Software Design Course

Universidad Distrital Francisco José de Caldas

Partial project delivery

Submitted by:

Andrés Felipe Vanegas Bogotá

Instructor:

Carlos Andrés Sierra Virguez

May 29, 2024

1. **Introduction**

The vehicle management system is an application designed to manage a catalog of vehicles, allowing users to add, remove, and search for vehicles based on different criteria such as speed and price. This technical report provides a detailed overview of the system architecture and design, highlighting technical decisions, design patterns, and software engineering principles applied in its development.

2. **System Architecture**

The system is composed of several interrelated modules that work together to provide specific functionalities. The following main components are highlighted:

- Catalog Subsystem (core_subsystem): Manages information related to vehicles, including the addition, removal, and searching of vehicles by speed and price. This subsystem utilizes a proxy pattern to interact with the actual catalog class, providing an additional layer of abstraction.
- Authentication Subsystem (core_subsystem): Responsible for authenticating users and managing their registration in the system. It uses a JSON file to store user data and permissions, enabling authentication and authorization of actions based on roles.

3. **Technical Decisions and Applied Design Patterns**

- Singleton: The Singleton pattern is implemented in the `catalog_proxy.py`, ensuring that only one instance of this object exists throughout the system. This improves efficiency by avoiding the creation of multiple unnecessary instances and ensures the consistency of catalog data.

- Proxy: The Proxy pattern is used to provide an additional level of control over access to the `Catalog`. The `CatalogProxy` acts as an interface between clients and the actual catalog, allowing the addition of additional functionalities such as operation logging and memory cleanup without directly modifying the catalog class.

- Decorator: The Decorator pattern is employed to add additional functionalities to the `Catalog`, such as runtime measurement and memory cleanup. The `TimeDecorator` and `MemoryDecorator` encapsulate these functionalities, allowing for easy extension and composition of behaviors.

4. Applied Software Engineering Principles

- SOLID Principle: The SOLID principles are applied to ensure a modular, flexible, and easy-to-maintain design. For example, the Single Responsibility Principle is respected by assigning specific responsibilities to each class and method, facilitating code understanding and modification.

- Coupling and Cohesion: Low coupling and high cohesion are maintained between system components. This is achieved by dividing the system into independent modules and defining clear interfaces between them, facilitating code modification and reuse.

5. Technical Challenges, Concerns, and Decisions

- One of the main challenges encountered in this project was working with Poetry and the virtual environment, primarily due to issues with paths or environment variables. The solution that worked was creating a virtual environment with `python -m venv asd-env` and installing Poetry. Additionally, modifying the `pyproject.toml` file to include the CLI script helped execute the program properly.

- In the `user_authentication` module, connecting the JSON file posed difficulties, requiring changes to the file path and implementing user registration and login validation. Issues with the JSON file path and Visual Studio Code were resolved by closing and reopening the editor.

- In the main module, more functionalities were added to the menu, and variables like name, speed, and price were defined in the catalog for future search functionalities, which were not completed. With the assistance of a peer, memory cleanup functionality was implemented in the main module.