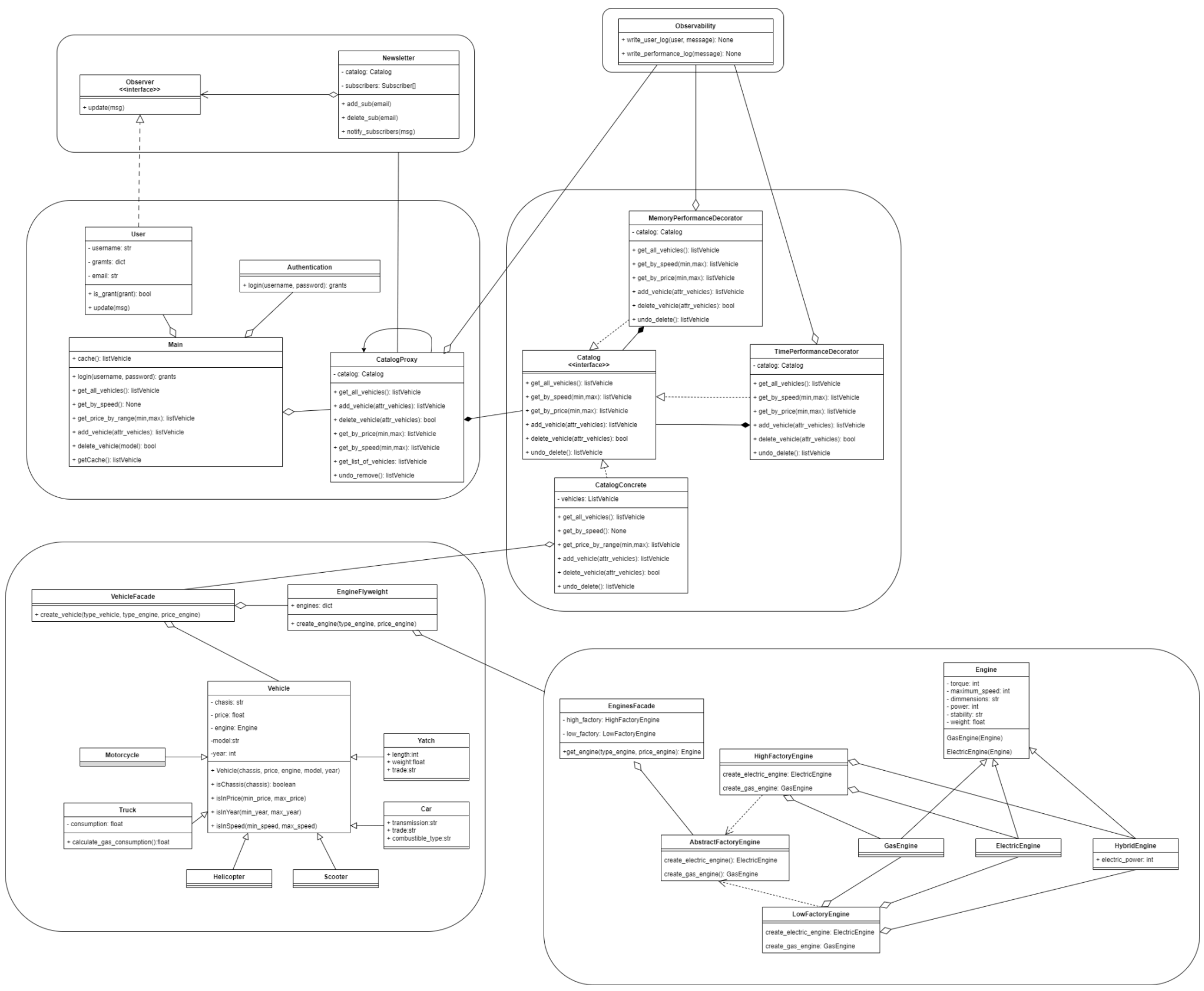
Technical Report

Class Diagram:


Observer
<<interface>>
+ update(msg)
Newsletter
- catalog: Catalog
- subscribers: Subscriber[]
+ add_sub(email)
+ delete_sub(email)
+ notify_subscribers(msg)
Observability
+ write_user_log(user, message): None
+ write_performance_log(message): None
User
- username: str
- gramts: dict
- email: str
+ is_grant(grant): bool
+ update(msg)
Authentication
+ login(username, password): grants
Main
+ cache(): listVehicle
+ login(username, password): grants
+ get_all_vehicles(): listVehicle
+ get_by_speed(): None
+ get_price_by_range(min,max): listVehicle
+ add_vehicle(attr_vehicles): listVehicle
+ delete_vehicle(model): bool
+ getCache(): listVehicle
CatalogProxy
- catalog: Catalog
+ get_all_vehicles(): listVehicle
+ add_vehicle(attr_vehicles): listVehicle
+ delete_vehicle(attr_vehicles): bool
+ get_by_price(min,max): listVehicle
+ get_by_speed(min,max): listVehicle
+ get_list_of_vehicles: listVehicle
+ undo_remove(): listVehicle
MemoryPerformanceDecorator
- catalog: Catalog
+ get_all_vehicles(): listVehicle
+ get_by_speed(min,max): listVehicle
+ get_by_price(min,max): listVehicle
+ add_vehicle(attr_vehicles): listVehicle
+ delete_vehicle(attr_vehicles): bool
+ undo_delete(): listVehicle
Catalog
<<interface>>
+ get_all_vehicles(): listVehicle
+ get_by_speed(min,max): listVehicle
+ get_by_price(min,max): listVehicle
+ add_vehicle(attr_vehicles): listVehicle
+ delete_vehicle(attr_vehicles): bool
+ undo_delete(): listVehicle
TimePerformanceDecorator
- catalog: Catalog
+ get_all_vehicles(): listVehicle
+ get_by_speed(min,max): listVehicle
+ get_by_price(min,max): listVehicle
+ add_vehicle(attr_vehicles): listVehicle
+ delete_vehicle(attr_vehicles): bool
+ undo_delete(): listVehicle
CatalogConcrete
- vehicles: ListVehicle
+ get_all_vehicles(): listVehicle
+ get_by_speed(): None
+ get_price_by_range(min,max): listVehicle
+ add_vehicle(attr_vehicles): listVehicle
+ delete_vehicle(attr_vehicles): bool
+ undo_delete(): listVehicle
VehicleFacade
+ create_vehicle(type_vehicle, type_engine, price_engine)
EngineFlyweight
+ engines: dict
+ create_engine(type_engine, price_engine)
Vehicle
- chasis: str
- price: float
- engine: Engine
-model:str
-year: int
+ Vehicle(chassis, price, engine, model, year)
+ isChassis(chassis): boolean
+ isInPrice(min_price, max_price)
+ isInYear(min_year, max_year)
+ isInSpeed(min_speed, max_speed)
Motorcycle
Yatch
+ length:int
+ weight:float
+ trade:str
Truck
- consumption: float
+ calculate_gas_consumption():float
Car
+ transmission:str
+ trade:str
+ combustible_type:str
Helicopter
Scooter
EnginesFacade
- high_factory: HighFactoryEngine
- low_factory: LowFactoryEngine
+get_engine(type_engine, price_engine): Engine
Engine
- torque: int
- maximum_speed: int
- dimmensions: str
- power: int
- stability: str
- weight: float
GasEngine(Engine)
ElectricEngine(Engine)
HighFactoryEngine
create_electric_engine: ElectricEngine
create_gas_engine: GasEngine
AbstractFactoryEngine
create_electric_engine(): ElectricEngine
create_gas_engine(): GasEngine
GasEngine
ElectricEngine
HybridEngine
+ electric_power: int
LowFactoryEngine
create_electric_engine: ElectricEngine
create_gas_engine: GasEngine

Technical concerns and decisions:

- For the item “Maybe an admin could make an error. So, provide an option to recover the last vehicle deleted” I could use Memento pattern or Command pattern, this was something more concrete and simple, so I could use specifically the Command pattern. However, it would add more complexity than it provided, so I chose to save the list of vehicles before each removal, and if the user wants to recover the last vehicle deleted just call the method “undo_delete()” and that’s it.
- In vehicles.facade I corrected the error of vehicle_type, in the code a comparison was being made with a literal word like “Car”, but it was receiving the number that agreed with the word “Car” in the menu, that was 1, so the code couldn’t make any comparison. Now the comparison is with the number, and not the word.
- In vehicles.facade, were some problems with the creation of vehicles, and the dict called “data”, the problem was that some keys doesn’t have the same name, when we want to create a vehicle, for example to get the information of char, the key was “engine_chassis”, but the key to create the car, called “chassis”, so it generated an error. Now I changed the key so the code works.
- When I choose the option of “Search all the vehicles” It generated an error, it was because in Decorator.py the class“class MemoryPerformanceDecoratorWindows:” in

the method “get_all_vehicles” didn’t have the return, so I added the return, also it happened in all methods of searching in that class, and I added the return, that fixed some errors.

- It was an error when searching by speed, because the numbers that were received were str, and not int, we can’t comparate < or > with str, so I fixed it with a cast, exactly the problem was in engines.py, in the class “Engine”, in the method “is_in_speed”.
- The problem with the register of user actions, was in catalog_proxy.py, in the class CatalogProxy, in the method add_vehicle, when we used Observability.write_user_log(username.get_username(), "A new vehicle had been added."), wasn’t necessary the .get_username(), because username it’s already an string with the username, so the method write_user_log, wasn’t receiving a string, so it caused the problem, just deleting the “.get_username” was enough to fix the problem.
- We used the Observer pattern to notify the users about the new cars in the catalog.
- Only users can add themselves as subscribers, and remove themselves as subscribers, and only admin can notify subscribers about the new cars in the catalog.
- Only the admin can undo the removal of a car. And only users can see the last three searches
- For the item of cache memory, I could use the prototype pattern, but I only need a few searches, so I preferred to save that in a list each time that a user made a search, because the pattern would add complexity to the code, and the way that I chose seemed easier.
- Because of time, I didn’t complete the last item.