DOCUMENTATION

1. **Conceptual changes from Workshop-1:**

   `Payment` Class Replaced by `CreditCard`:

   a. The `Payment` class, with its responsibilities of "Process user payments" and "Update user balance," has been replaced by the `CreditCard` class. This new class will focus on the management of information and operations related to credit cards. The collaborations previously established with `User` and `Trip` will now involve the `CreditCard` class.

   Implementation of Polymorphism in Inherited Methods:

   b. The decision has been made to implement polymorphism for the `requestTrip` method, which the `Driver` class inherits from the `User` class. In the `Driver` class, this method will be overridden and will have a different implementation named `offerTrip`. This will allow the behavior of handling trip requests to be specific to drivers, differing from the way users request trips.

2. **How we applied OOD:**

   - **Polymorphism:**

   As already said in the conceptual changes we applied this concept for overwrite method.

   - **Inheritance:**

     This concept has been applied to the class user and class driver because driver use the attributes of name, password, email, number and roles and the methods that validate these attributes. This allows us to make an inheritance, guarantees that the code will not be duplicated

   - **Abstraction:**

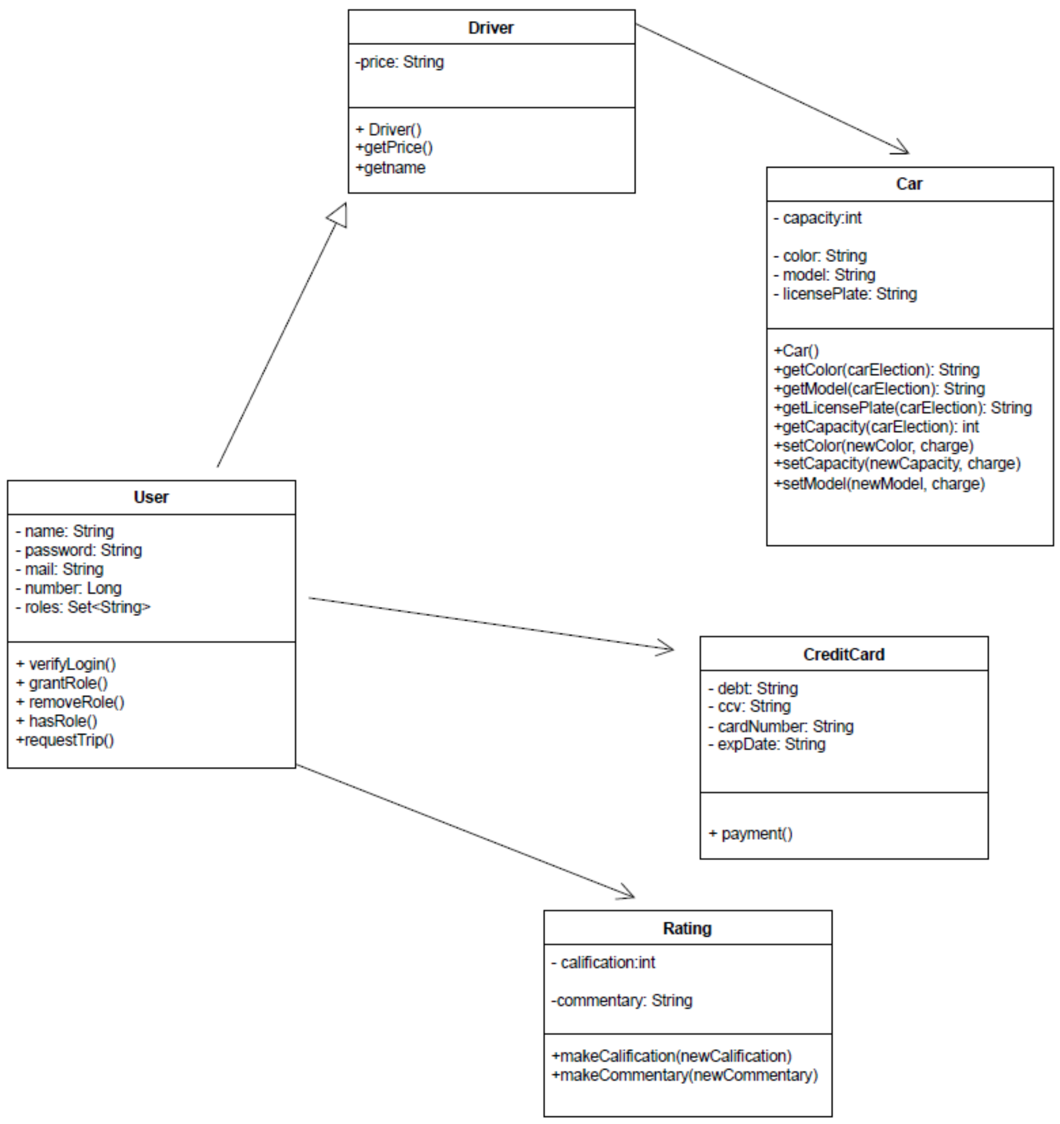
     In this case we used to know an applied which information is   important, this is shown in the classes, because the attributes that we made are the most important for the functionality of the program. Also, we will use it for the interaction between the objects, for example: we can hide information that is not revealed from the class driver, making that the class user only sees the necessary information.

   - **Encapsulation:**

     This is the concept that we more applicate in the program, because it is important to have a method that allows to see the information of the driver and the user. But only in specific contexts, for example: a user wants to see the model of the driver´s car, in this case we make a getter that analyze if the person who wants to see the information, has choose this car to make a trip. This guarantees that the personal information of the driver will not be public.

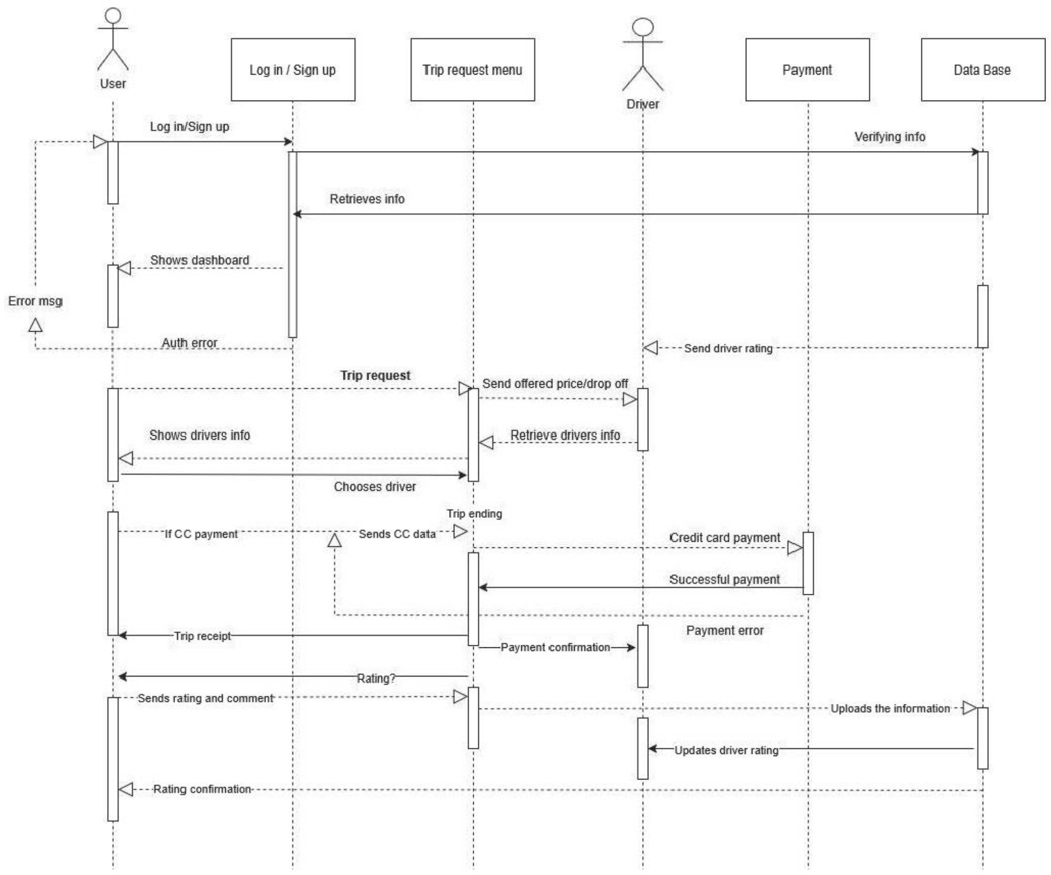
3. **Class and sequence diagram:**

- **Class diagram:**



This diagram shows the relation that exists between the classes. Notice that the user class is the most important, because it interacts with most of the classes: make inheritance with the driver class, have an interaction with the rating class and the credit card. Another class that interacts is the driver class with interacts with the car class.

- **Sequence diagram:**



This diagram shows how the object interacts to make a correct functionality.