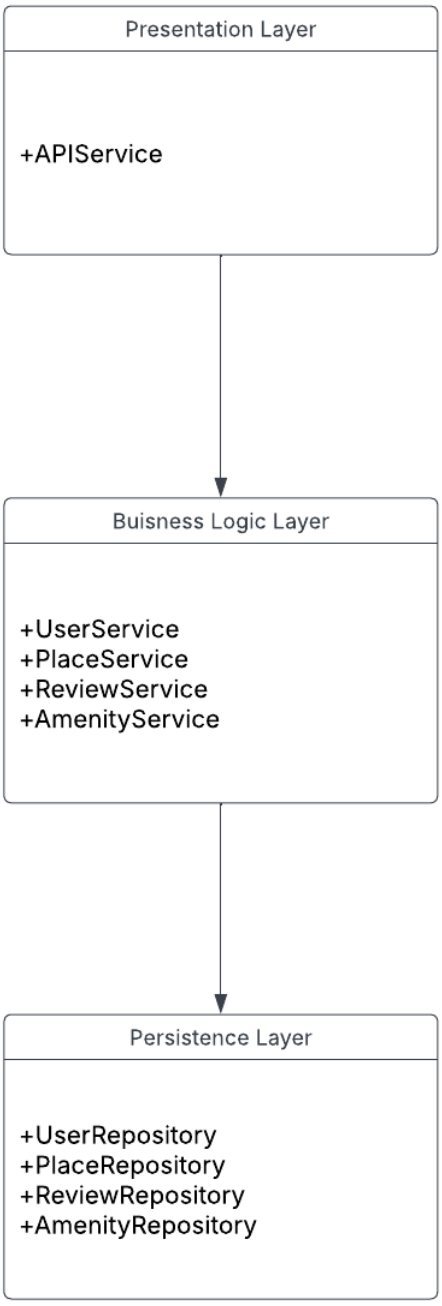
HBNB Evolution – Part 1 : TECHNICAL DOCUMENTATION

In this documentation you will find the architecture and simplified version of an Airbnb-like application, called HBnB. This application is an online platform for booking accommodation.

1- HIGH-LEVEL PACKAGE DIAGRAM :



This package diagram represents a three-layer architecture for an Airbnb-like app, organized to clearly separate responsibilities.

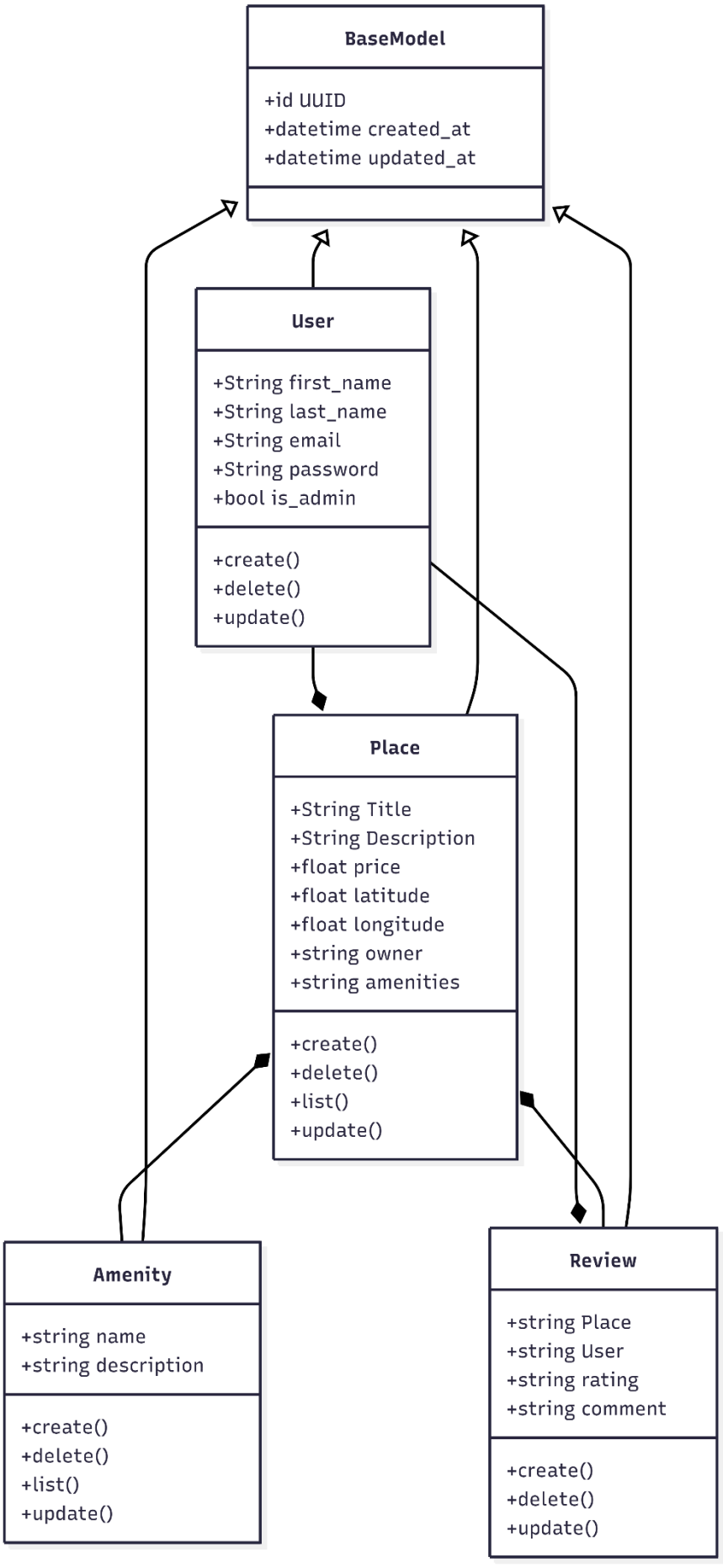
The Presentation layer contains the entry point for the application (APIService). It receives requests from users and forwards them to the business layer.

The Business Logic layer includes the core services (UserService, PlaceService, ReviewService, AmenityService). It contains the management rules and data processing: management of users, accommodations, reviews and equipment.

The Persistence layer is responsible for accessing the data through repositories (UserRepository, PlaceRepository, ReviewRepository, AmenityRepository).

It communicates with the database to store and retrieve information. The arrows show that each layer depends solely on the bottom layer, which improves organization, maintainability, and separation of responsibilities.

2- DETAILED CLASS DIAGRAM FOR BUSINESS LOGIC LAYER :


BaseModel
+id UUID
+datetime created_at
+datetime updated_at
User
+String first_name
+String last_name
+String email
+String password
+bool is_admin
+create()
+delete()
+update()
Place
+String Title
+String Description
+float price
+float latitude
+float longitude
+string owner
+string amenities
+create()
+delete()
+list()
+update()
Amenity
+string name
+string description
+create()
+delete()
+list()
+update()
Review
+string Place
+string User
+string rating
+string comment
+create()
+delete()
+update()

This class diagram represents the main entities of the application and their relationships.

The BaseModel class is the parent class that is common to all the others. It contains the attributes shared by each object: a unique identifier (id) as well as the creation and update dates (created_at, updated_at)

The User, Place, Review, and Amenity classes inherit from this class to avoid duplication.

The User class represents a user of the application. It contains their personal information (first_name, last_name, email, password) as well as a is_admin indicator. It is used to manage basic operations such as creation, modification and deletion.

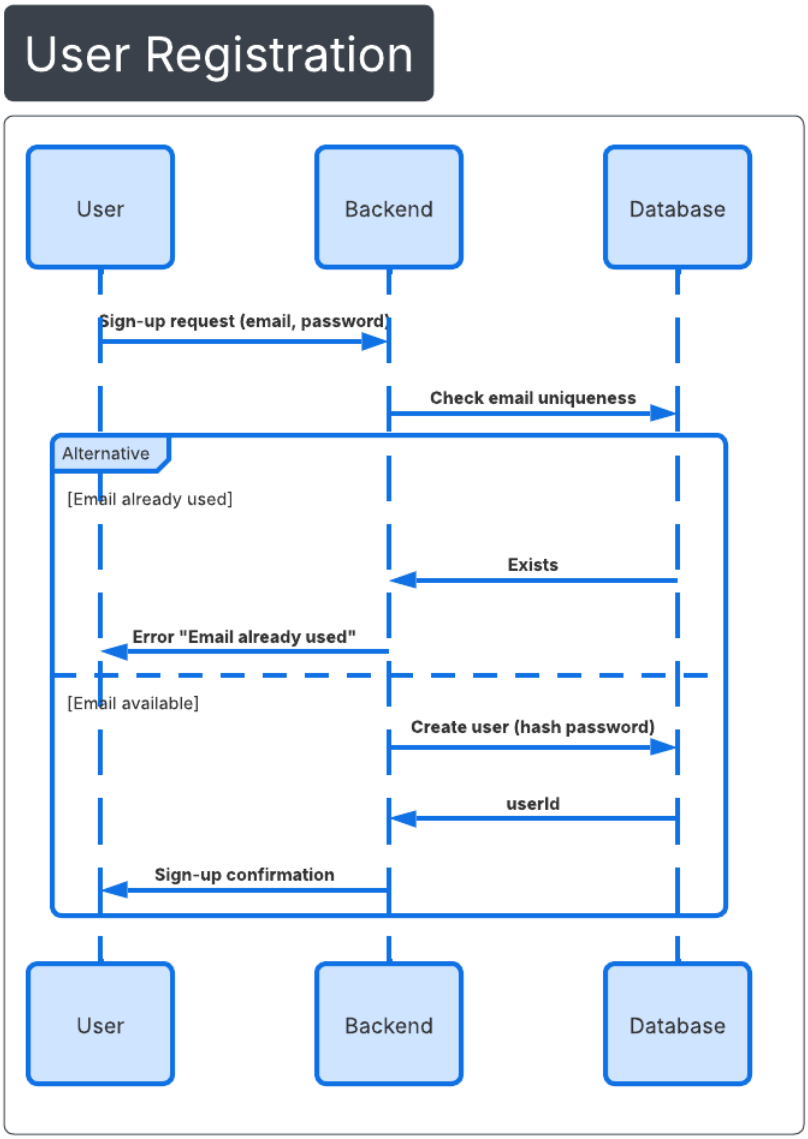
The Place class represents a listing published on the platform. It contains the main information such as the title, description, price, location (latitude, longitude), owner and amenities. It is linked to a user (the owner) and can be associated with multiple reviews and equipment.

The Review class represents a review left by a user on a listing. It contains the reference to the place and the user, a rating and a comment. A review is therefore associated with a single user and a single dwelling.

The Amenity class represents the amenities available in a home (e.g., Wi-Fi, swimming pool). It contains a name and description, and can be associated with multiple listings. The reports show that : all entities inherit from BaseModel; A user can own multiple dwellings; a property can have several reviews; A dwelling can be linked to several facilities.

3- SEQUENCE DIAGRAMS FOR API CALLS :

3.1 User Registration :

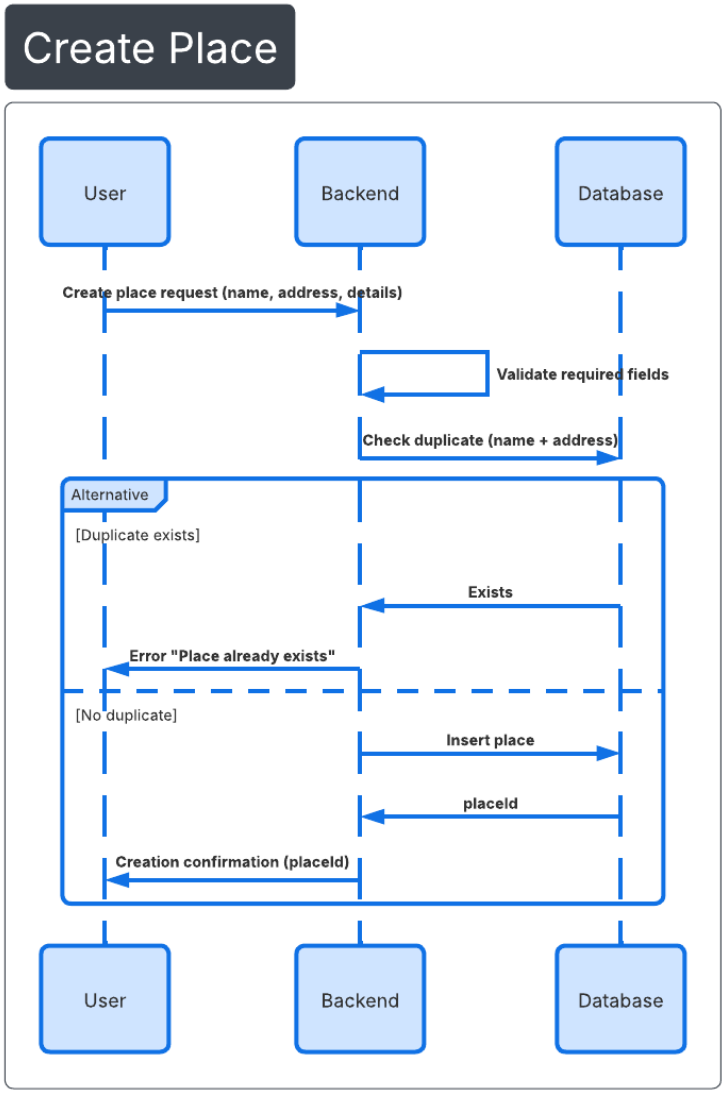


This sequence diagram illustrates the process of registering a user in the application and the interactions between the different elements of the system: User, Backend, and Database.
The process begins when the user sends a registration request to the backend by providing their email and password.
The backend receives this request and checks with the database to see if the email is already in use.
Two cases are then possible :

- If the email already exists, the database reports it to the backend, which returns an error message to the user indicating that the address is already in use.
- If the email is available, the backend creates a new user by securing the password.

The information is then stored in the database, which returns a user ID (userId). Finally, the backend confirms to the user that the registration has been completed successfully.

3.2 Place Creation :

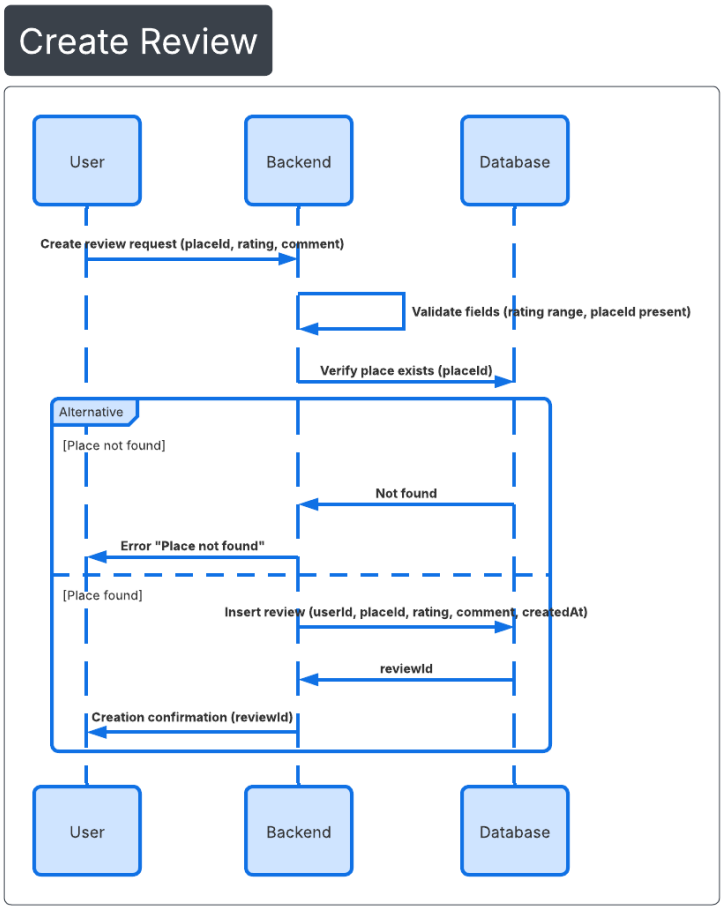


This diagram illustrates the process of creating a place (place) in an Airbnb-like app.
The diagram shows the interaction between three actors : the user, the backend server, and the database when creating a new place.

Initiating the request : The user submits a creation request with the location information (name, address, details).
Data validation : The backend first performs validation of the required fields to ensure that the data is complete and correct.
Duplicate checking : The backend queries the database to check if a place with the same name and address already exists. Two possible scenarios :

- Scenario A - Duplicate Detected : Database returns "Exists". The backend returns a "Place already exists" error to the user. The process stops.
- Scenario B - No duplicates : The backend proceeds to insert the place into the database. The database confirms the insertion with an identifier (placeId). The backend forwards the creation confirmation to the user

3.3 Review Submission :

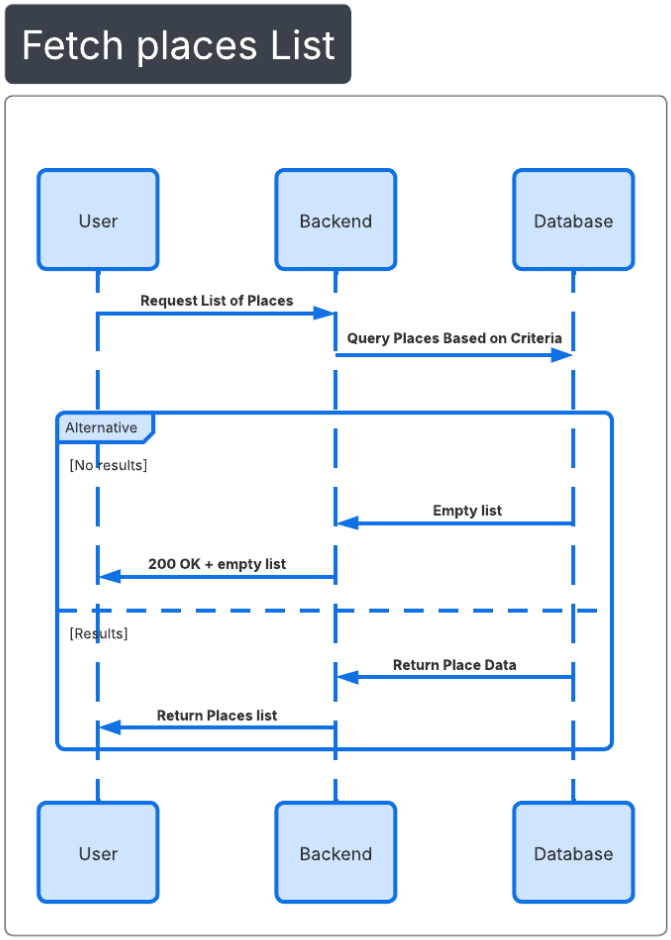


This diagram shows how a user creates a review for a location in the app.
The user submits his request : He submits the placeId, its rating and its comment.
The backend verifies the data : It validates the required fields and verifies that the grade is within the allowed range.
He checks in the database if the place exists. Two possible outcomes:

- If the place doesn't exist : "Place not found" error message.
- If the place exists : the review is saved in the database with the user ID, place ID, rating, comment and creation date.

If successful, the user receives a confirmation with the review ID created

3.4 Fetch Places List :



This diagram shows how a user views the list of available places in the app.
The user requests the list : he sends a request to obtain the list of places.
The backend queries the database : it makes a request to retrieve the premises according to the requested criteria
Two possible outcomes :

- If no place matches : the database returns an empty list, the backend returns a "200 OK" code with an empty list.
- If places match : the database returns the data of the places, the backend passes the complete list to the user.