Universidade do Minho
Licenciatura em Engenharia Informática

Development of Software Systems

2023/2024

E.S.IDEAL

https://github.com/LEI-DSS/trabalho-dss-grupo-09

Luís Ribeiro, 100608

Martim Félix, A100647

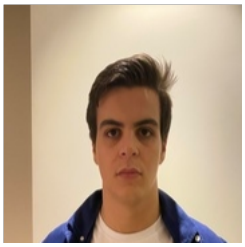
Diogo Santos, A100714

Filip Markovic, E11133

DSS

E.S.IDEAL

Luís Ribeiro - 

 **- Martim Félix**

Diogo Santos - 

 **- Filip Markovic**

01, 2024

Introduction

This project involves designing and implementing a system to automate customer admission and routing at E.S.Ideal's service stations. The system will be developed using a model-based approach with UML and a multi-layered architecture, focusing on Java. It aims to enhance operational efficiency by managing diverse vehicle types and service requirements, ensuring services match vehicle specifications. The project will be documented and tracked via GitHub, with an emphasis on practical understanding and application of theoretical concepts.

Use Cases

The diagram of the Use Cases here's this:

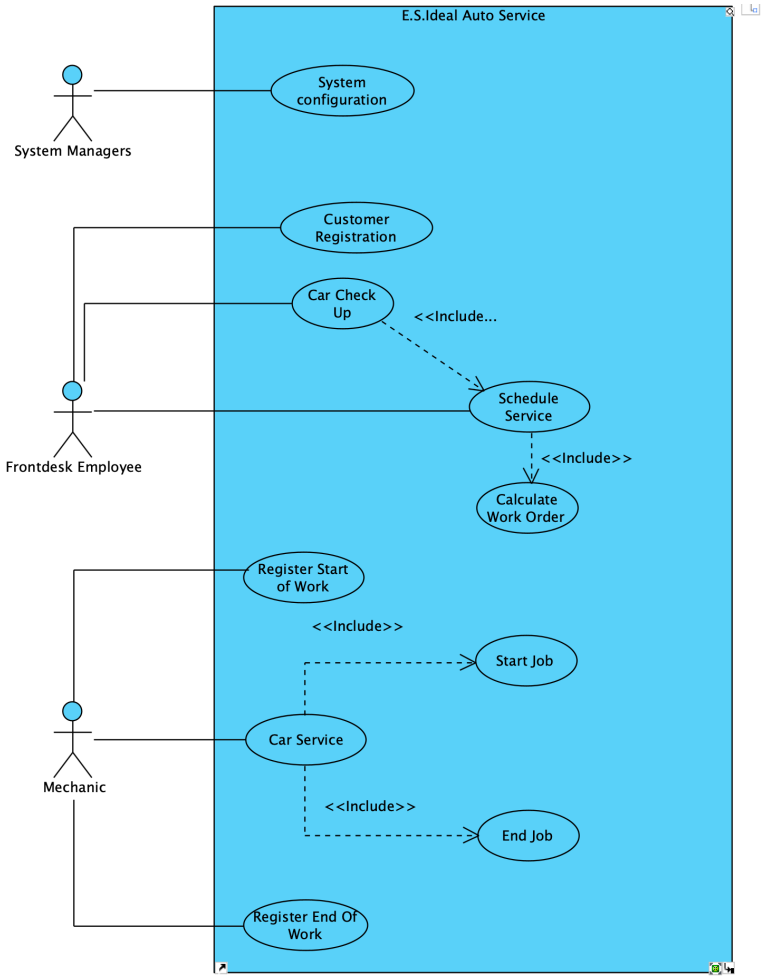


Figure 1: Use Cases Diagram

Start Job

Use Case: Start Job

Description: Mechanic starts a Job

Pre-Condition: There is an available job, Mechanic is on shift

Normal Behavior:

1- Mechanic Starts the Job in the Mechanic UI

2- The Job is updated as started in the DB

End Job

Use Case: End Job

Description: Mechanic ends a Job

Pre-Condition: Job started and assigned to corresponding mechanic , Mechanic is on shift

Normal Behavior:

1- Mechanic selects option to end the Job in the Mechanic UI

2- The Job is updated as finished in the DB

Start of Work/Shift

Use Case: Start of Work/ Start Shift

Description: Mechanic starts his shift

Pre-Condition: True

Normal Behavior:

1- Mechanic selects the option to Start his Shift

End of Work/Shift

Use Case: End of Work/ End Shift

Description: Mechanic ends his shift

Pre-Condition: Shift Started

Normal Behavior:

1- Mechanic selects the option to end his Shift.

System Configuration

Use case: System Configuration

Description: System Manager makes changes to the database and program

Scenario: Scenario 1

Precondition: true

Post-Condition: Changes made

Normal Flow:

1 - User enters credentials

2 - User adds new workstation

3 - User confirms changes

Flow Exception 1 [Invalid credentials for System Manager] (step 1)

1.1 - System informs that the credentials are invalid

1.2 Operation Canceled

Alternative Flow 2 [User Adds new employee] (step 2)

2.1 - User adds new employee and their credentials

2.2 - Return to step 3

Customer Registration

Use Case: Customer Registration
Description: The Frontdesk employee registers a new client in the E.S.Ideal's system
Scenario: A new customer enters in the E.S.Ideal car workshop and needs to be registered
Pre-Condition: True
Post-Condition: António gets registered in the system
Normal behaviour:

1- The frontdesk employee enters the name of the customer, VAT number, address and contact number.

Three Layer Architecture

Package Diagram

Our project is organized through a three layer architecture layers, visible through the existence of the UI, Business and Data packages, which constitute the presentation, business logic and data layers, respectively. This way, the following diagram was obtained:

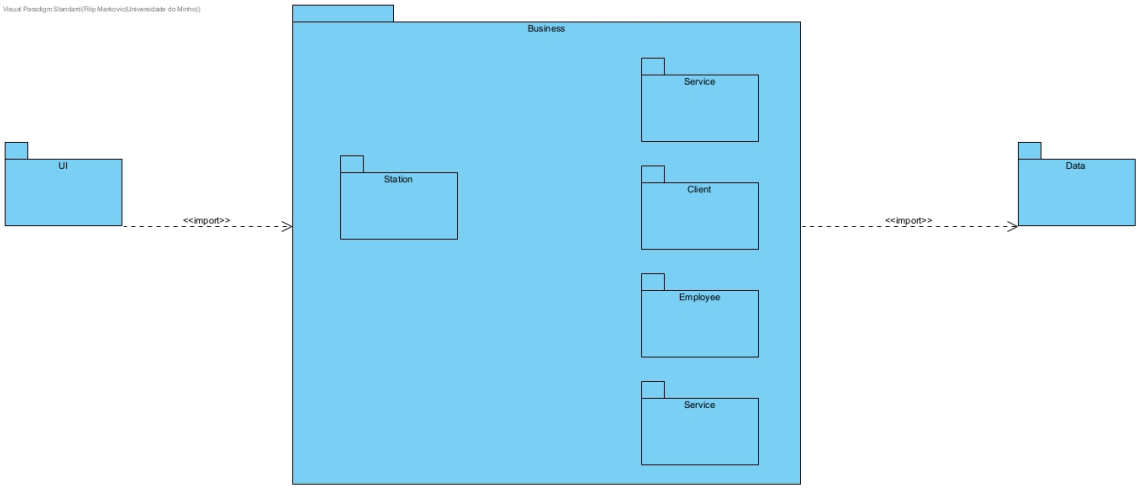


Figure 2: Packages Diagram

Component Diagram

The component diagram was made thinking about the database of the car workshop.

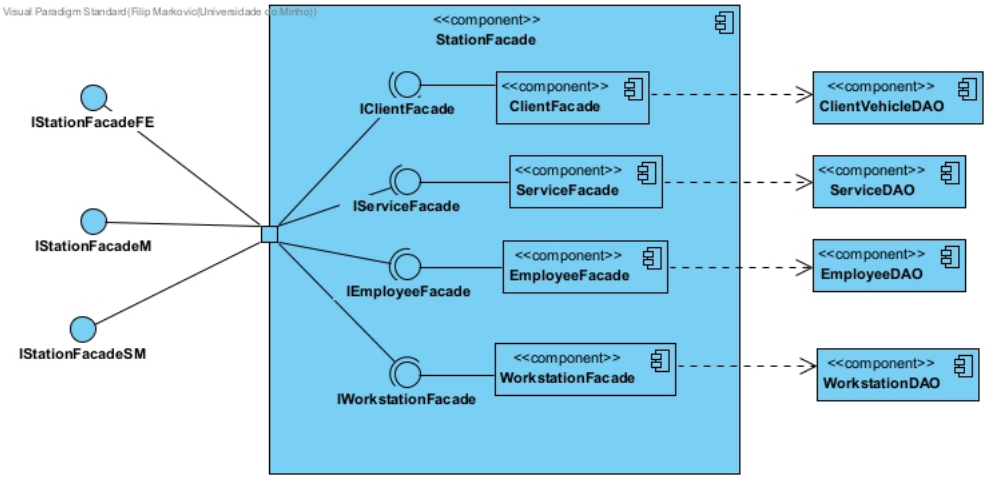


Figure 3: Component Diagram

DAOS

In order to implement a Database in the program, it was necessary to consider DAOs (Data Access Objects) and select classes to be persisted. DAOs allow you to persist objects in database, create objects from the database information and encapsulate sql queries. Furthermore, for each DAO, it was necessary to implement a set of base operations that allow the communication with the database and, consequently, access to stored data.

The methods effectively implemented were the methods used to support the use case selected.

Database

The implemented DAOs each have a table in the database excluding the ClientVehicleDAO that accesses both the Client and the Vehicle's tables. The following figure shows the logical model of the database that supports the program.

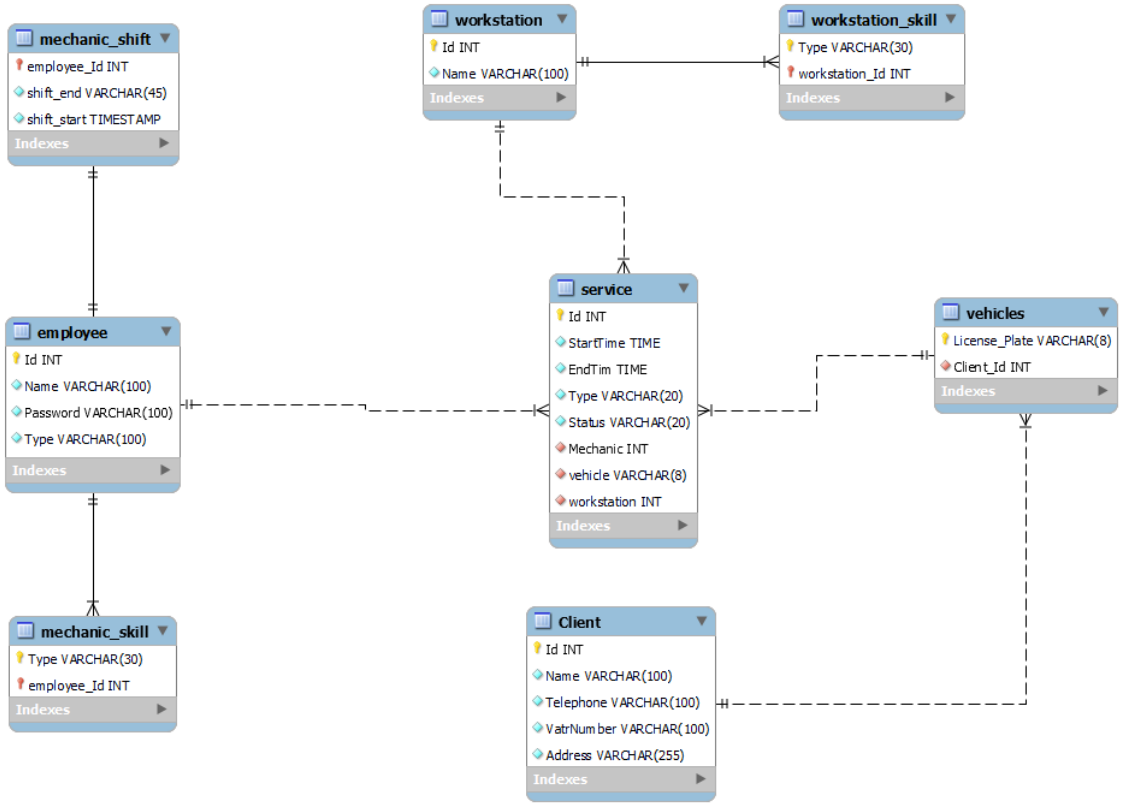


Figure 4: Logic Model of the Database

Domain Model

The Domain model identifies the key components gathered from the assignment sheet. We started by establishing relationships between more obvious entities like Mechanic, Workstation and Station. Then we went on to subdivide services into more specific types of services as in Universal Services or Specialized Services. Specialized Services were also divided into more categories. Lastly, we defined the car and the relationship between the client and his car. We also noted on the car's need for a datasheet.

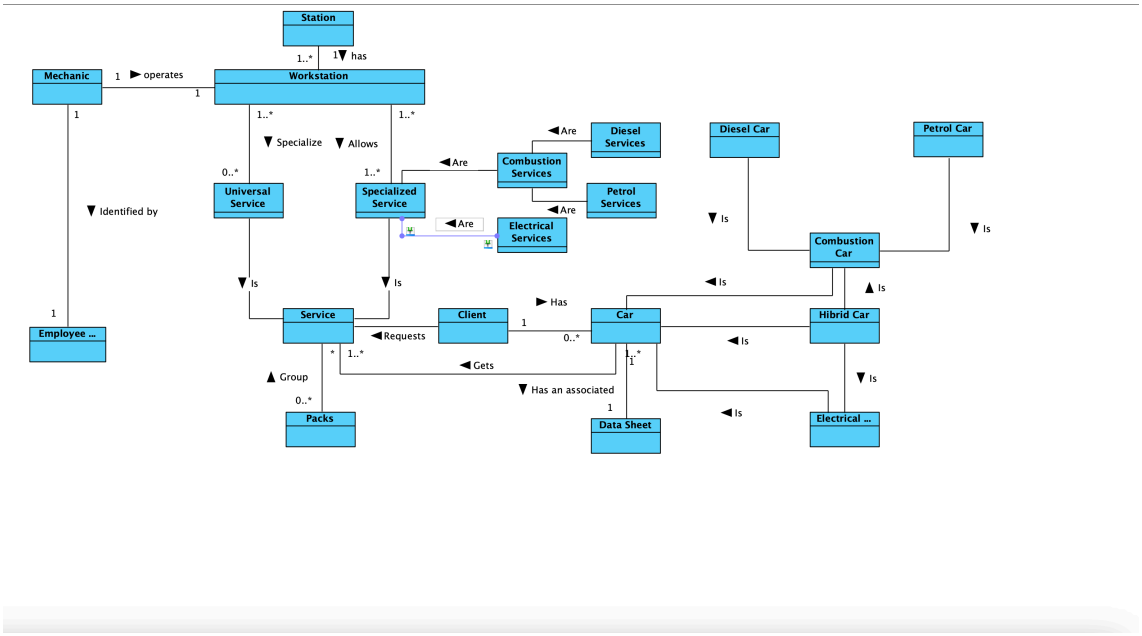


Figure 5: Domain Model

Class Diagram – Before Implementation

Next, we are going to show the class diagrams that we made before started the actual implementation of the project.

Station

This Diagram specifies the way the StationInterface is defined and how the other interfaces offer their own functionality to "solve" the station's facade methods.

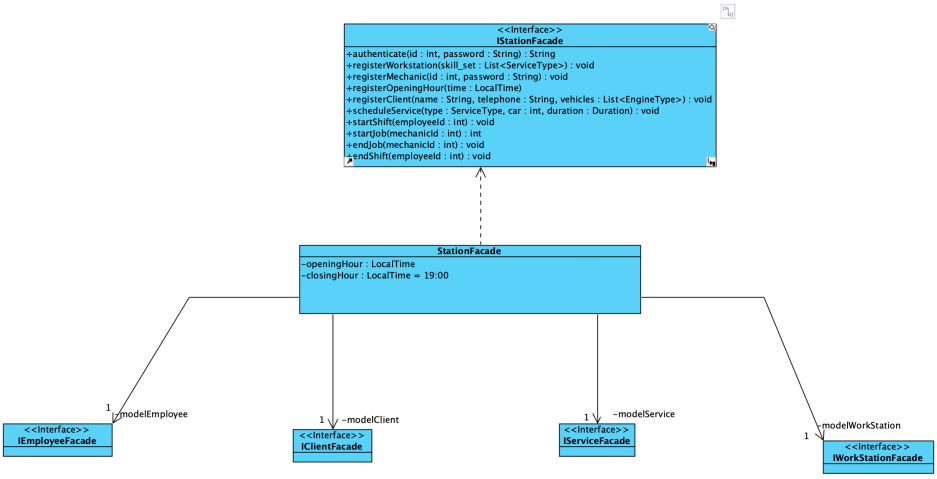


Figure 6: Class Diagram of the Station

Workstation

We implement the required IWorkStationFacade in the WorkStationFacade. This implementation requires some methods from the WorkStationDAO. This methods are also specified.

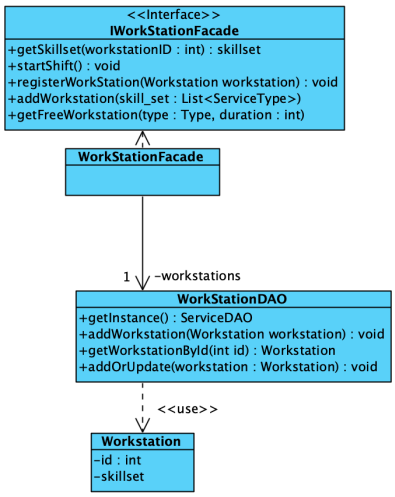


Figure 7: Class Diagram of the Workstation

Employee

Same logic behind the Workstation Diagram. Special atention to the definitions of FrontDesk and Mechanic as specifications of the Employee class.

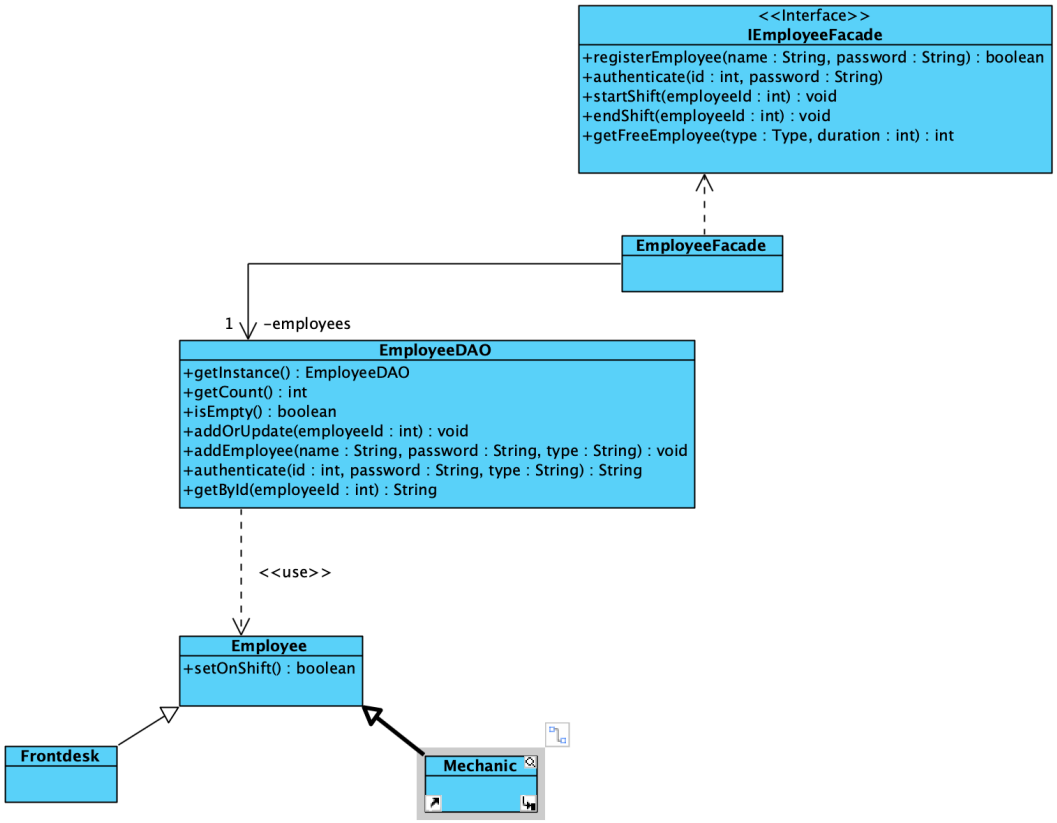


Figure 8:Class Diagram of the Employee

Client

The IClientFacade was latter on changed to be the IClient/VehicleFacade. We already did our opperations on the vehicles via the ClientDAO. We also added the methods that opperated on the Vehicle to this Facade.

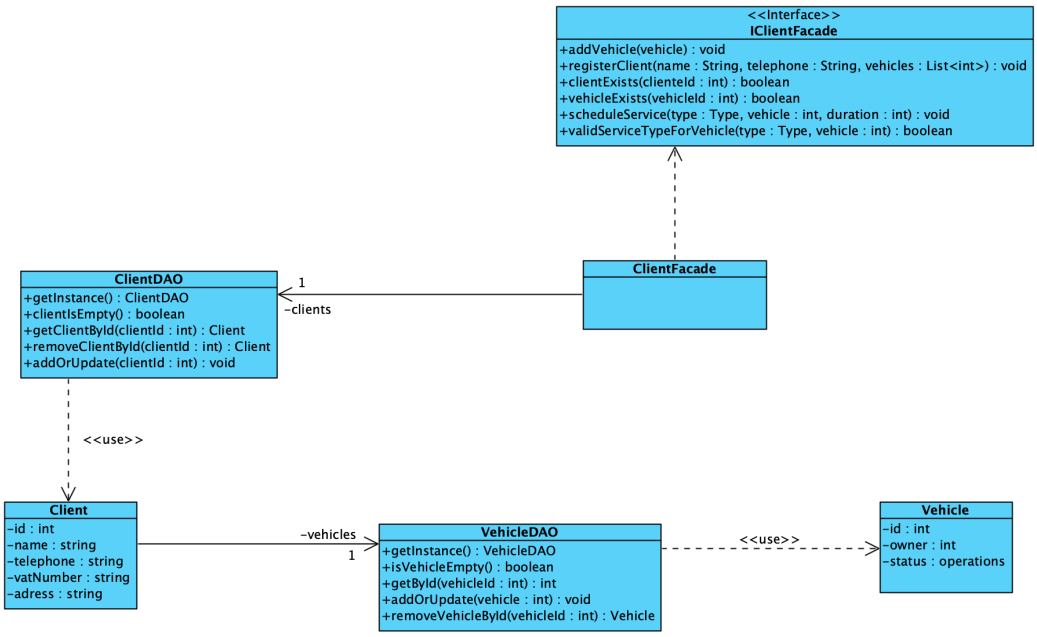


Figure 9: Class Diagram of the Client

Service

Same as the Workstation Diagram. Methods to implement and ServiceDAO to interact with the DataBase

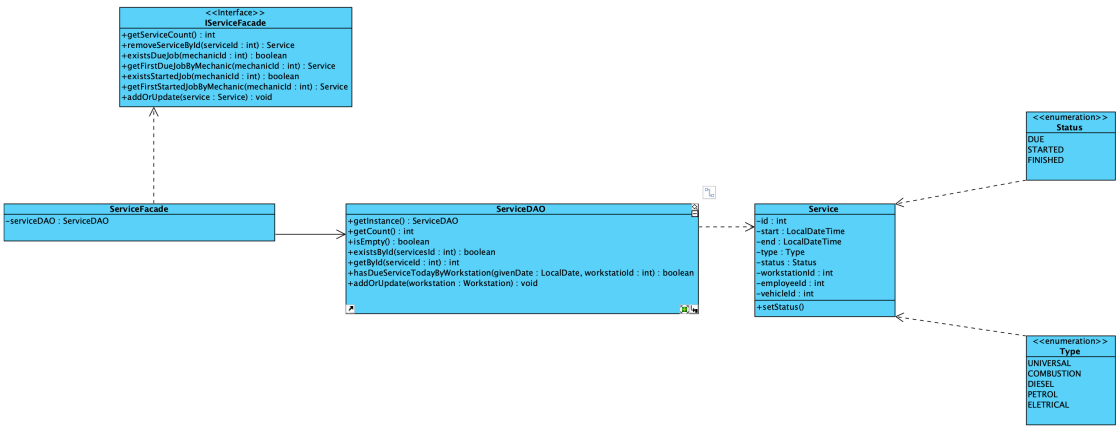


Figure 10: Class Diagram of the Service

Sequence Diagrams – Before implementation

The sequence diagrams that were designed, considering possible functionality, are the following:

Register Client

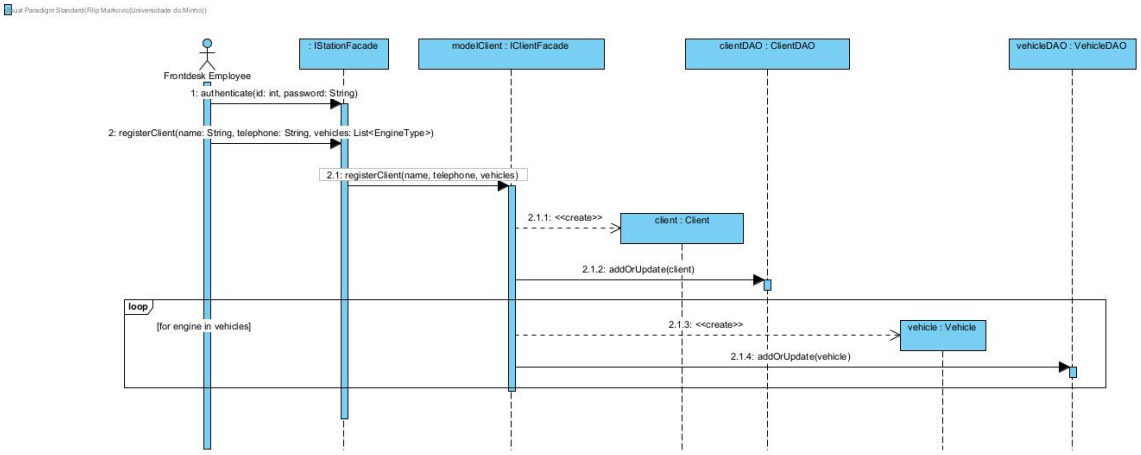


Figure 11: Sequence Diagram of the method registerClient

Start Shift

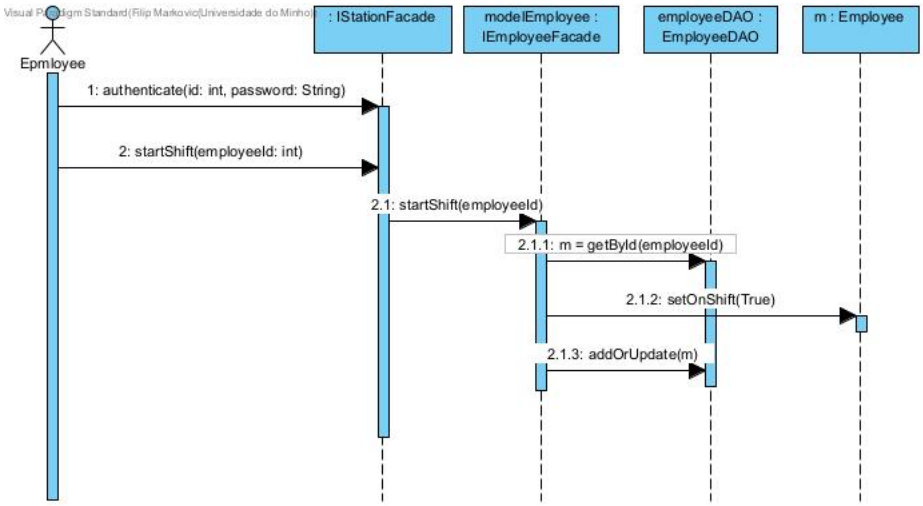


Figure 12: Sequence Diagram of the method startShift

End Shift

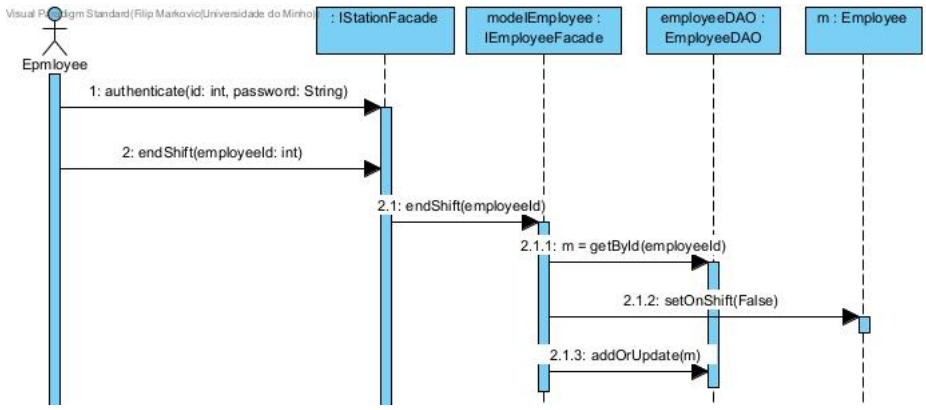


Figure 13: Sequence Diagram of the method endShift

Register Start of Job

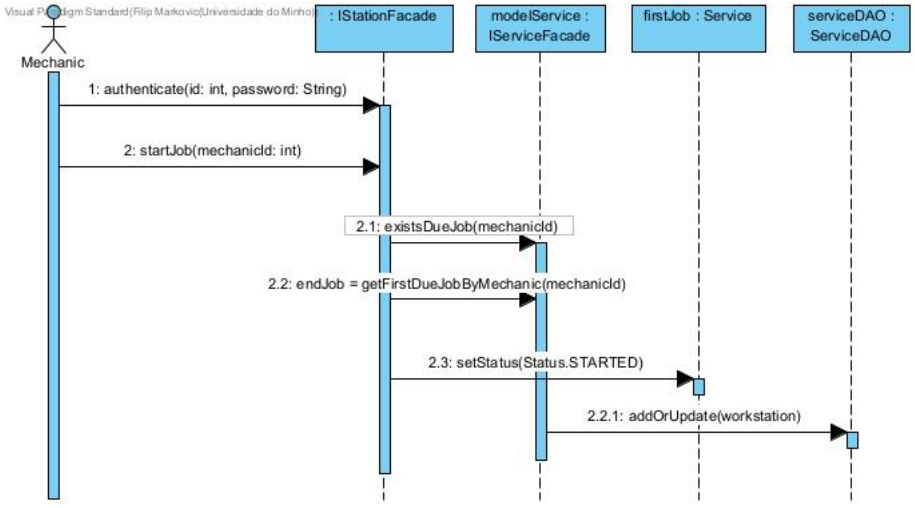


Figure 14: Sequence Diagram of the method startJob

Register End of Job

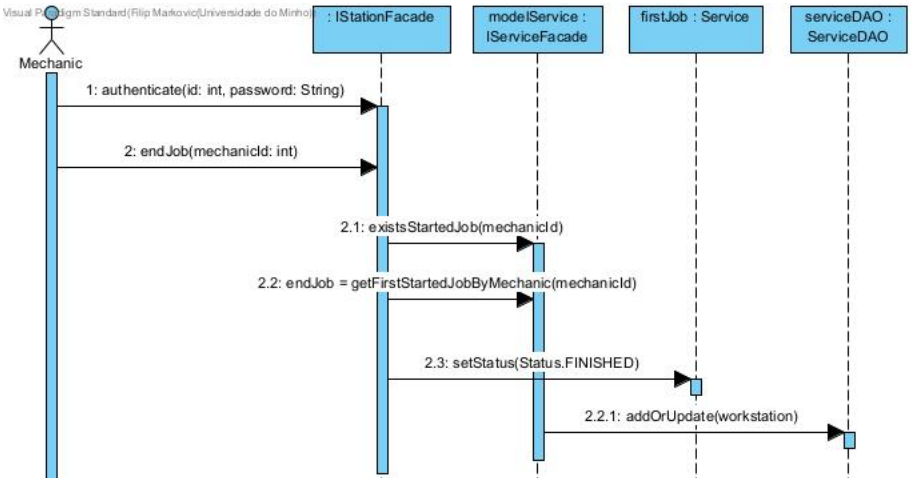


Figure 15: Sequence Diagram of the method endJob

Register Workstation

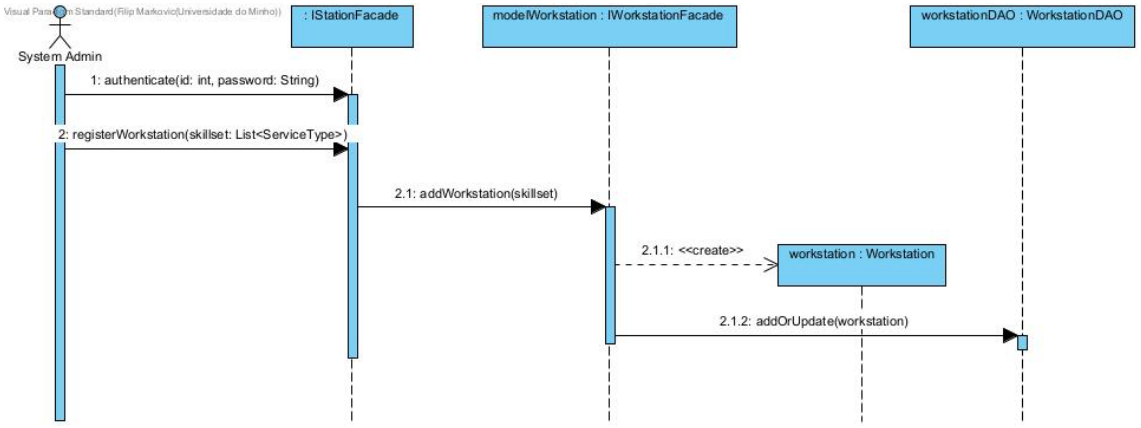


Figure 16: Sequence Diagram of the method registerWorkstation

Schedule Service

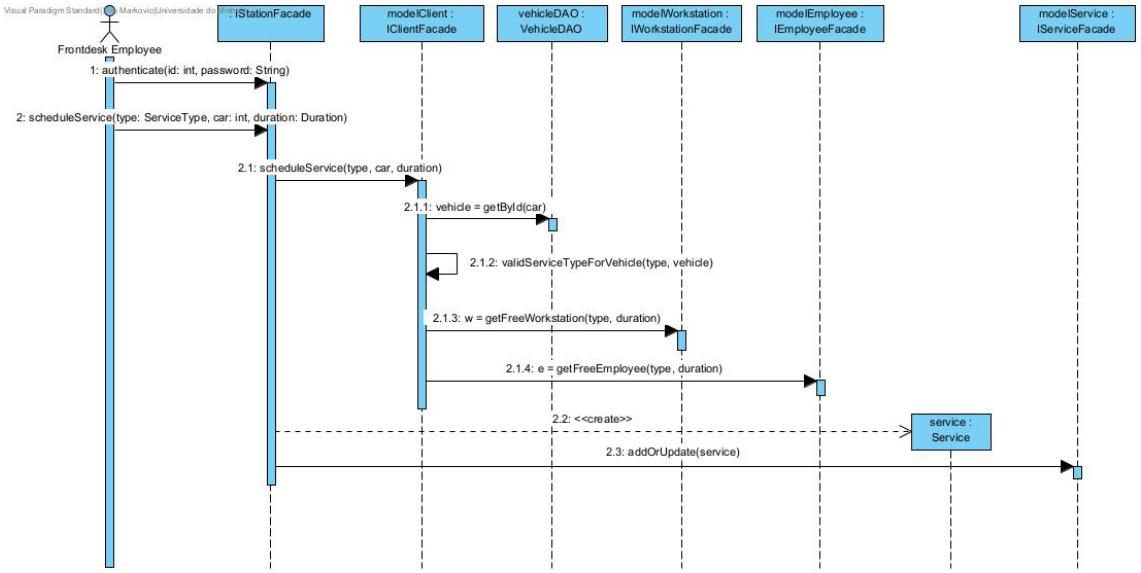


Figure 17: Sequence Diagram of the method scheduleService

Sequence Diagrams – After implementation

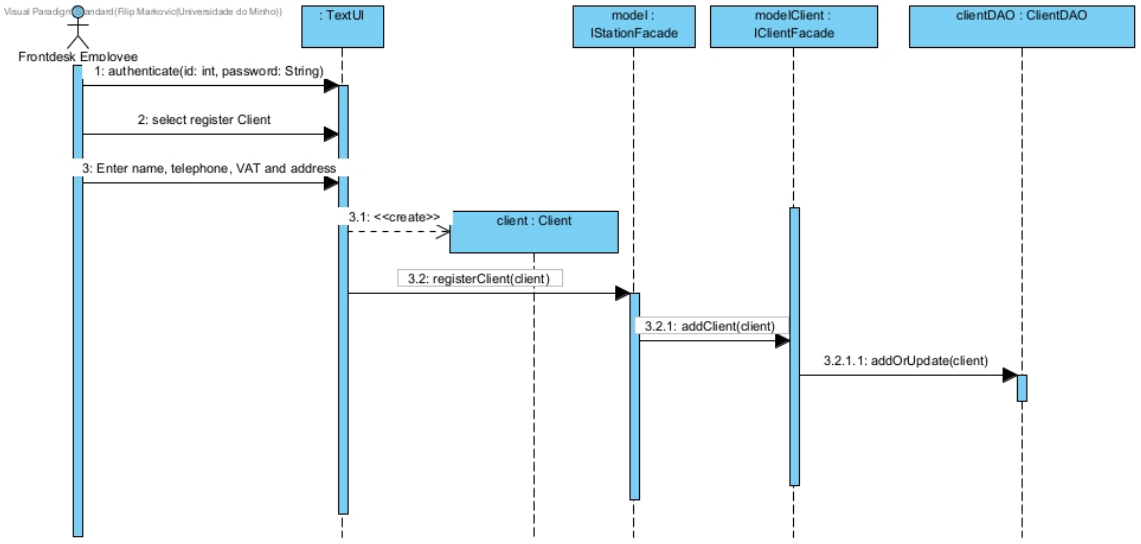


Figure 18: Sequence Diagram of the method registerClient

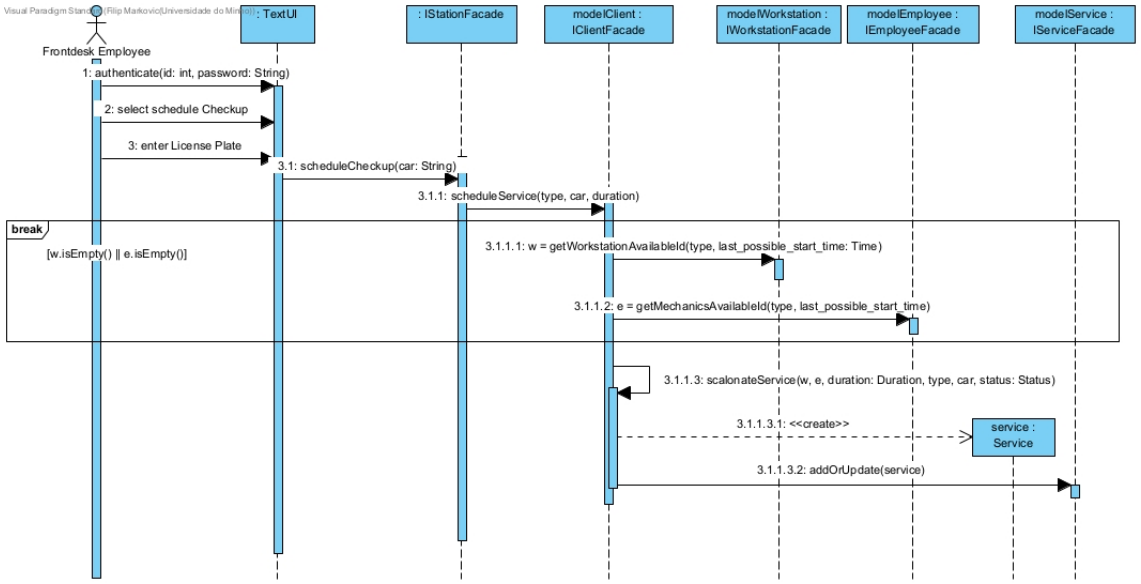


Figure 19: Sequence Diagram of the method scheduleService

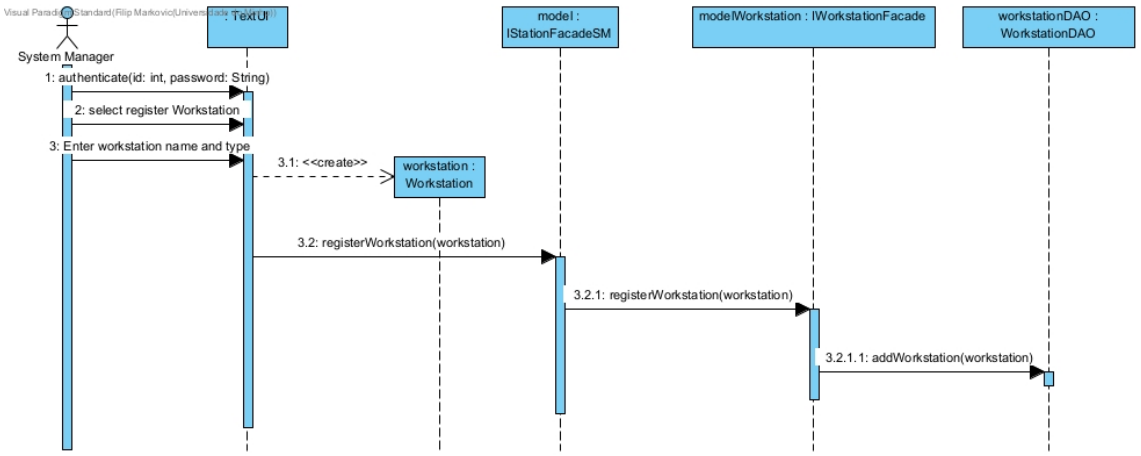


Figure 20: Sequence Diagram of the method registerWorkstation

Class Diagram – After the implementation

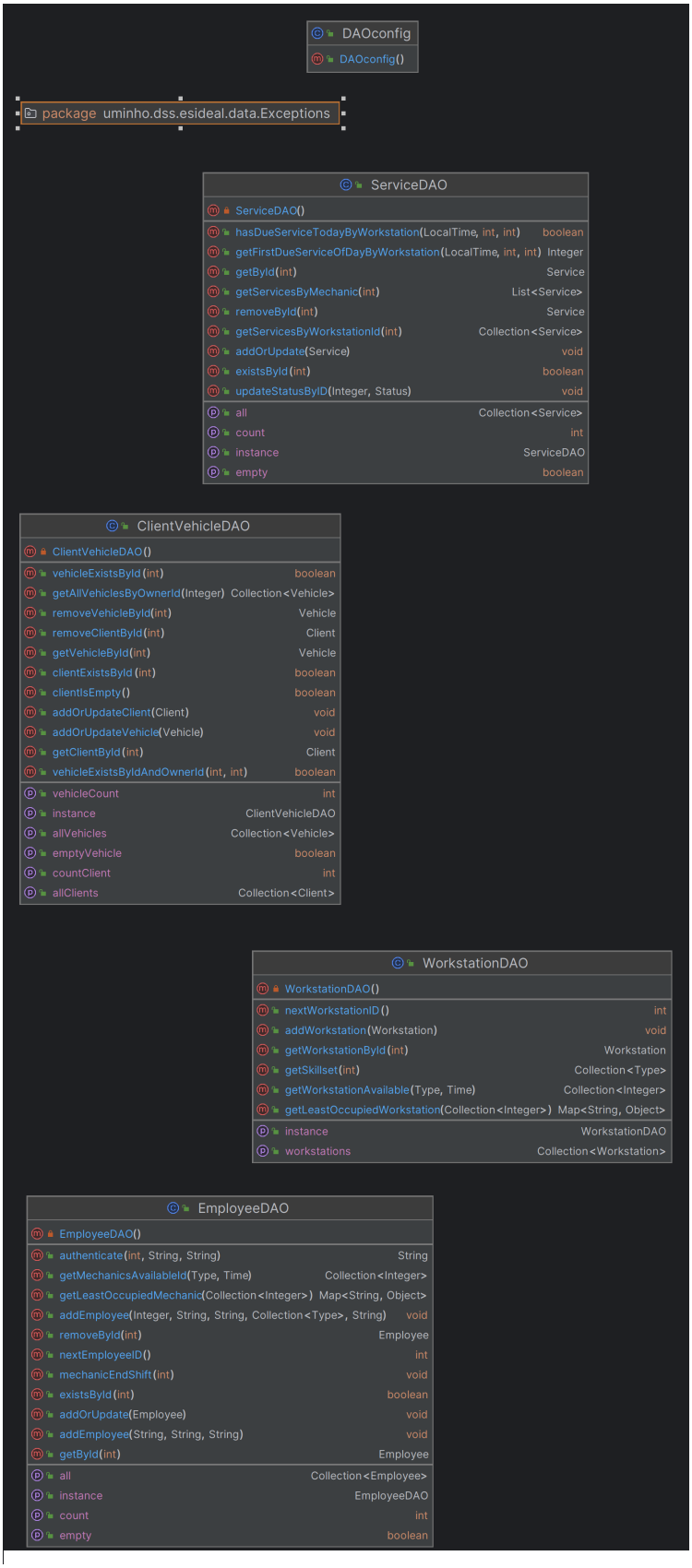


Figure 21: Data package

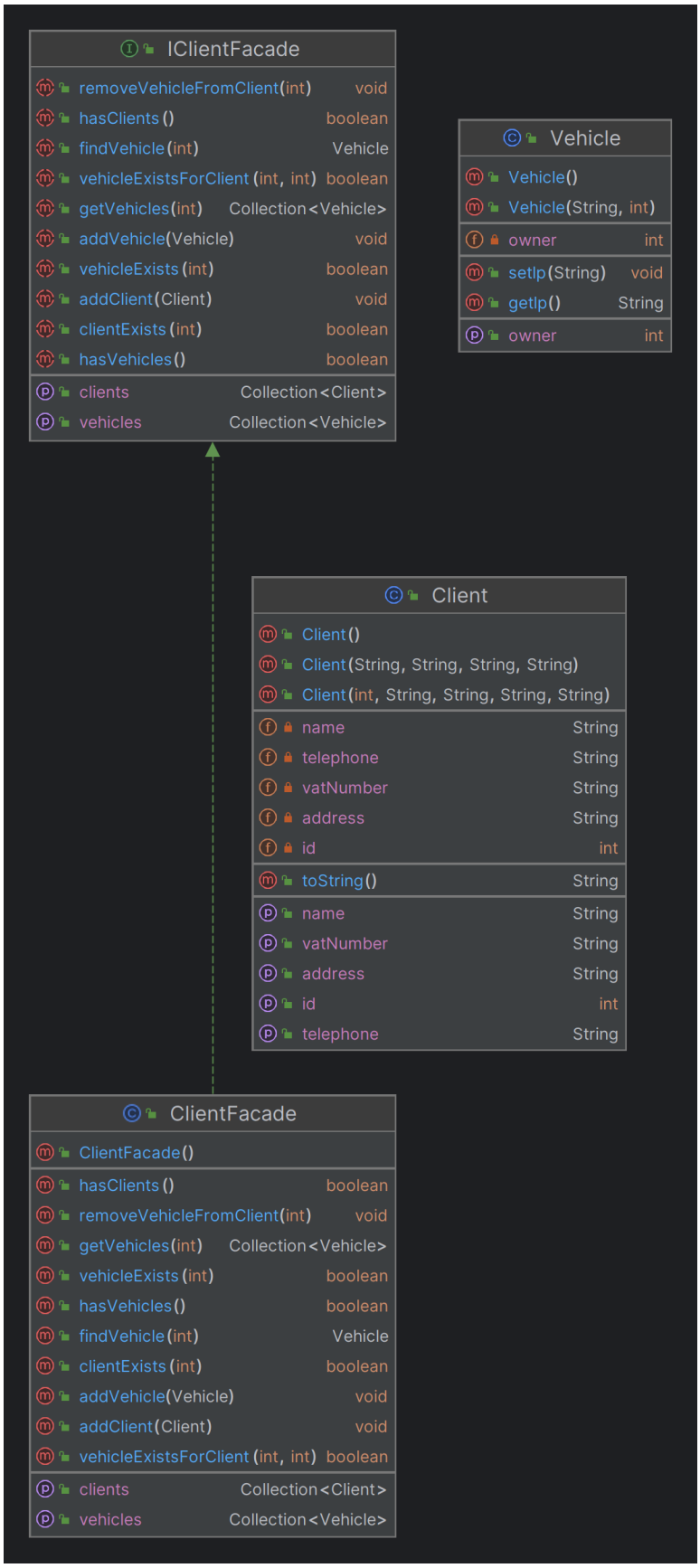


Figure 22: Client

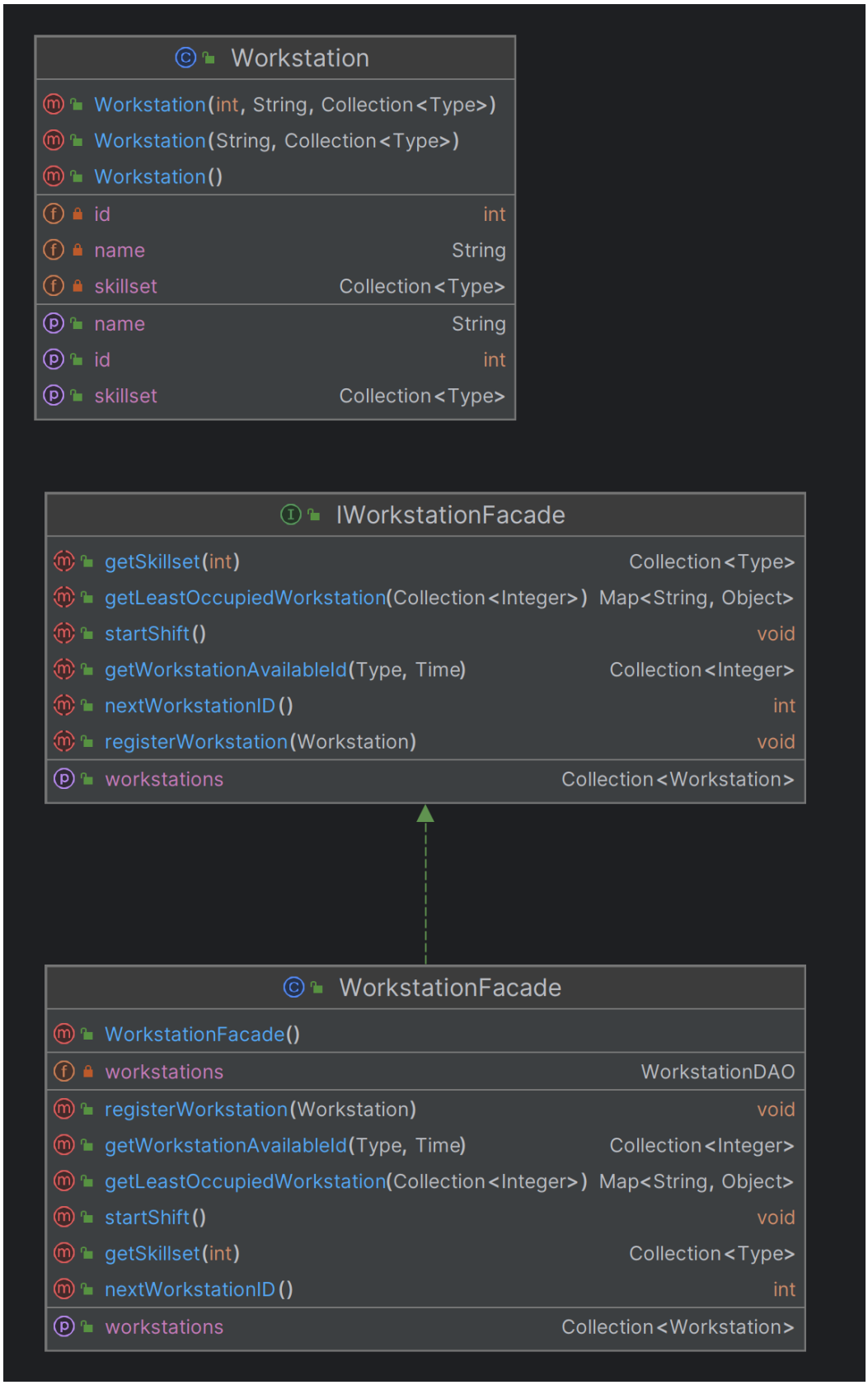


Figure 23: Workstation

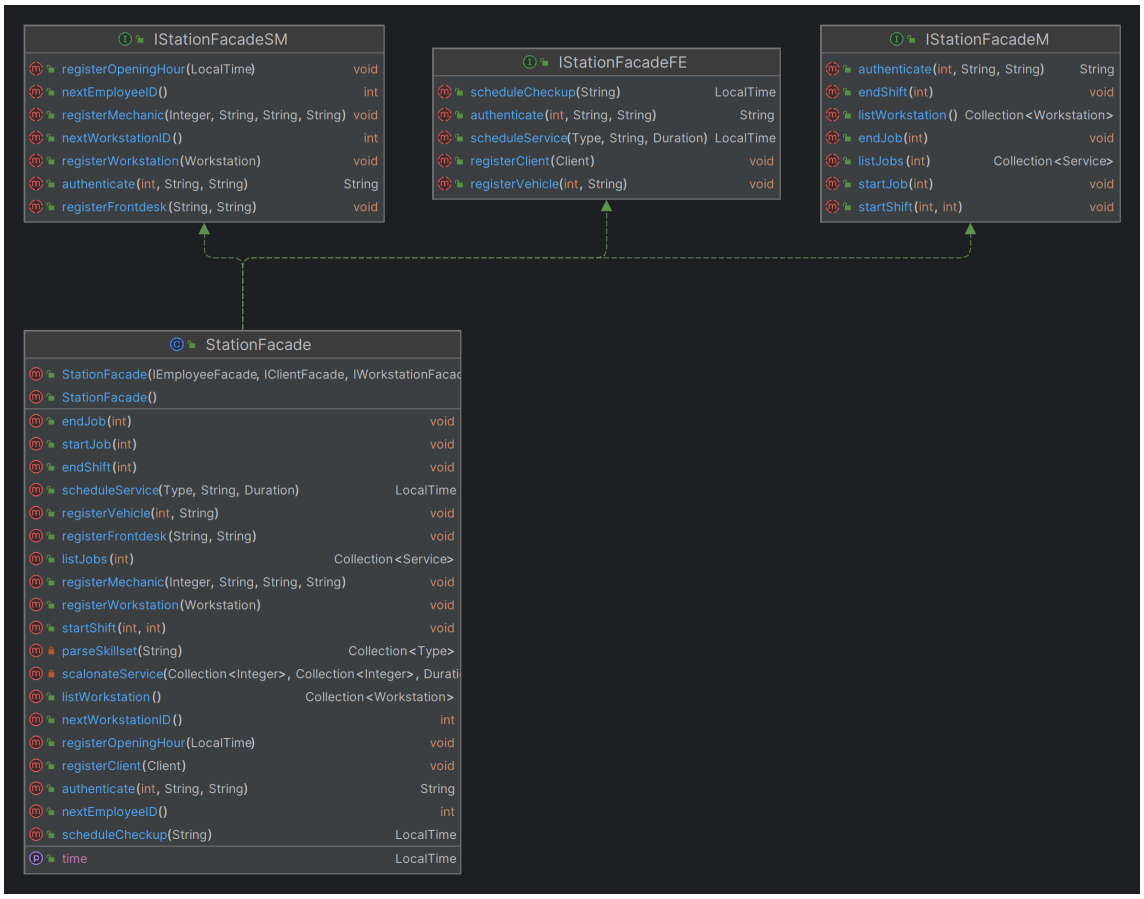


Figure 24: Station

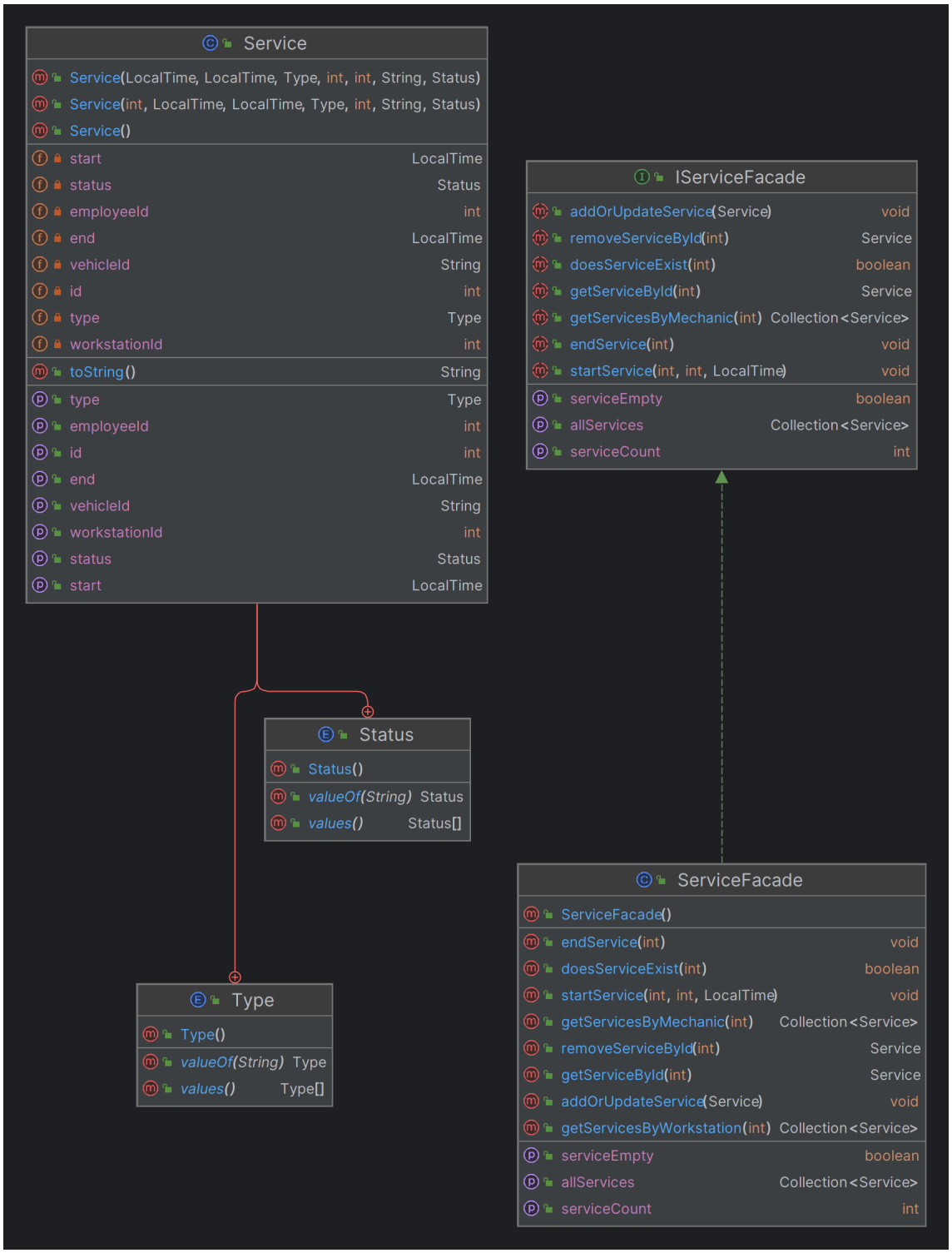


Figure 25: Service

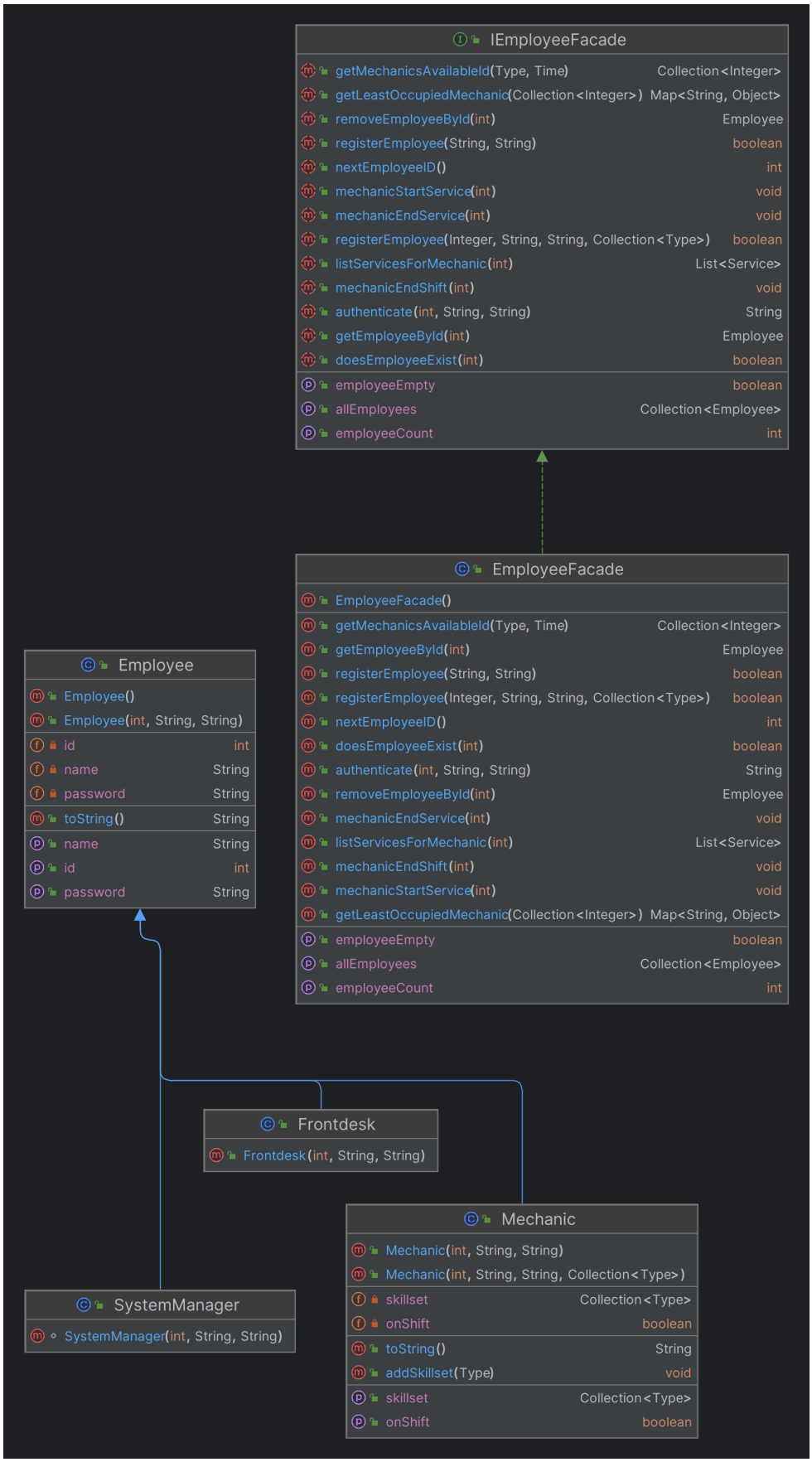


Figure 26: Employee

Conclusion

Even though we severly lacked the time to implement a finished product we are satisfied with the general application architecture and all the diagrams produced from the earlyer planning.
Stepping from the current application state to the full application we planned wouldn't be hard considering the support material and generally clean aproaches we took to tackle the different aspects of the problem.
In restrospect we can only hope that our work in the diagrams is considered and the functionality we did implement shows a glimpse of what could have been.

Note: As time was ticking, we apologize but we were only able to put some images without description