Documentation project

Automobile registry

Students: *Boruz Dumitru Bogdan Mateiş*

Pirlea Costinel Florentin

Application functional requirements

The first requirement for the application is that it is on a web platform, running on a server, preferably using the MVC design pattern.

The application must store 3 types of items on its database with varying attribute types, firstly it must store cars, labeled "**autoturism**" the autoturism item must be able to register the following fields:

- *Car name***,** preferably stored as string
- *Color*
- *Expiration dates* for vignette, ITP and insurance
- *Name* of person entrusted with car
- *Manufacture year*
- *License plate*

The second item in the application representing service, labeled "**automanopera**" must store the *date* when work was done, *total price* and kilometer *turnover* for the serviced car.

The last item, labeled "**autoservice**", has to store details like *name* of part or parts used during service and *price* for that part.

The application should only have one **administrator-type** user, the administrator having the ability to execute **CRUD**(create, read, update, delete) operations on each of the items described earlier. To add an item the administrator should have a button available on the main page that will take him to the form to fill out the item information in, the application should prompt the user if incorrect type of data has been input in the field or invalid characters were used (for example text as a price).

When viewing content(cars, service or parts) they must have the option to edit or delete them and, when editing, the fields of the selected item should be filed out with the current data. Additionally, when viewing a car there must be an option to see if there is any service on it and to car, similarly when viewing service on a car, a means to view and add parts must be present along with option to edit or delete for the existent parts. Another important desired feature is the ability to sort cars by their attributes(name, color, expiration dates etc.).

A final feature requirement is the ability to view a monthly report which will contain prices and total prices for that month for parts, service and parts + service, the viewing report function must also contain information about each car that had service done that month such as name, license plates and date of service. Additionally the view report functionality must be able to view reports from previous months and years if there is data to view.

The application should contain a **navigation bar** in order to facilitate quick movement and operations during execution.

Project Title: Automobile Registry	
Start Date: 23/10/2017	**End Date:** 15/01/2018
Project Manager: Boruz Bogdan	
Project Sponsor: Universitatea din Craiova	
Customer: Popescu Ion	
Users: Car delivery service data managing	
Stakeholders and Expectations: Team: Have ready access to individuals with the authority to make decisions regarding software requirements. Delivery service: Gain an application that will simplify keeping track of a delivery service's cars and work being done on them.	
Purpose (Problem or opportunity addressed by the project): Users who take advantage of the Automobile registry application will have a comfortable way to keep track of data about the cars entrusted to their drivers. Currently there are only inhouse software applications which are not available to anyone looking to manage delivery car service and drivers. The application is also smartphone compatible due to it being hosted online.	
Goals and Objectives: Main goal is giving delivery services a free alternative of keeping data of drivers and the cars entrusted to them. • Providing an user friendly interface. • Provide a portable application.	

Application vision

Automobile registry is an application for both mobile and desktop users who want to keep track of a variety of information about cars. The automobile registry application allows an user to keep evidence of a car's history in the service, costs, current owner, expiration dates of vignettes, insurances and technical verifications.

The application provides a user friendly interface for managing data about cars that have been or are currently in service and the accounting side of a service procedure. Another feature offered by the application is a monthly report function which can be used to track total costs over each month, the application supports a single user type as an administrator which has full authority and is meant to handle the data management(deleting, editing, inserting etc.).

Schedule Information (Major milestones and deliverables):

22/10/2017 – Gather requirements
23/10/2017 – Project Charter Complete
24/10/2017 – Application vision completed
30/10/2017 –Architecture and hierarchy Document Complete
18/11/2017 – Database schema plan Complete
30/11/2017 – Basic use cases Complete
15/12/2017 – Class and Sequence diagrams Complete
13/01/2018 – Test Report Complete

Application architecture and technologies

The **MVC**(model-view-controller) architecture pattern is used for this application, more precisely this architecture is provided by the Spring MVC framework along with components that can be used to create loosely coupled, flexible web applications. As a result of this architecture, different aspects of the application are separated with loose coupling between them, the Model encapsulates the data in POJOs (plain old java objects) while the View renders the model data and generates HTML output for the client's browser, lastly the Controller handles user requests and builds the right model for the view to render.

The entire framework is based on a **DispatcherServlet** which handles HTTP responses and requests, an example of a typical workflow in this framework starts after an HTTP request has been received, using HandlerMapping the DispatcherServlet calls the appropriate controller which then uses the request to call the right methods, the model data is set based on the defined business logic and then the view name is returned to DispatcherServlet. The dispatcher makes use of the ViewResolver to retrieve the defined view, once it is finalized it gets passed the model data and is rendered in the browser. [1]

Another technology the application makes use of is the **Bootstrap**, a free open-source framework exclusively meant for the front-end side of web development. It contains HTML and CSS design templates for typography, forms, buttons, navigation and other interface components, it also has optional JavaScript extensions. This technology is feature rich, flexible and modular. Some of the key features that can be found in it are a set of Stylesheets that provide basic style definitions for key HTML components, giving the elements a modern appearance. Additionally it contains common user interface elements implemented as CSS classes which are to be applied to certain HTML elements in the page. [2]

Since the Bootstrap technology is exclusively concerned with front-end development, it is used to enhance the visual appeal of the View components used in this application.

The final technology used is **Hibernate**, a framework used for object-relational mapping in Java, it handles the interaction with the database using the Java Persistence API.

Application hierarchy components

The application makes use of a high level hierarchy, it has multiple layers between the presentation and the lowest layer which is the database. Thanks to the abstraction of the presentation layer and the UI by consequence, the interface can be optimized in line with a customers expectations and needs in mind. The high level view of the architecture consists of 9 major components:

- DispatcherServlet, the key component which was described earlier, provided by Spring
- View, implemented by developers as JSP pages and sometimes provided by Spring
- Controller, implemented by developers
- Model as POJOs implemented by developers
- Service, Business logic layer with implementation required
- Repository, data access interface to be implemented by developers
- HandlerMapping, provided by Spring framework
- ViewResolver, provided by Spring framework
- Database, provided by the Spring framework

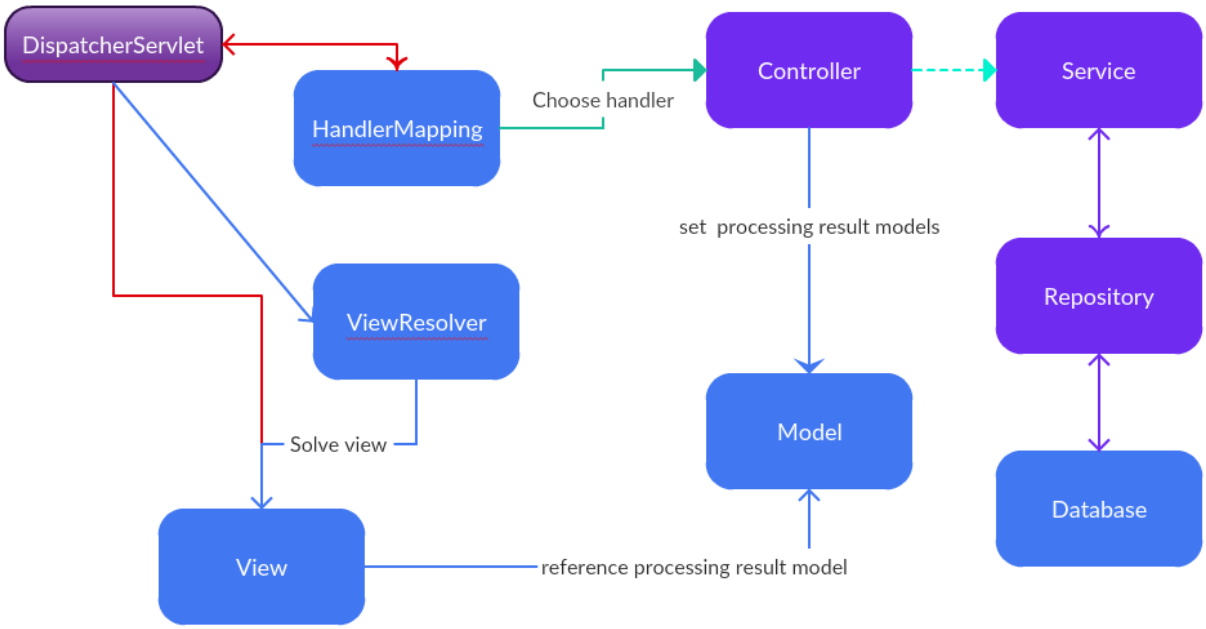

Figure 1 Spring MVC components interaction diagram

Database schema

The database is composed of 3 tables, namely **Automanopera**, **Autoservice** and **Autoturism**. Two of the tables contains a foreign key for another added for the Many to One relationship between them. The schema of the tables and their relationships can be viewed in Figure 2 below.

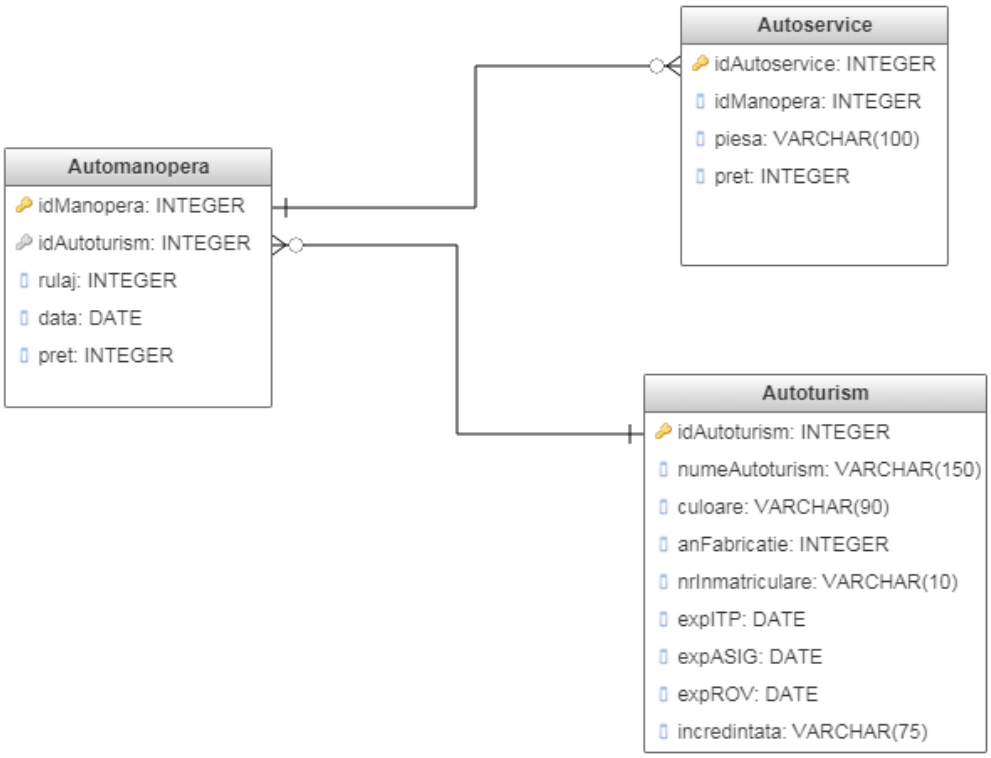

Figure 2 Application Database schema

Automanopera represents the cost, date and kilometer count for each instance when a vehicle suffers changes, it's fields are described as such:

- *ID* field, the primary key
- *idAutoturism*, foreign key used for the many to one relationship with the Autoturism table (for example a car can have multiple procedures done to it)
- *rulaj* field represents the number of kilometers the car had when the repair or workmanship was done on it, this field is especially helpful for parts that are changed after they have been used for long enough – stored as integer
- *data*, the date of the procedure – stored as date
- *pret*, the price for the parts and labor, it is also used to create monthly reports. – stored as integer

Note: all the fields in this table are not nullable.

The **Autoservic*e*** table represents a part used in repairs, it has the following fields:

- *ID* field as primary key
- *ID manopera* field as a foreign key representing the repair in which the part was used, relationship is Many to One since multiple parts can be used for one procedure
- *Piesa* field containing the name of the part – stored as string
- The *Pret* field containing the price of that part. This is the only field in this table which can be null, as a part might be free or provided by customer or manufacturer in some cases

The **Autoturism** table contains multiple data fields about cars and their owner:

- *ID* field as a primary key, also used in the manopera table as a foreign key
- *numeAutoturism* field which holds the car's name, culoare field for color and anFabricatie for manufacturing date – stored as string
- *nrInmatriculare* which contains the car's registry number plates – stored as string
- three date fields. namely *expITP*,*expASIG*,*expROV* for verifications, insurance and vignette in that order – all are stored as dates
- The *incredintata* field contains the name of the person entrusted with the car – stored as string

Note: all the fields in this table are not nullable as well, on delete and edit the application defaults to cascade and removes orphans(for example if a car has been removed all the parts and work done on it gets removed along with it)

Use cases

In this part of the document we will go through each step of the common use cases in this application that the average user will encounter

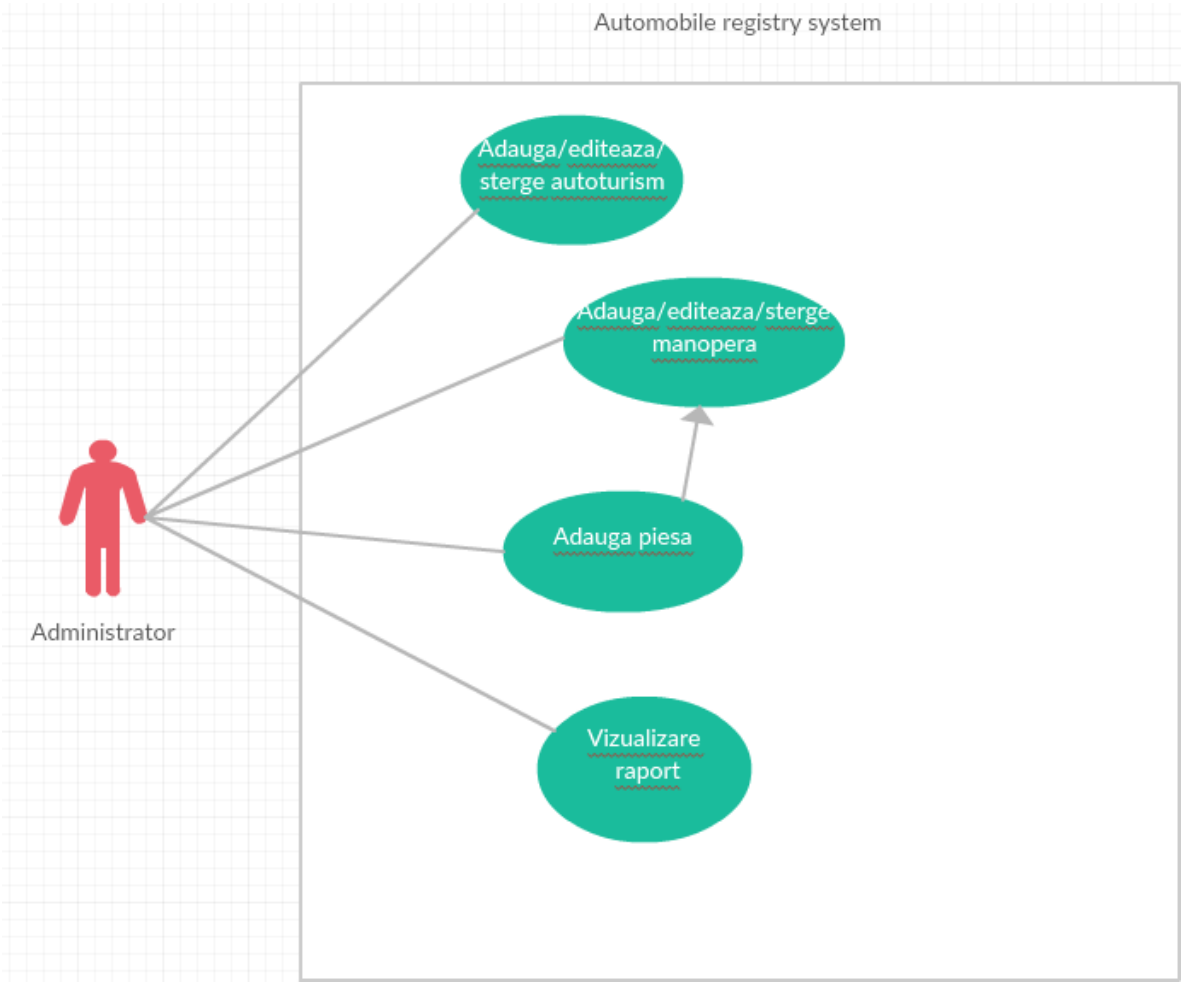

Figure 3 Diagram of use cases

1. **Adding service**

Title	Adauga manopera
Primary Actor	Administrator
Description	Administrator can add manopera to a car and parts to the manopera
Requirements	At least one car has been added
Use frequency	High
Success scenario	• Click on "Adauga manopera" • Fill out information in the form • Click "Adauga manopera" button at bottom of form

In order to add service to a car, firstly there must be a car added to the database, the **administrator** must open the application in the browser by typing it's link in the addressbar(http://localhost/myapp/index); click on "Adauga manopera" located on the topside navigation bar of the application; then the user is then presented with 4 fields a select drop down menu to pick the *car* labeled "Selecteaza autoturism", two number fields in which the *turnover*(labeled "Rulaj") and *price*("Pretul") can be entered and a date("Data"), the administrator then selects the desired car and fills out the remaining information; if the information inserted is correct then the user has successfully added service to that car. An alternate failure scenario here would be if the user does not pick a car or the entered information is of incorrect type or the field is left empty.

2. Adding part

Title	Adauga piesa
Primary Actor	Administrator
Description	Administrator can add parts to the manopera
Requirements	At least one car has been added and a manopera has been made on it
Use frequency	High
Success scenario	• Click on "Adauga piesa" • Select manopera to add parts to • Fill out the rest of the form information • Click "Adauga piesa" button at the bottom of the form

If there is a car with a manopera added to it in the system, the **administrator** can **add parts** to a *manopera* in the following way. After opening the application, click on "Vezi autoturisme" which can be found in the navigation bar at the top of the application interface, after the table with cars loads up, the administrator can click on manopere to view if the car has any, after ascertaining the targeted manopera and car, he must click on "Adauga piese" on the navigation bar which brings up a form containing a drop down select menu, field for part name labeled "Piesa" and a price for the part, in the drop down menu the administrator will select the target manopera seen in the vehicle's manopere table and fill out the remaining fields then click on "Adauga Piesa" to finish the procedure. An alternative scenario can be to go directly to the "Adauga piesa" part of the application and select the targeted manopera if the user already knows the manopera he wants to add parts to.

3. Adding vehicle

Title	Adaugare autoturism
Primary Actor	Administrator
Description	Administrator can add a car to the system
Requirements	None
Use frequency	High
Success scenario	• Click on "Adauga autoturism" • Fill information • Click "Adauga autoturism" button at the bottom of the form

The process of **adding a car** to the service has no underlying requirements; in order for an administrator to achieve this, after opening the application in the browser and clicking on "Adauga Autoturism" in the navigation bar; the user will get prompted with multiple fields which must be filled for data about the car being added, namely: *license plates*, *model and manufacturer name*, *color*, *manufacture year*, *name* of person entrusted with the car and expiration dates for insurance vignette and ITP. After the fields have been filled by the administrator, to finalize the operation the "Adauga autoturism" button under the form must be clicked.

4. Viewing monthly report

Title	Vizualizare raport lunar
Primary Actor	Administrator
Description	Administrator can view monthly report for total costs on each car and in total
Requirements	Cars have had manopera added to them in current month
Use frequency	Medium
Success scenario	• Click on "raport lunar" • Select month when cars have had manopera added to them

In order to view the **monthly report**, the application must have something to show first(for example at least one car with manopera added since the application has been used); if the conditions are met the administrator can click on "Vizualizare raport" found in the navigation bar; this will bring up the monthly report table; if the administrator wants to see a different month he can click on the drop down menu labeled "Luna" found above the table or see reports from previous years months by using the "Anul" drop down menu.

5. Editing service

Title	Editeaza manopera
Primary Actor	Administrator
Description	The administrator can modify data in a manopera
Requirements	At least one manopera has been added to a car
Use frequency	Medium
Success scenario	• Click on "Vezi Service" • Click on "Editeaza" button on manopera row • Modify information • Click on "Update Manopera"

If the administrator wants to edit an existing **manopera,** the first thing to do after opening the application is to click on "Vezi Service" which will prompt a table containing the car license plates as an identifier and various other data regarding the manopera, for editing a manopera, the administrator can click on "Editeaza" in its row, this will change the page to a form with the manopera's information similar to the "Add manopera" form, after the values in the field have been adjusted according to the administrator's wish, in order to finish the procedure the user can click on the "Update manopera" button under the form.

The most typical failure scenario is entering the wrong type of data in any of the fields or leaving fields which are not nullable empty.

6. Editing vehicle

Title	Editare autoturism
Primary Actor	Administrator
Description	Administrator can edit a car from the system
Requirements	A car has been added already
Use frequency	Medium
Success scenario	• Click on "Vezi autoturisme" • Click on "Editeaza" in the autoturism row • Modify information • Click "Editeaza autoturism" button at the bottom of the form

The only necessity for the e**diting car** use case is that a car is added in the system already, if that is met, the first step for the administrator to start this procedure is clicking on "Vezi autoturisme" in the navigation bar, after finding the vehicle to be edited the administrator can click on "Editeaza" to move to the next step, this will bring up the edit form containing the vehicle's information in all its fields, the administrator can modify the desired information in each field, after that is done he must click on "Editeaza autoturism" to finish the operation, if no errors have occurred due to the data in the fields, he will be returned to the "Vezi autoturisme" page and see the edited vehicle there

Class diagrams

Below is a diagram which highlights the repository and service pattern:

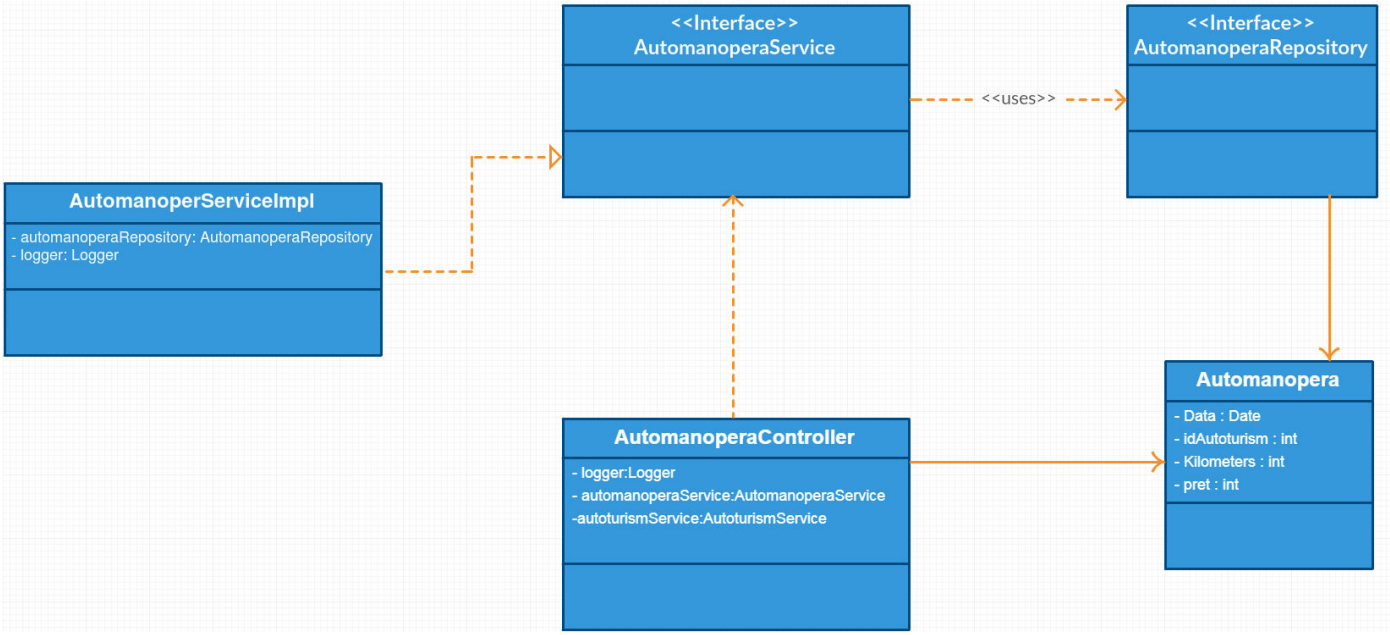

Figure 4 Automanopera model and controller diagram

Not pictured here is the transfer object pattern also used in the application, it's relations with the items in Figure 4 can be seen in figure 5. This class simply adds multiple values at once to the model to minimize calls to the service.

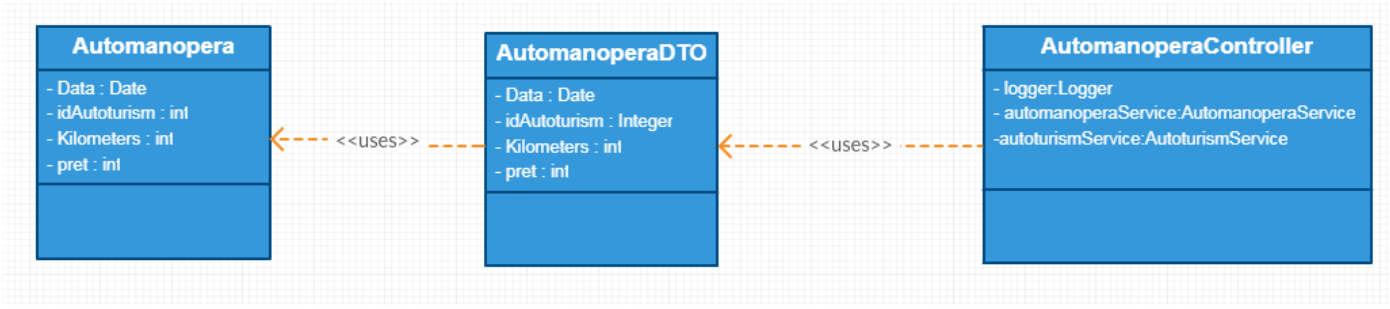

Figure 5 Transfer object pattern diagram from application classes

The other classes are structured in a similar manner using the same patterns.

The figure below illustrates the relationships between entities in the database using an UML diagram.

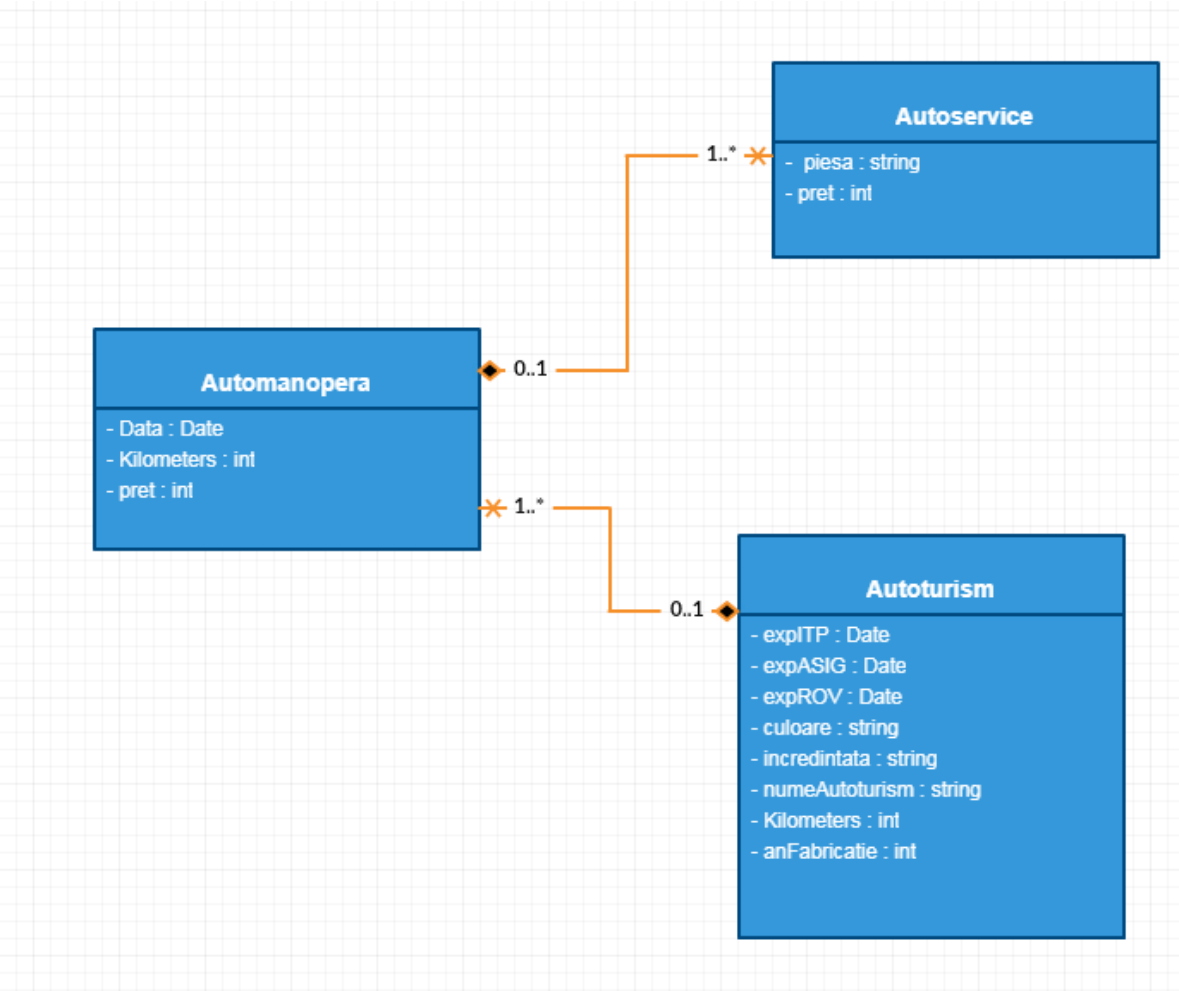

Figure 6 Entity relationship diagram

Sequence diagrams

Below there are some sequence diagrams which show how the application methods get called during runtime in a few common scenarios.

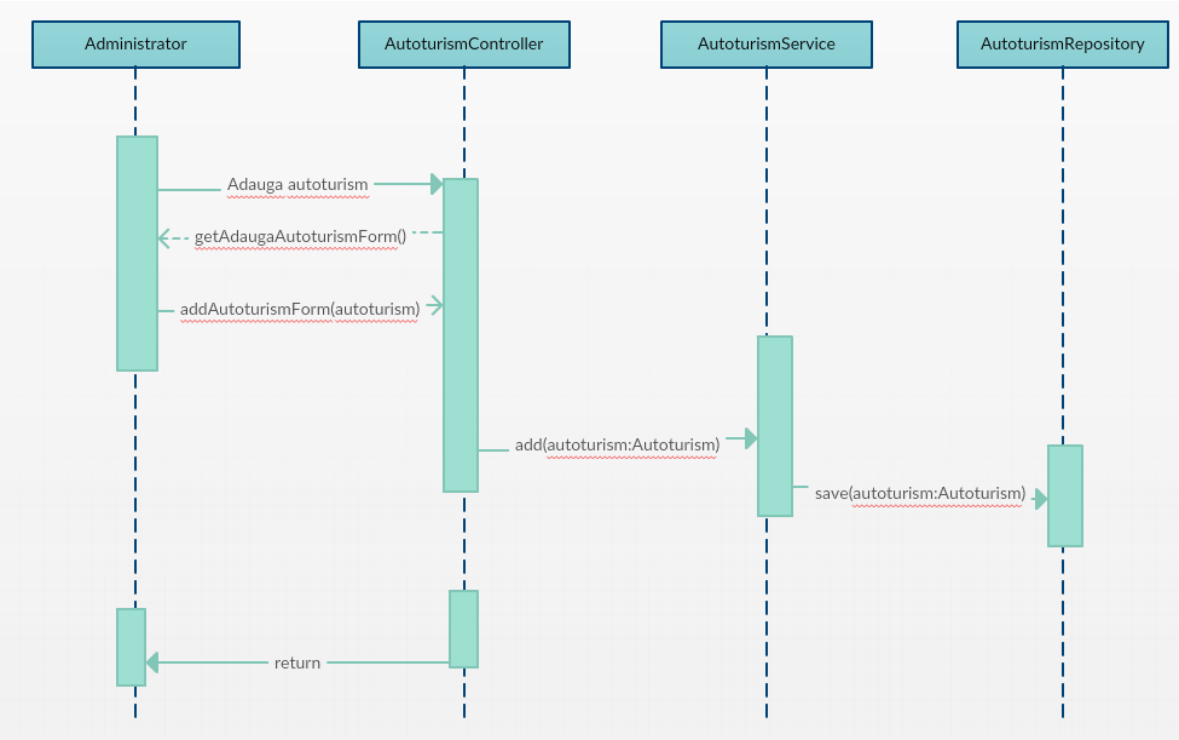

Figure 7 Adding car sequence diagram

In the diagram from figure 7 we can see what methods are called when a vehicle is added to the system.

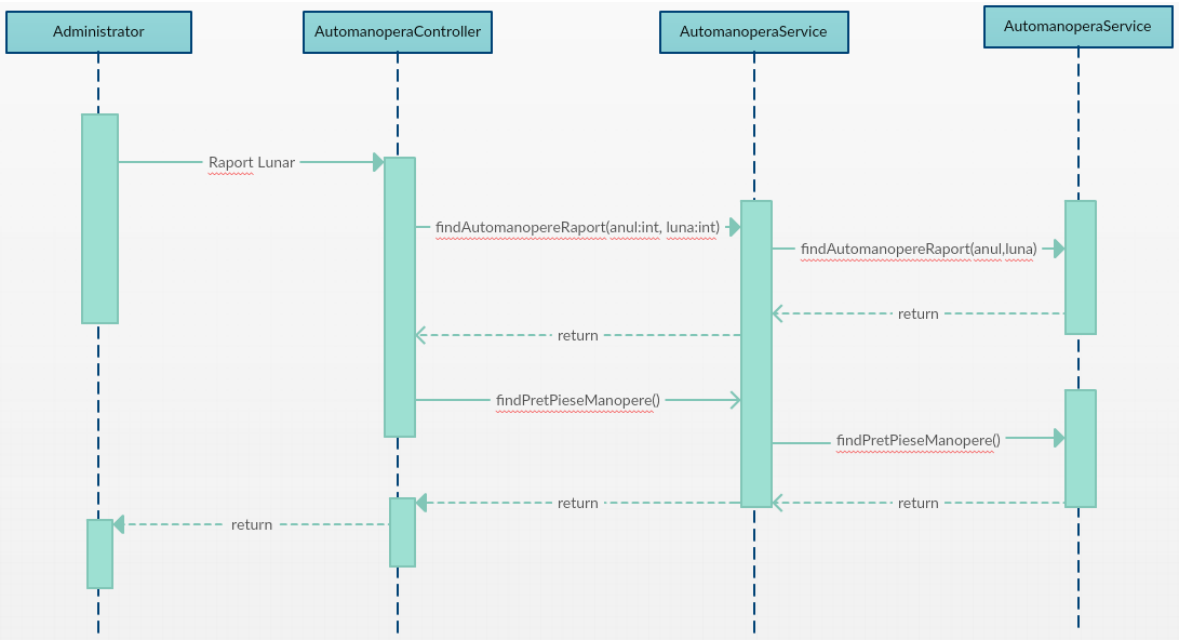

Figure 8 Viewing monthly report sequence diagram

In the diagram above(figure 8) we can see the method calls when a monthly report is requested by an administrator.

Bibliography

1. Model view controller – 3.01.2018

   https://en.wikipedia.org/wiki/Model-view-controller

2. Bootstrap - 4.01.2018

   https://en.wikipedia.org/wiki/Bootstrap_(front-end_framework)