Staycation: Technical Report

Josue Urrego Lopez // Ticonsky
Universidad Distrital Francisco José de Caldas
Email: jurregol@udistrital.edu.co

***Abstract*—This technical report presents the design and implementation of the backend for the Staycation platform. The report details the use of various object-oriented classes, design patterns, APIs, Dockerization, and database management to create a modular, scalable, and maintainable system.**

I. Volatile Classes

Volatile classes are used in the Staycation platform to handle the dynamic instantiation of objects and enhance system modularity. These classes allow the creation and manipulation of specific object instances at runtime, facilitating flexibility and extensibility. By using volatile classes, we can encapsulate data and behaviors coherently and structurally, resulting in a cleaner and more maintainable software design. Moreover, this approach allows the system to adapt easily to future changes or the inclusion of new functionalities without significantly modifying the existing code.

II. Value Object (VO) Classes

A. UserVO

1) Description: The `UserVO` class represents a user on the platform. Its purpose is to encapsulate user-related data in a structured manner.

2) Attributes:

- `userRole` (str): The user's role on the platform (e.g., admin, owner, guest).
- `name` (str): The user's name.
- `email` (str): The user's email address.
- `hashedPassword` (str): The user's password, securely stored using hashing.
- `phone` (str): The user's phone number.

3) Methods:

- `__init__`: Initializes a new instance of `UserVO`.
- `getUserRole`: Returns the user's role.
- `setUserRole`: Sets the user's role.
- `getName`: Returns the user's name.
- `setName`: Sets the user's name.
- `getEmail`: Returns the user's email.
- `setEmail`: Sets the user's email.
- `getPhone`: Returns the user's phone number.
- `setPhone`: Sets the user's phone number.
- `hashPassword`: Method to apply hashing to the user's password.

B. PropertyVO

1) Description: The `PropertyVO` class represents a property on the platform. Its purpose is to encapsulate property-related data in a structured manner.

2) Attributes:

- `userId` (int): The unique identifier of the user who owns the property.
- `propertyType` (str): The type of property (e.g., apartment, house).
- `propertyAddon` (str): Add-ons available in the property (e.g., wifi, kitchen).
- `location` (str): The property's location.
- `guests` (int): The property's guest capacity.
- `rooms` (int): The number of rooms.
- `beds` (int): The number of beds.
- `bathrooms` (int): The number of bathrooms.
- `photos` (list): A list of property photos.
- `name` (str): The property's name.
- `description` (str): The property's description.
- `price` (float): The price per night of the property.

3) Methods:

- `__init__`: Initializes a new instance of `PropertyVO`.
- `get_userId`: Returns the user ID of the property owner.
- `set_userId`: Sets the user ID of the property owner.
- `get_propertyType`: Returns the property type.
- `set_propertyType`: Sets the property type.
- `get_propertyAddon`: Returns the property's add-ons.
- `set_propertyAddon`: Sets the property's add-ons.
- `get_location`: Returns the property's location.
- `set_location`: Sets the property's location.
- `get_guests`: Returns the guest capacity.
- `set_guests`: Sets the guest capacity.
- `get_rooms`: Returns the number of rooms.
- `set_rooms`: Sets the number of rooms.
- `get_beds`: Returns the number of beds.
- `set_beds`: Sets the number of beds.
- `get_bathrooms`: Returns the number of bathrooms.
- `set_bathrooms`: Sets the number of bathrooms.
- `get_photos`: Returns the list of property photos.
- `set_photos`: Sets the list of property photos.
- `get_name`: Returns the property's name.
- `set_name`: Sets the property's name.
- `get_description`: Returns the property's description.
- `set_description`: Sets the property's description.
- `get_price`: Returns the property's price per night.
- `set_price`: Sets the property's price per night.

C. PropertyTypeVO

1) Description: The `PropertyTypeVO` class represents the types of properties on the platform. Its purpose is to encapsulate data related to property types in a structured manner.

2) Attributes:

- `name` (str): The name of the property type.
- `description` (str): The description of the property type.

3) Methods:

- `__init__`: Initializes a new instance of `PropertyTypeVO`.
- `get_name`: Returns the name of the property type.
- `set_name`: Sets the name of the property type.
- `get_description`: Returns the description of the property type.
- `set_description`: Sets the description of the property type.

D. PropertyAddonVO

1) Description: The `PropertyAddonVO` class represents property add-ons on the platform. Its purpose is to encapsulate data related to property add-ons in a structured manner.

2) Attributes:

- `wifi` (bool): Indicates if the property has wifi.
- `kitchen` (bool): Indicates if the property has a kitchen.
- `parking` (bool): Indicates if the property has parking.
- `staffService` (bool): Indicates if the property offers staff service.
- `pool` (bool): Indicates if the property has a pool.
- `securityCameras` (bool): Indicates if the property has security cameras.
- `laundry` (bool): Indicates if the property has a laundry facility.
- `gym` (bool): Indicates if the property has a gym.

3) Methods:

- `__init__`: Initializes a new instance of `PropertyAddonVO`.
- `set_wifi`: Sets if the property has wifi.
- `set_kitchen`: Sets if the property has a kitchen.
- `set_parking`: Sets if the property has parking.
- `set_staffService`: Sets if the property offers staff service.
- `set_pool`: Sets if the property has a pool.
- `set_securityCameras`: Sets if the property has security cameras.
- `set_laundry`: Sets if the property has a laundry facility.
- `set_gym`: Sets if the property has a gym.
- `get_wifi`: Returns if the property has wifi.
- `get_kitchen`: Returns if the property has a kitchen.
- `get_parking`: Returns if the property has parking.
- `get_staffService`: Returns if the property offers staff service.
- `get_pool`: Returns if the property has a pool.
- `get_securityCameras`: Returns if the property has security cameras.
- `get_laundry`: Returns if the property has a laundry facility.
- `get_gym`: Returns if the property has a gym.

E. CommentVO

1) Description: The `CommentVO` class represents a comment on the platform. Its purpose is to encapsulate comment-related data in a structured manner.

2) Attributes:

- `commentId` (int): The unique identifier of the comment.
- `bookingId` (int): The booking identifier associated with the comment.
- `userId` (int): The user identifier of the commenter.
- `content` (str): The content of the comment.
- `uploadDate` (str): The date the comment was uploaded.
- `rating` (int): The rating of the comment.

3) Methods:

- `__init__`: Initializes a new instance of `CommentVO`.
- `get_commentId`: Returns the comment ID.
- `set_commentId`: Sets the comment ID.
- `get_bookingId`: Returns the booking ID associated with the comment.
- `set_bookingId`: Sets the booking ID associated with the comment.
- `get_userId`: Returns the user ID of the commenter.
- `set_userId`: Sets the user ID of the commenter.
- `get_content`: Returns the content of the comment.
- `set_content`: Sets the content of the comment.
- `get_uploadDate`: Returns the upload date of the comment.
- `set_uploadDate`: Sets the upload date of the comment.
- `get_rating`: Returns the rating of the comment.
- `set_rating`: Sets the rating of the comment.

F. CardVO

1) Description: The `CardVO` class represents a payment card on the platform. Its purpose is to encapsulate payment card-related data in a structured manner.

2) Attributes:

- `cardId` (str): The unique identifier of the card.
- `userId` (int): The user identifier associated with the card.
- `cardNumber` (str): The card number.
- `cardOwner` (str): The owner of the card.
- `dueDate` (str): The expiration date of the card.
- `cvv` (str): The card's security code.
- `balance` (float): The card's balance.

3) Methods:

- `__init__`: Initializes a new instance of `CardVO`.
- `get_userId`: Returns the user ID associated with the card.
- `set_userId`: Sets the user ID associated with the card.

- get_cardNumber: Returns the card number.
- set_cardNumber: Sets the card number.
- get_cardOwner: Returns the card owner.
- set_cardOwner: Sets the card owner.
- get_dueDate: Returns the card's expiration date.
- set_dueDate: Sets the card's expiration date.
- get_cvv: Returns the card's security code.
- set_cvv: Sets the card's security code.
- get_balance: Returns the card's balance.
- set_balance: Sets the card's balance.
- get_cardId: Returns the card ID.
- set_cardId: Sets the card ID.

G. BookingVO

1) Description: The BookingVO class represents a booking on the platform. Its purpose is to encapsulate booking-related data in a structured manner.

2) Attributes:

- bookingId (str): The unique identifier of the booking.
- propertyId (int): The identifier of the booked property.
- userId (int): The identifier of the user making the booking.
- startingDate (str): The start date of the booking.

3) Methods:

- __init__: Initializes a new instance of BookingVO.
- get_bookingId: Returns the booking ID.
- set_bookingId: Sets the booking ID.
- get_propertyId: Returns the property ID of the booking.
- set_propertyId: Sets the property ID of the booking.
- get_userId: Returns the user ID of the booking.
- set_userId: Sets the user ID of the booking.
- get_startingDate: Returns the start date of the booking.
- set_startingDate: Sets the start date of the booking.

H. BillVO

1) Description: The BillVO class represents a bill on the platform. Its purpose is to encapsulate bill-related data in a structured manner.

2) Attributes:

- billId (int): The unique identifier of the bill.
- bookingId (int): The booking identifier associated with the bill.
- propertyId (int): The property identifier of the booked property.
- propertyPrice (float): The price of the booked property.
- userId (int): The user identifier of the person who made the booking.
- billStatus (str): The status of the bill.

3) Methods:

- __init__: Initializes a new instance of BillVO.
- get_billId: Returns the bill ID.
- set_billId: Sets the bill ID.
- get_bookingId: Returns the booking ID associated with the bill.
- set_bookingId: Sets the booking ID associated with the bill.
- get_propertyId: Returns the property ID of the booked property.
- set_propertyId: Sets the property ID of the booked property.
- get_propertyPrice: Returns the property price.
- set_propertyPrice: Sets the property price.
- get_userId: Returns the user ID of the person who made the booking.
- set_userId: Sets the user ID of the person who made the booking.
- get_billStatus: Returns the bill status.
- set_billStatus: Sets the bill status.

III. Data Access Object (DAO) Classes

DAO (Data Access Object) classes are fundamental to the database access logic on the Staycation platform. These classes perform CRUD (Create, Read, Update, Delete) operations on various system entities, allowing a clear separation between business logic and data access logic.

A. CardDAO

1) Description: The CardDAO class handles operations related to payment cards in the database.

2) Methods:

- create_card: Inserts a new card into the database.
- delete_card: Deletes a card from the database.
- update_card: Updates card data in the database.
- get_card: Retrieves card data from the database.

B. PropertyDAO

1) Description: The PropertyDAO class handles operations related to properties in the database.

2) Methods:

- create_property: Inserts a new property into the database.
- addAddonsProperty: Adds add-ons to a property in the database.

C. PropertyAddonDAO

1) Description: The PropertyAddonDAO class handles operations related to property add-ons in the database.

2) Methods:

- select_propertyAddon: Retrieves add-on data from the database.
- get_propertyAddons: Retrieves property add-ons based on specified characteristics.
- setAllCombinations: Inserts all possible combinations of property add-ons into the database.

D. PropertyTypeDAO

1) Description: The `PropertyTypeDAO` class handles operations related to property types in the database.

2) Methods:

- `create_propertyType`: Inserts a new property type into the database.
- `delete_propertyType`: Deletes a property type from the database.
- `update_propertyType`: Updates property type data in the database.
- `get_propertyType`: Retrieves property type data from the database.

E. UserDAO

1) Description: The `UserDAO` class handles operations related to users in the database.

2) Methods:

- `create_user`: Inserts a new user into the database.
- `delete_user`: Deletes a user from the database.
- `upgradeUser`: Updates a user's role in the database.
- `get_userID`: Retrieves a user ID from the database.

F. BillDAO

1) Description: The `BillDAO` class handles operations related to bills in the database.

2) Methods:

- `create_bill`: Inserts a new bill into the database.
- `delete_bill`: Deletes a bill from the database.
- `update_bill`: Updates bill data in the database.
- `get_bill`: Retrieves bill data from the database.

G. BookingDAO

1) Description: The `BookingDAO` class handles operations related to bookings in the database.

2) Methods:

- `create_booking`: Inserts a new booking into the database.
- `delete_booking`: Deletes a booking from the database.
- `update_booking`: Updates booking data in the database.
- `get_booking`: Retrieves booking data from the database.

H. CommentDAO

1) Description: The `CommentDAO` class handles operations related to comments in the database.

2) Methods:

- `create_comment`: Inserts a new comment into the database.
- `delete_comment`: Deletes a comment from the database.
- `update_comment`: Updates comment data in the database.
- `get_comment`: Retrieves comment data from the database.

IV. Design Patterns Used

In the Staycation platform, we used several design patterns to ensure a modular, scalable, and maintainable system. The most prominent patterns include Model-View-Controller (MVC), Singleton, and Factory.

A. Model-View-Controller (MVC) Pattern

1) Description: The MVC pattern is a software architecture that separates an application into three main components: Model, View, and Controller. This separation allows independent management of business logic, user interface, and application flow control.

- **Model**: Represents the structure of the data and the business logic of the application. In Staycation, the VO (Value Objects) and DAO (Data Access Objects) classes are part of the model.
- **View**: Handles the presentation of data to the user. Although Staycation focuses on the backend, any user interface built would interact with the data provided by the model.
- **Controller**: Acts as an intermediary between the model and the view, handling the application flow logic. In Staycation, controllers manage user requests, interact with DAOs to retrieve data, and then provide that data to the views.

2) Advantages of the MVC Pattern:

- **Modularity**: Separating business logic, presentation, and flow control facilitates system maintenance and evolution.
- **Reusability**: Model components can be reused in different parts of the application or even in other applications.
- **Scalability**: The separation of responsibilities allows each component to be scaled independently.

B. Singleton Pattern

1) Description: The Singleton pattern ensures that a class has only one instance and provides a global point of access to it. In Staycation, we use the Singleton pattern for the database connection and the property catalog.

2) Advantages of Singleton for Database Connection:

- **Consistency**: Ensures that all parts of the application use the same database connection.
- **Efficiency**: Reduces the overhead of creating multiple database connections.
- **Centralized Control**: Facilitates the management of the database connection from a single point.

3) Advantages of Singleton for Property Catalog:

- **Consistency**: Ensures that all parts of the application access the same property catalog.
- **Efficiency**: Avoids the need to load the property catalog multiple times.
- **Centralized Control**: Facilitates the management and update of the property catalog from a single point.

C. Factory Pattern

1) Description: The Factory pattern provides an interface for creating objects in a superclass but allows subclasses to alter the type of objects that are created. In Staycation, we use the Factory pattern for property creation.

2) Advantages of Factory Pattern:

- **Flexibility**: Allows the creation of different types of properties without changing the client code.
- **Reusability**: Facilitates the reuse of object creation code.
- **Centralization**: Centralizes object creation logic, making the code easier to maintain and scale.

V. APIs Used

In the Staycation platform, we use several APIs to enhance functionality and user experience. The most notable are the OpenStreetMap API for property location and a messaging API to facilitate communication between users and hosts.

A. OpenStreetMap API

1) Description: OpenStreetMap (OSM) is a collaborative project to create a free and editable map of the world. We use the OSM API to obtain geographical and location information for properties listed on the Staycation platform.

2) Functionalities:

- **Geocoding**: Converts addresses into geographic coordinates (latitude and longitude).
- **Mapping**: Provides interactive maps to show the location of properties.
- **Routing**: Calculates routes and distances between different locations.

3) Advantages of Using the OpenStreetMap API:

- **Free Access**: OpenStreetMap is open-source and free.
- **Collaborative Updates**: Data is continuously updated by a global community of contributors.
- **Flexibility**: The API allows easy and customizable integration into the platform.

B. Messaging API

1) Description: To facilitate communication between users and hosts on the Staycation platform, we use a messaging API that allows the sending and receiving of real-time messages.

2) Functionalities:

- **Message Sending**: Allows users to send messages to hosts and vice versa.
- **Message Receiving**: Users and hosts can receive messages in real-time.
- **Conversation History**: Maintains a record of all conversations between users and hosts.

3) Advantages of Using the Messaging API:

- **Real-Time Communication**: Facilitates immediate communication between users and hosts.
- **Simple Integration**: The API can be easily integrated into the Staycation platform.
- **Message History**: Keeps a record of all interactions for future reference.

VI. Dockerization of Staycation Project

Dockerizing the Staycation project allows packaging the application and its dependencies into a container, ensuring it runs uniformly across any environment. Docker facilitates the creation, deployment, and execution of containerized applications, ensuring consistency and portability.

A. Benefits of Dockerization

- **Portability**: Docker containers can run on any system with Docker installed, regardless of OS configuration differences.
- **Isolation**: Each container runs its own instance of the application and its dependencies, avoiding conflicts between different applications or versions.
- **Consistency**: Docker ensures the application runs the same way in development, testing, and production.
- **Ease of Deployment**: Docker simplifies the deployment process, enabling quick and reliable application implementation.

B. Project Structure

The Staycation project is divided into several components, each of which is dockerized separately to ensure modularity and scalability.

C. Dockerfile

Below is an example `Dockerfile` for the Staycation backend:

```
# Use an official Python base image
FROM python:3.9-slim

# Set the working directory
WORKDIR /app

# Copy the requirements file
and install dependencies
COPY requirements.txt .
RUN pip install --no-cache-dir
-r requirements.txt

# Copy the rest of the application code
COPY . .

# Specify the command to run the application
CMD ["python", "app.py"]
```

D. docker-compose.yml

To orchestrate multiple containers, we use `docker-compose`. Below is an example of a `docker-compose.yml` file that defines the services for the Staycation application:

```
version: '3.8'

services:
  db:
    image: mysql:5.7
    container_name: staycation_db
    environment:
      MYSQL_ROOT_PASSWORD: rootpassword
      MYSQL_DATABASE: staycation
      MYSQL_USER: user
      MYSQL_PASSWORD: userpassword
    ports:
      - "3306:3306"
    volumes:
      - db_data:/var/lib/mysql

  app:
    build: .
    container_name: staycation_app
    environment:
      - DATABASE_HOST=db
      - DATABASE_PORT=3306
      - DATABASE_USER=user
      - DATABASE_PASSWORD=userpassword
      - DATABASE_NAME=staycation
    ports:
      - "5000:5000"
    depends_on:
      - db

volumes:
  db_data:
```

E. Steps to Dockerize the Project

1) 1. Create the Dockerfile: The first step is to create a `Dockerfile` in the project's root directory. This file defines the Docker image to be used and how the container will be configured.

2) 2. Create the docker-compose.yml File: The `docker-compose.yml` file is used to define and run multiple Docker containers. In our case, we define services for the database and the backend application.

3) 3. Build and Run the Containers: Using Docker Compose, we can build and run the containers with a single command.

```
# Build the containers
docker-compose build

# Run the containers
docker-compose up
```

4) 4. Verify Deployment: Once the containers are running, the deployment can be verified by accessing the application at the specified port (e.g., `http://localhost:5000`) and connecting to the MySQL database at port 3306.

F. Additional Considerations

- **Data Persistence**: We use Docker volumes to maintain the persistence of MySQL database data.
- **Environment Variables**: Environment variables are used to configure the database and other application parameters.
- **Security**: Ensure secure handling of credentials and other application secrets. Docker Secrets is a recommended option for managing credentials in production.

VII. Staycation Database

The Staycation database is fundamental for storing and managing information about properties, users, bookings, and other relevant data. For the design and management of the database, we used SQL, DBeaver as the Database Management System (DBMS), and XAMPP as the database host.

A. Technologies Used

1) SQL: SQL (Structured Query Language) is used to create, modify, and query the database. SQL is an industry standard for managing relational databases and provides an efficient way to interact with data.

2) DBeaver: DBeaver is a universal database management tool that allows developers and administrators to interact with SQL and NoSQL databases. We used DBeaver to design, manage, and query the Staycation database.

3) XAMPP: XAMPP is a free software package that includes the Apache web server, the MySQL database management system, and tools for PHP and Perl. We used XAMPP to host the Staycation MySQL database during local development and testing.

B. Database Structure

The Staycation database is designed to be relational, with several interconnected tables representing the system's main entities. Below are descriptions of the most important tables:

1) User Table: The `User` table stores information about the users of the platform. It includes fields such as user ID, role, name, email, hashed password, and phone number. Each user is uniquely identified by a user ID, which is generated using a UUID.

2) Property Type Table: The `PropertyType` table stores different types of properties available on the platform, such as apartments or houses. Each property type is identified by a unique property type ID, which is auto-incremented.

3) Property Addons Table: The `PropertyAddon` table stores the available add-ons for properties, such as Wi-Fi, kitchen, parking, staff service, pool, security cameras, laundry, and gym. Each add-on type is stored as a boolean value indicating its availability.

4) Property Table: The `Property` table stores detailed information about each property listed on the platform. This includes the property ID, owner ID, type ID, add-ons ID, location, guest capacity, available rooms, beds, bathrooms, media (photos and videos), name, description, and price. The table uses foreign keys to link properties to their owners, types, and add-ons.

5) Card Table: The `Card` table stores payment card information associated with users. It includes the card ID, user ID, card number, card owner, expiration date, CVV, and balance. The card ID is generated using a UUID, and the balance must be greater than zero.

6) Booking Table: The `Booking` table stores information about property bookings made by users. It includes the booking ID, property ID, user ID, and the starting date of the booking. The table uses foreign keys to link bookings to properties and users.

7) Comment Table: The `Comment` table stores user comments on bookings. It includes the comment ID, booking ID, user ID, content, upload date, and rating. The table uses foreign keys to link comments to bookings and users. The rating must be between 0 and 5.

8) Bill Table: The `Bill` table stores billing information for bookings. It includes the bill ID, booking ID, property ID, user ID, and bill status. The table uses foreign keys to link bills to bookings, properties, and users.

C. Design and Management with DBeaver

DBeaver was used to design and manage the Staycation database. This tool provides a graphical interface that simplifies the creation and modification of tables, the execution of SQL queries, and the administration of data.

D. Hosting with XAMPP

During local development and testing, the Staycation MySQL database was hosted using XAMPP. This integrated server environment provides all the tools necessary to run a database server on a local machine, enabling quick and efficient development and testing.

VIII. Conclusion

The Staycation backend is designed to be robust, efficient, and maintainable, leveraging SQL, DBeaver, XAMPP, Docker, and various design patterns. The modular structure, combined with the use of industry-standard tools and practices, ensures the system can adapt to future needs and scale effectively.