Marivellisse Garcia Lebron

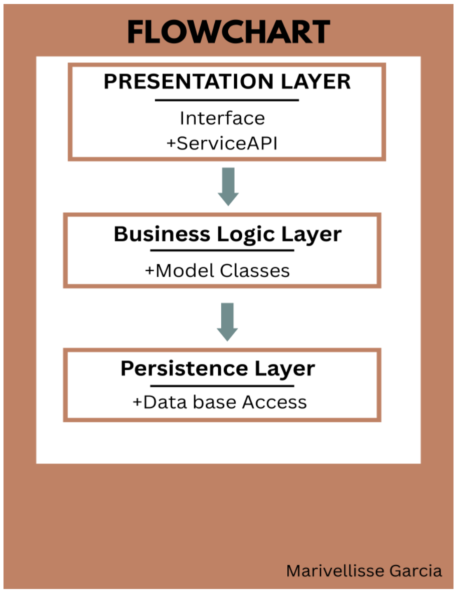
HBnb-UML

Introduction

This technical document serves as a comprehensive blueprint for the HBnB Evolution project, a simplified Airbnb-like web application designed to manage users, properties, bookings, and reviews. The project follows a modular, layered architecture that promotes scalability, maintainability, and clarity in design.

The primary purpose of this document is to guide the development team through the system’s architecture, class structure, and component interactions. It compiles all major diagrams—including the high-level package diagram, detailed class diagram for the business logic layer, and sequence diagrams for API interactions—along with explanatory notes that justify design choices and outline data flow throughout the system.

By offering a structured and visual representation of the system, this document supports both current implementation efforts and future development phases. It ensures that all team members share a consistent understanding of the system’s components, responsibilities, and design rationale, thereby minimizing miscommunication and enhancing development efficiency.



Task 0-Layered Architecture of the HBnB Evolution System

1. PresentationLayer

- **Stereotype**: <<Interface>>

This indicates that the PresentationLayer exposes an interface, not a concrete class.

- **Member**: +ServiceAPI

This is the public-facing interface used by clients (such as web or mobile apps) to interact with the system.

- **Relationship**:

○ **Arrow to BusinessLogicLayer** labeled **"Facade Pattern"**

This means the Presentation Layer acts as a *facade*, simplifying interaction with the complex logic underneath.

2. BusinessLogicLayer

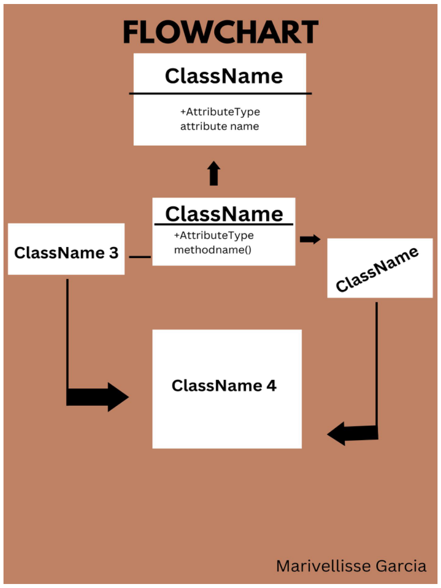
- **Member**: +ModelClasses
  This represents the domain logic or business rules—like user management, booking, review processing, etc.
- **Role**: Core of the application; processes data coming from the presentation layer before accessing or modifying stored data.
- **Relationship**:
  - **Arrow to PersistenceLayer** labeled **"Database Operations"**
    Indicates that the business logic delegates all data storage/retrieval operations to the persistence layer.

3. PersistenceLayer

- **Member**: +DatabaseAccess
  This is responsible for direct interaction with the database. Could include ORM models, SQL queries, or file access logic.

Relationships Summary

- PresentationLayer --> BusinessLogicLayer: **Facade Pattern**
- BusinessLogicLayer --> PersistenceLayer: **Database Operations**



Task 1 Detailed Class Diagram for Business Logic Layer

classDiagram

This line indicates that the diagram being defined is a **UML class diagram**.

class ClassName

This defines a **class** named ClassName with:

Attributes & Methods

- +AttributeType attributeName:

A **public attribute** called attributeName of type AttributeType.

- +MethodType methodName():

A **public method** named methodName that returns MethodType.

ClassName1 --|> ClassName2: Inheritance

This shows an **inheritance relationship**:

- ClassName1 is a **child** (subclass) of ClassName2, inheriting its attributes and methods.
- The arrow points **from child to parent** (--|>).
- Label: Inheritance (optional for understanding but describes the relationship).

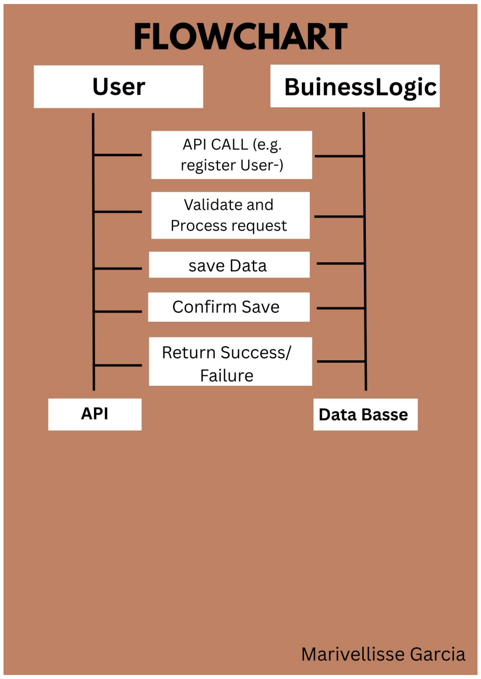
ClassName3 o-- ClassName : Composition

This denotes a **composition relationship**:

- ClassName3 is **composed of** ClassName, meaning ClassName3 has a **strong ownership** of ClassName.
- Represented with a **filled diamond** at ClassName3.
- If ClassName3 is destroyed, so is ClassName.

ClassName4 --> ClassName : Association

Task 2 – Sequence Diagrams for API Calls



This diagram represents the interaction flow between different components of the system when a user initiates a request (e.g., user registration).

Participants:

- **User**: The end user who initiates the interaction.
- **API**: The entry point that receives external requests.
- **BusinessLogic**: The business logic layer that validates and processes the request.
- **Database**: The storage system responsible for saving the data.

Process Flow:

1. **User → API**
   API Call (e.g., Register User)
   The user sends a request to the system, such as registering an account.
2. **API → BusinessLogic**
   Validate and Process Request
   The API forwards the request to the business logic to validate and process it.
3. **BusinessLogic → Database**
   Save Data
   The business logic layer requests to save the data in the database.
4. **Database → BusinessLogic**
   Confirm Save
   The database confirms that the data has been successfully saved.
5. **BusinessLogic → API**
   Return Response
   The business logic layer sends the result back to the API.
6. **API → User**
   Return Success/Failure
   The system returns a final response to the user indicating whether the operation was successful or failed.
