TECHNICAL REPORT

User stories:

- As an admin, I want to create, delete, and update both engines and vehicles, so there will be a complete and updated vehicle catalog.
- As a user, I want to see the vehicle catalog so I can make a better decision about which vehicle to buy.
- As a user, I want to do filtered searches in the catalog, to make sure I find the best vehicle that suits my needs.
- As an admin, I want to calculate the gasoline consumption of trucks, to provide more accurate information on performance.
- As an admin, I want to create high-end and low-end engines, so I could offer vehicles of different price ranges.
- As a user and admin, I want to be able to log in with my account, or to be able to register in the system.
- As an admin, I want to be able to fully track all actions or changes in the catalog, as well as searches.
- As an admin, I want to monitor system resource consumption.

Entities:

Vehicles
Catalog
Engines
Gamma → This is a classification.
User
Admin

CRC Cards:

Vehicles	
- provide vehicle information	Engine Catalog

Truck	
- provide truck information - calculate gas consumption	Vehicle

Catalog	
- show a list of vehicles - create new vehicle, delete, update (admin only) - let user apply filters to find vehicles	Vehicles Engines User

Engines	
- provide engine information	Vehicles Catalog

FactoryLowEngines	
- create low price engines	Vehicles

FactoryHighEngines	
- create high price engines	Vehicles

User	
- show menu - handle user interaction options - login and registration	FacadeVehicleShop

Admin	
- create, delete, and update the catalog of vehicles	User

ProxyVehiclesFactory	
- Perform login, registration, monitoring, and caching for your most wanted vehicles. - It is the intermediary between access to the system and the Facade	SystemAcces FacadeVehiclesFactory

FacadeVehiclesFactory	
- It is the main interface for engine and vehicle system services.	ProxyVehiclesFactory SystemAcces

Technical decisions:

Complications:

When applying these two patterns together, I was not very sure how it should be done, because I initially proposed the proxy as an intermediary of the main interface (Which I later established as the Facade) and the catalog one which at that time was my service main, but when implementing the Facade for the two systems, I realized that something was wrong, since it only serves as a proxy for the vehicle system, noting that it was a design problem.

After noticing this error, I decided to create a "superior" interface, that is, a system access interface (SystemAccess) with the objective that the client would connect to it first, and then my proxy would act as an intermediary between it, and my main service, that is, the Facade.

Proxy pattern implementation

First, to meet the requirements of:

- Authentication system
- Change logging and search system
- Resource monitoring system

It was decided that the best way to implement it was through the structural pattern, Proxy, due to its ability to offer secure and centralized access control.

Since it acts as an intermediary, guaranteeing that only users can access the functionalities, performing a login, thus improving security and management flexibility.

In addition, it allows us to keep a detailed record of the changes and searches carried out, without affecting the operation of the systems, or for this to be noticed by the user, thus facilitating the control and analysis of the system.

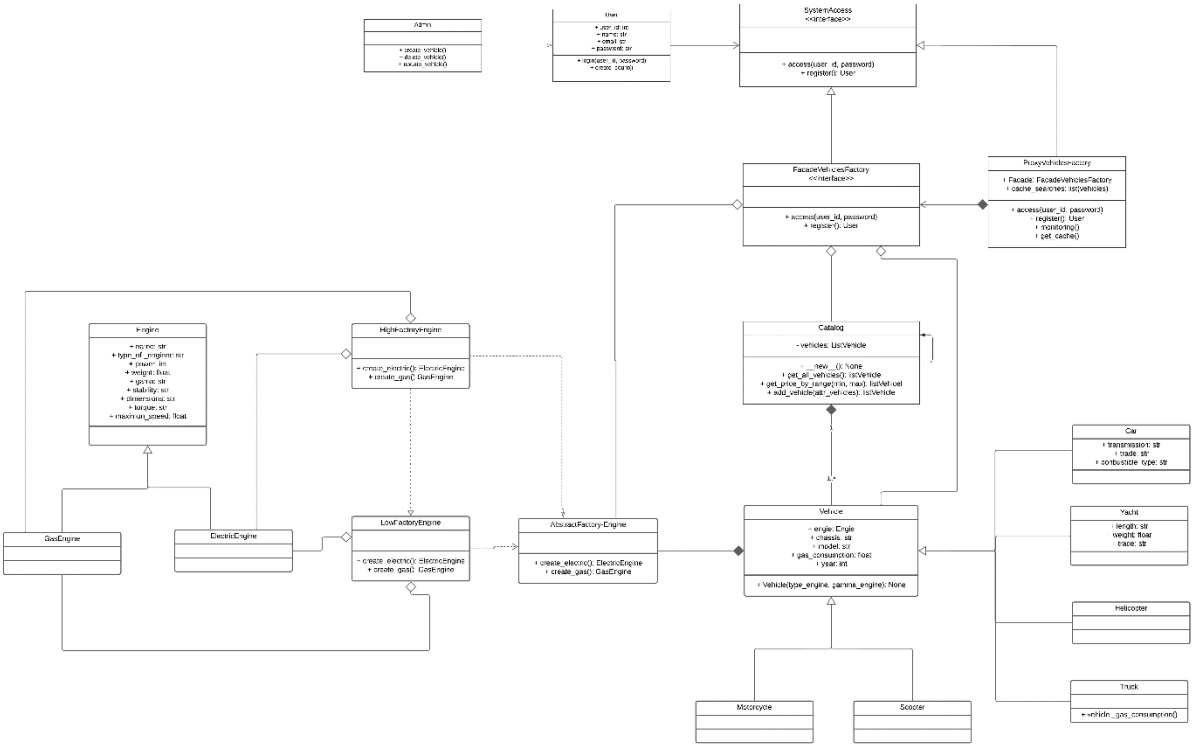
Finally, it also optimizes performance by allowing us to reduce resource consumption, being used as a cache memory for data, in this case the most wanted vehicles, ensuring efficient operation of the system.

After a design problem, it was established that this proxy should be the intermediary for all access to the system, and my main service, which is the Facade Vehicle Shop. After a design problem, it was established that this proxy should be the intermediary for all access to the system, and my main service, which is the Facade Vehicle Shop.

Facade pattern implementation

The Facade pattern was implemented with the objective of reducing system coupling and implementing a simpler user interface for the user. For this, it was necessary to establish the divisions of the subsystems, two being the engine system and the catalog system. Then, thanks to Facade, it allows us to encapsulate the internal complexity of the system behind a single and simplified interface, it allows us to facilitate the user's interaction with the system's functionalities without needing to understand its internal structure. Which not only reduces the complexity perceived by the user, but also helps us with better organization of the code.

UML


Admin
+ create_vehicle()
- delete_vehicle()
+ locate_vehicle()
User
+ user_id: int
- name: str
+ email: str
+ password: str
+ register(): e, password)
+ create_user()
SystemAccess
<<interface>>
+ access(user_id, password)
+ register(): User
FacadeVehiclesFactory
<<interface>>
+ access(user_id, password)
+ register(): User
ProxyVehiclesFactory
+ Facade: FacadeVehiclesFactory
+ cache_searches: list(vehicles)
+ access(user_id, password)
+ register(): User
+ monitoring()
+ get_cache()
Catalog
- vehicles: ListVehicle
__new__(): None
+ get_all_vehicles(): listVehicle
+ get_price_by_range(min, max): listVehicel
+ add_vehicle(all_vehicles): listVehicle
1
1..*
Vehicle
+ engle: Engle
+ chassis: str
+ model: str
+ gas_consumption: float
+ year: int
+ Vehicle(type_engine, panma_engine): None
Car
+ transmission: str
+ trade: str
+ combustible_type: str
Yacht
+ length: str
weight: floar
trace: str
Helicopter
Truck
+ vehicle_gas_consumption()
Motorcycle
Scooter
IEngine
+ no.rec: str
+ type_of_engine: str
+ power: int
+ weight: float
+ gear: str
+ stability: str
+ dimensions: str
+ torque: str
+ maximun_speed: float
HighFactoryEngine
+ create_electric(): ElectricEngine
+ create_gas(): GasEngine
LowFactoryEngine
- create_electric(): ElectricEngine
+ create_gas(): GasEngine
AbstractFactory-Engine
+ create_electric(): ElectricEngine
+ create_gas(): GasEngine
GasEngine
ElectricEngine