Table of Contents

1. **Introduction**
    - 1.1 Purpose of the Document
    - 1.2 Context
    - 1.3 Target Audience
    - 1.4 Document Structure
2. **Overall Architecture**
    - 2.1 Three-Layer Architecture
    - 2.2 Package Diagram
    - 2.3 Facade Pattern
    - 2.4 Repository Pattern
    - 2.5 Layer Details
    - 2.5.1 Presentation Layer
    - 2.5.2 Business Logic Layer
    - 2.5.3 Persistence Layer
3. **Business Logic**
    - 3.1 Role
    - 3.2 Class Diagram
    - 3.3 Relationships
    - 3.4 Business Constraints
    - 3.5 Justification
    - 3.6 UML Relationship Legend
4. **Sequence Diagrams**
    - 4.1 API Use Cases
    - 4.2 Creating a Place
    - 4.3 Retrieving a Review
    - 4.4 Fetching a List of Places
    - 4.5 Call Chain
        - Legend – UML Sequence Diagram
5. **Technical Justifications**
    - 5.1 SOLID Principles
    - 5.2 Technical Choices
    - 5.3 System Qualities
    - 5.4 Challenges Faced
6. **Conclusion**
    - 6.1 Summary

6.2 Next Steps
6.3 Document Usefulness

7. **Appendices**
7.1 Glossary
7.2 GitHub Repository

1. Introduction

1.1 Purpose of the Document

This document provides a comprehensive technical overview of the HBnB project. It compiles architectural choices, UML diagrams, and technical foundations useful for development, review, and system maintenance.

1.2 Context

HBnB is an application inspired by Airbnb, developed as part of the Holberton School curriculum. It offers both a command-line interface and a web interface, and is based on a three-layer architecture.

1.3 Target Audience

This document is intended for developers, contributors, and reviewers who wish to understand the internal mechanisms of the application.

1.4 Document Structure

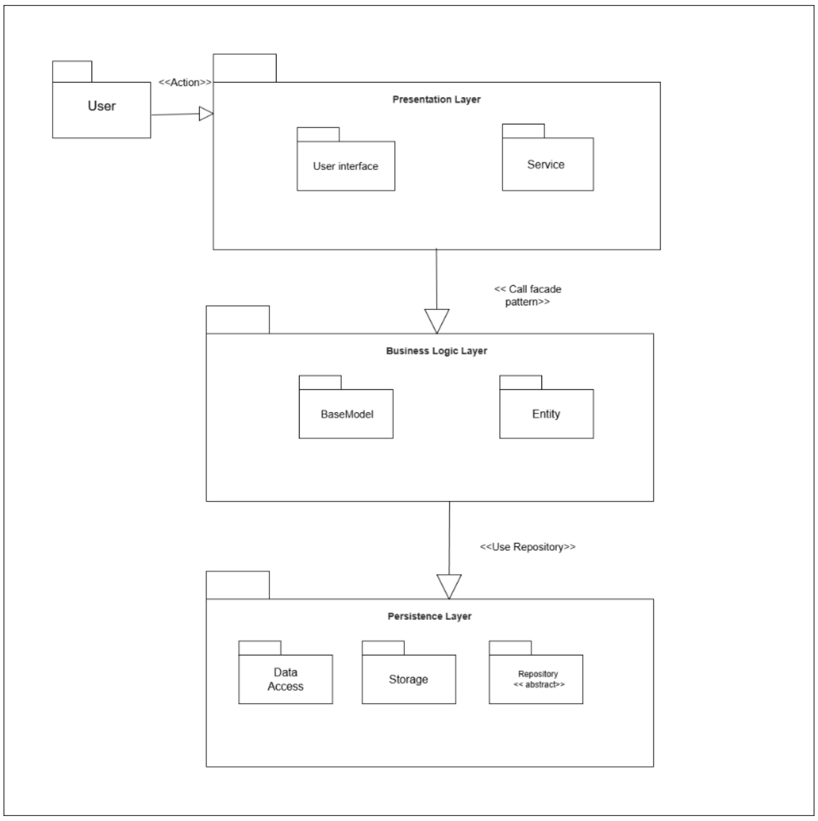
The document is organized into 7 sections: system architecture, business logic, sequence diagrams, technical choices, glossary, and appendices.

2. **General Architecture**

2.1 Three-Layer Architecture

- **Presentation:** User interface and exposed services.
- **Business Logic:** Business rules and coordination of entities.
- **Data (Persistence):** Data storage, reading, and writing.

2.2 Package Diagram:



See the Package Diagram, illustrating dependencies between modules.

2.3 Façade Pattern

The Façade centralizes calls from external layers to the business logic, simplifying exchanges and hiding complexity.

2.4 Repository Pattern

The Repository serves as an interface between the business logic and persistence. It isolates business logic from storage details. It is declared as an abstract interface (<<abstract>>) to define a contract that each concrete implementation must follow.

2.5 Layer Details

Presentation

- **UI:** Command-line (cmd) or web (Flask).
- **Service:** Processes requests, calls the Façade, and adapts responses.

Business Logic

- **BaseModel:** Provides a common foundation for entities. It contains generic methods (save(), to_dict(), etc.) and manages metadata (timestamps, unique ID)

- **Business Entities:** User, Place, Review, Amenity, which extend BaseModel to include their own logic.

BaseModel plays a fundamental role in the business architecture. It represents an abstraction layer common to all entities without carrying domain-specific business logic. It enables the factoring of recurring behaviors (IDs, serialization, timestamps), but does not correspond to a business object in itself. Conversely, User, Place, etc., are concrete business entities that extend BaseModel to add specific business rules and validations. Thus, BaseModel structures the functional base of each entity without embodying its own business logic.

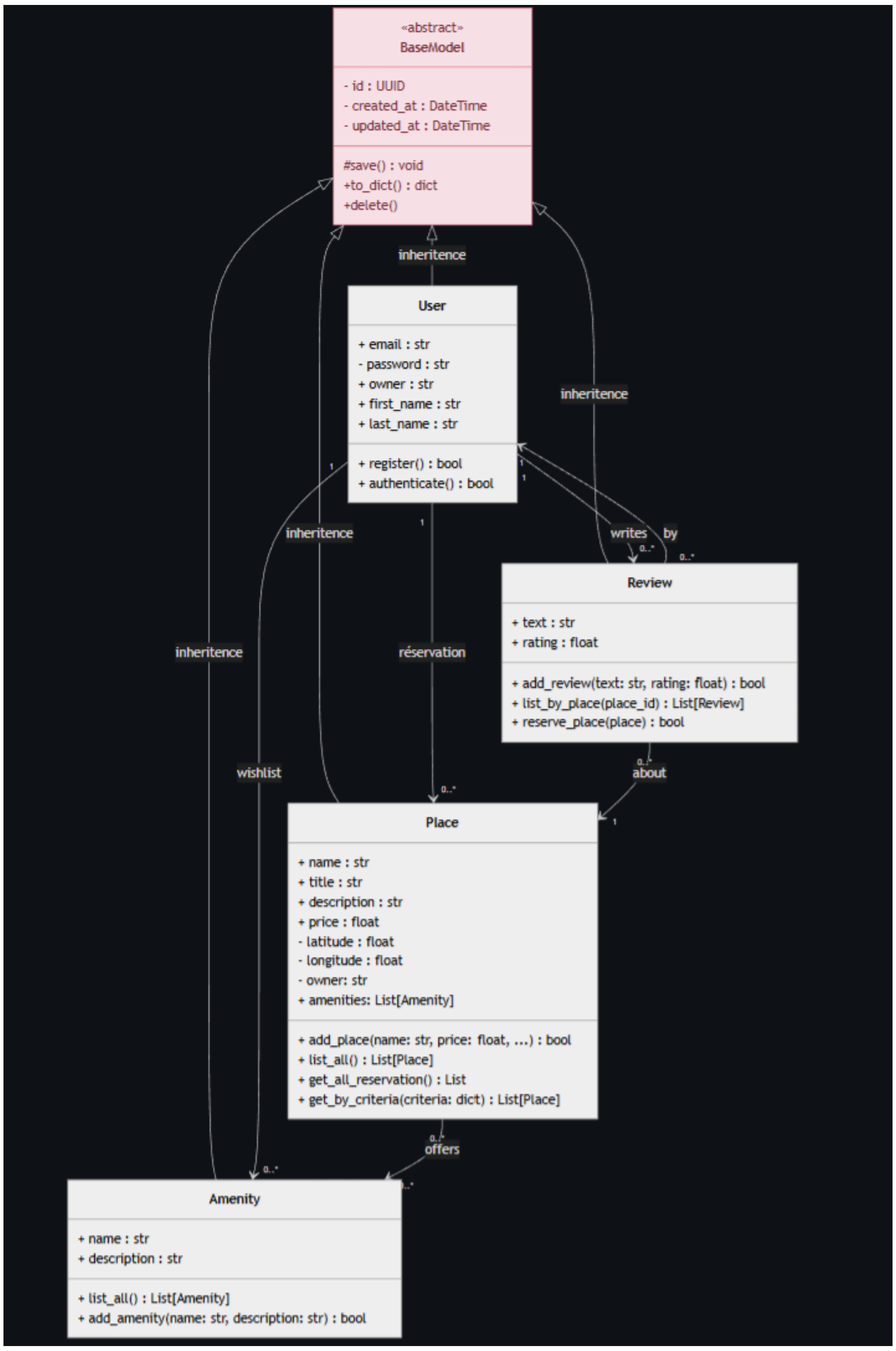
Persistence

- **Repository (<<abstract>>):** Defines a generic interface (e.g., UserRepository) for accessing entities.

The <<abstract>> stereotype indicates that this class is not instantiated directly. The Repository acts as a clear business interface, defining a contract for retrieving, creating, or updating entities. This abstraction allows business logic to be completely decoupled from storage logic. The backend (file, SQL database, etc.) can thus be substituted without impacting business code.

- **Data Access Layer (DAL):** Concretely implements Repository methods to interact with a given storage type.
- **Storage:** Final persistence layer (e.g., JSON file, SQL database, etc.). This layer transforms business objects into an appropriate storage format (JSON, SQL, etc.), and vice versa. It handles serialization, deserialization, queries, or writing to disk or database.

3. Business Logic



3.1 Role

Apply business rules, validate data, and coordinate entities.

3.2 Class Diagram

The class diagram below represents the main classes used in the **Business Logic Layer** of the application. This diagram follows the **UML (Unified Modeling Language)** conventions and illustrates the static structure of the system.

Purpose

This diagram shows:

- The main classes and their relationships (inheritance, composition, association).
- Attributes and key methods for each class.
- Visibility of attributes/methods (+ public, - private, # protected).
- Inheritance hierarchies and dependencies between entities.

Description of Key Classes

- **BaseModel**
    - Common parent for all models.
    - Handles id, created_at, updated_at, and basic save() and to_dict() methods.
- **User**
    - Inherits from BaseModel.
    - Represents a system user.
    - Attributes: email, password, first_name, last_name.
- **Place**
    - Inherits from BaseModel.
    - Represents a place to be rented.
    - Attributes: name, description, number_rooms, latitude, longitude, price_by_night, etc.
    - Has a many-to-many relationship with **Amenity**.
- **Amenity**
    - Inherits from BaseModel.
    - Represents an extra feature for a place (e.g., Wi-Fi, Pool).
    - Attribute: name.
- **Review**
    - Inherits from BaseModel.
    - Linked to both a User and a Place.

- Attributes: text.

3.3 Relationships

- User 1 → * Place
- User 1 → * Review
- Place * → * Amenity
- Place * → * Review

3.4 Business Constraints

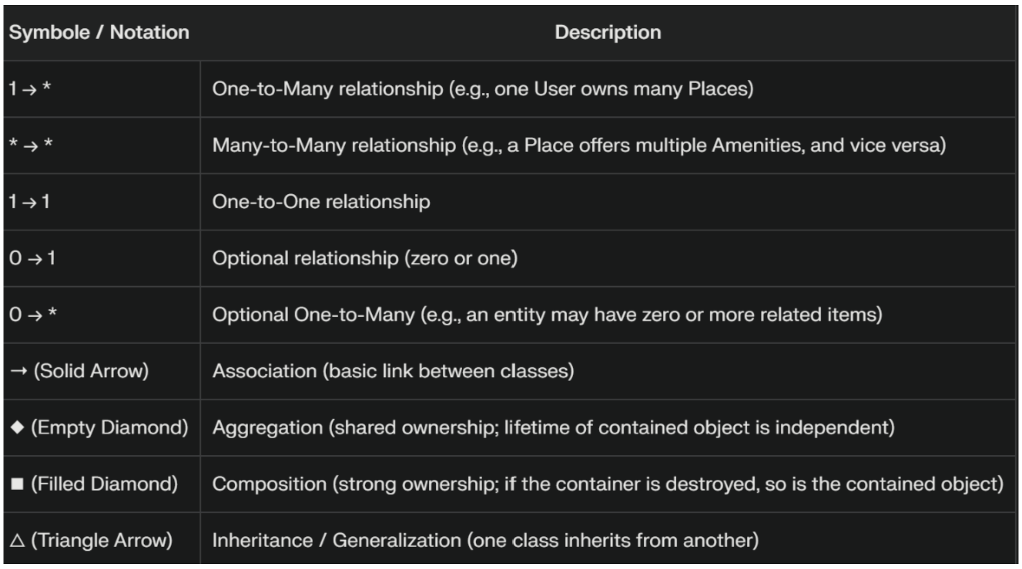
- Unique emails
- Only users who have booked can leave a review
- A place must have a price, a location, and an owner

3.5 Justification

- Isolation of persistence
- Centralized validation
- Relationships modeled via collections or association tables

3.6 Légende des relations UML

To interpret the UML class diagrams, here's a quick legend for the **relationship arrows**:



Symbole / Notation	Description
1 → *	One-to-Many relationship (e.g., one User owns many Places)
* → *	Many-to-Many relationship (e.g., a Place offers multiple Amenities, and vice versa)
1 → 1	One-to-One relationship
0 → 1	Optional relationship (zero or one)
0 → *	Optional One-to-Many (e.g., an entity may have zero or more related items)
→ (Solid Arrow)	Association (basic link between classes)
◆ (Empty Diamond)	Aggregation (shared ownership; lifetime of contained object is independent)
■ (Filled Diamond)	Composition (strong ownership; if the container is destroyed, so is the contained object)
△ (Triangle Arrow)	Inheritance / Generalization (one class inherits from another)

4. Sequence Diagrams

Purpose of Using a Sequence Diagram in the HBnB Project

The **sequence diagram** is a crucial UML tool used to **visualize the dynamic behavior of the system**. In the context of the HBnB project, it shows how different components (user interface, business logic, and data layer) **interact over time** to complete a specific action.

Objective

The primary goal of a sequence diagram is to **illustrate the flow of messages** between system elements during the execution of a particular use case. For instance, when a user creates a new Place object, multiple layers are involved:

4.1 API Use Cases

https://github.com/Ravou/holbertonschool-hbnb/blob/main/Part1/sequence_user.md

4.2 Creating a Place

https://github.com/Ravou/holbertonschool-hbnb/blob/main/Part1/sequence_place.md

4.3 Retrieving a Review

https://github.com/Ravou/holbertonschool-hbnb/blob/main/Part1/sequence_review.md

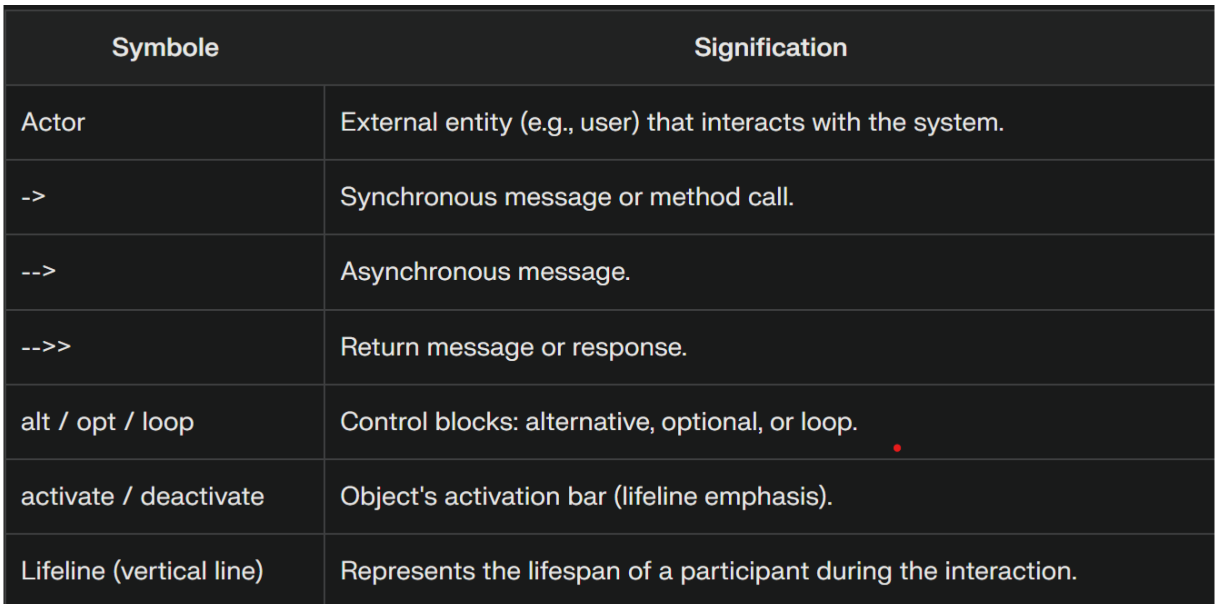
4.4 Fetching a List of Places

https://github.com/Ravou/holbertonschool-hbnb/blob/main/Part1/sequence_place.md

4.5 Call Chain

UI → Service → Façade → Business Logic → Repository → DAL → Storage

Legend - UML Sequence Diagram



Symbole	Signification
Actor	External entity (e.g., user) that interacts with the system.
->	Synchronous message or method call.
-->	Asynchronous message.
-->>	Return message or response.
alt / opt / loop	Control blocks: alternative, optional, or loop.
activate / deactivate	Object's activation bar (lifeline emphasis).
Lifeline (vertical line)	Represents the lifespan of a participant during the interaction.

5. Technical Justifications

5.1 Compliance with SOLID Principles

- **S:** Single Responsibility Principle
- **O:** Open/Closed Principle (Extensible via interfaces)
- **L:** Liskov Substitution Principle
- **I:** Interface Segregation Principle (Specific interfaces per type)
- **D:** Dependency Injection (via Façade / Repository)

5.2 Technical Choices

- Strong decoupling to facilitate testing and maintenance
- Inspired by backend best practices

5.3 System Qualities

- Testability by modules
- Replaceable Façade
- Interchangeable components

5.4 Challenges Encountered

- Validation moved to business logic
- Multiple relationships managed via intermediate structures

6. Conclusion

6.1 Summary

Modular three-layer architecture with Façade and Repository patterns. Focus on readability, maintainability, and scalability.

6.2 Next Steps

- Linking frontend and REST API
- Unit and integration test coverage
- Performance improvements

6.3 Usefulness of the Document

Reference for development, code review, or onboarding.

7. Appendices

7.1 Glossary

- **UML (Unified Modeling Language):**
  Standard graphical modeling language used to represent the components, behaviors, and structures of a software system.
- **Façade:**
  Simplified interface that groups access to a set of complex functionalities. It hides the internal complexity of a subsystem.
- **Repository:**
  Software component that centralizes access to data (database, files, etc.) in a layered architecture, separating business logic from persistence logic.
- **Package Diagram:**
  Represents the modular organization of a system by grouping classes or components into logical packages (or modules). It allows visualization of dependencies between the different layers or modules of the project.
- **Class Diagram (Business Logic):**
  Shows classes, their attributes, methods, and relationships (association,

inheritance, composition) between them. Here, it is focused on the business logic layer of the system.

- **Sequence Diagram:**
  Represents interactions between objects or components over time. It illustrates the chronology of calls (messages) between system actors and internal objects to perform a specific feature.

7.4 GitHub Repository

- Project source code:
  https://github.com/Ravou/holbertonschool-hbnb.git