handling and user feedback whether places are found or not, providing a clear path for each outcome

Architectural Fit

This diagram is a fundamental part of the application's core place Browse functionality. It builds upon Ref User Login, indicating that accessing place lists often requires an authenticated user. It clearly demonstrates the division of responsibilities across the application's layered architecture, showing how user queries are processed from the Presentation Layer through the Business Logic and Persistence Layers to retrieve and display relevant data from the database.