STUDENT: Laura Daniela Muñoz Ipus

CODE: 20221020022

COURSE: Software design

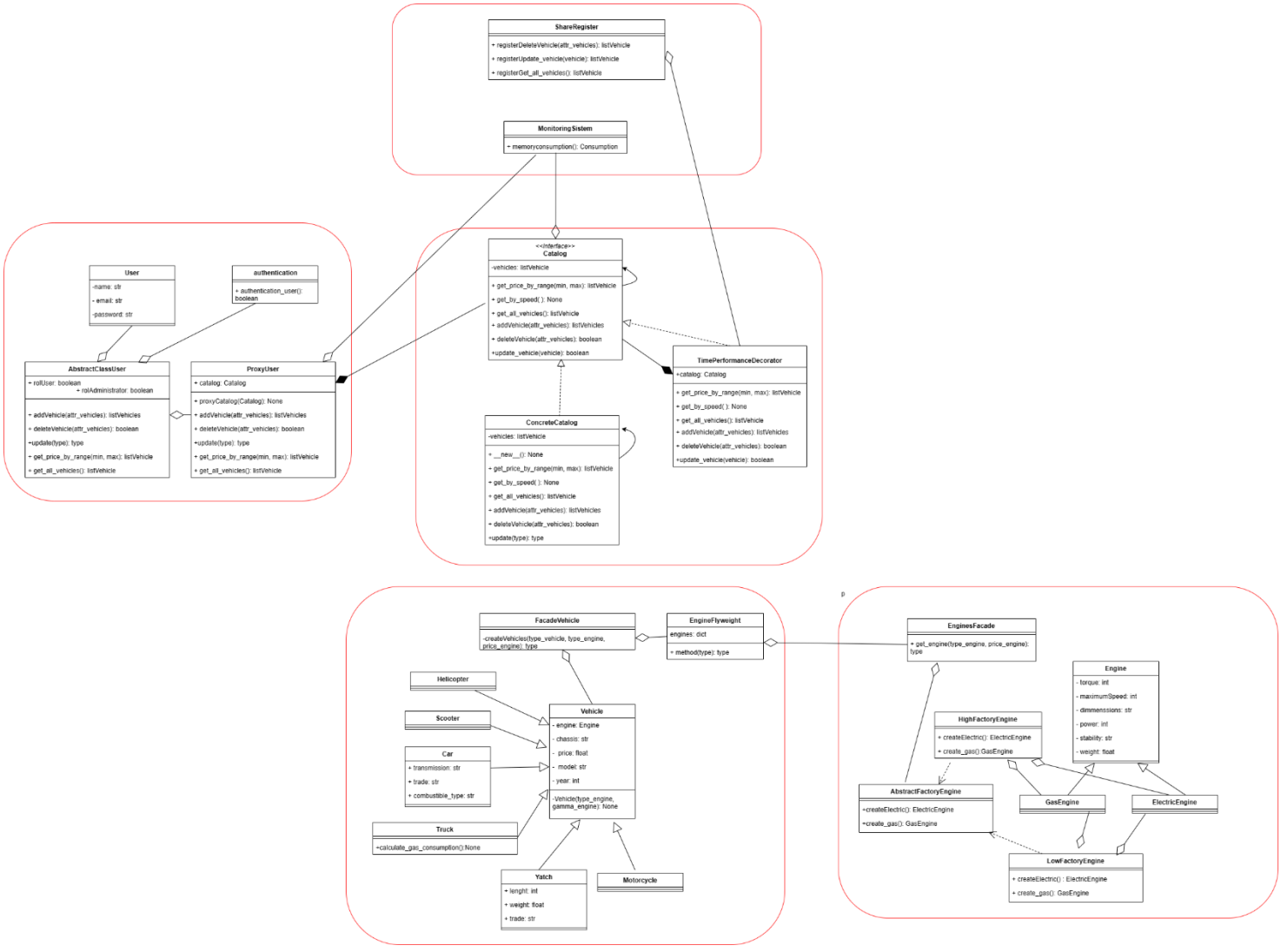
DATE: 26th April 2024

WORKSHOP_3

TECHNICAL REPORT

For this workshop number 3 we wanted to implement an application of a vehicle catalog, as in workshop 1 and 2, however, in this case some modifications were made so that there is a user interaction with the vehicle catalog through an interface, doing this in a way that can optimize memory and flexibility in the application using structural patterns.

The following is the class diagram divided by components.



To manage the logic and organize the application in a more optimal way, we had to manage subsystems, so that the user at the time of interacting with the interface does not have to see the complexity of the application and it is hidden, and this was achieved by using facade for the subsystem of the creation of engines and for the subsystem of the creation of vehicles.

To reduce memory usage and increase system performance, we had to implement the Flyweight structural pattern for vehicle engines, since this pattern allows us to store several objects that are the same (in this case the engines) in the same memory space, without the need to occupy a memory space for each engine being created for each vehicle.

Another way in which memory usage is being reduced in an efficient way is through the user, as this is divided into two, users who can view and search for vehicles and the administrator which is responsible for creating, updating or deleting vehicles, to avoid having to handle two different users by inheriting a user authentication class was created and the Proxy pattern, which makes a kind of encapsulation or wrapper to the user class and makes an access control and security to the user class.

In order to analyze the memory consumption of the application, a monitoring system was created to analyze the changes made to the vehicles and the searches performed.

In order to guarantee the security of the data, the attributes and methods of vehicles, engines and user attributes were encapsulated, making them private.

Implementation of SOLID principles

In order to promote a code that is easy to maintain, and that has code quality, being robust and scalable, the SOLID principles had to be respected and for this we are going to analyze if the 5 principles were respected:

- Single Responsibility Principle: The logical part of the application is very large and with many classes, precisely to ensure that each class has only one responsibility and that reuse and modification of code without affecting other parts of the system is not hindered.
- Open/Closed Principle: open for extension and addition of new functionalities, but closed for modification. Since the decorator pattern has been used for the catalog class, it allows extending the behavior of a class without modifying the existing code, which promotes system extensibility.
- Liskov Substitution Principle: It guarantees that a child class can be replaced as a parent class without affecting the code and it can be seen that inheriting classes have the same methods.
- Interface Segregation Principle: To comply with this principle, the interface was taken and divided into smaller interfaces. The catalog, vehicles and engines part was

divided, which the logical interface of vehicles and engines is represented by the facade structural pattern. The dependencies that the classes had were also taken into account.

- Dependency inversion principle: Abstractions were used so that low level modules do not interact directly with high level modules, allowing modularity and flexibility in the code.