Documentation project

Automobile registry

Students: *Boruz Dumitru Bogdan Mateiş*

*Pirlea Costinel Florentin*

**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.

|  |  |
| --- | --- |
| **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 service accounting | |
| **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 accounting data for drivers and the cars entrusted to them.   * Providing an user friendly interface. * Provide a portable application. | |

**Schedule Information (Major milestones and deliverables):**

22/10/2017 - Gather requirements  
23/10/2017 - Project Charter Complete

30/10/2017 – Basic use cases Complete

15/11/2017 - Release Plan completed

18/11/2017 – Project Plan Complete  
30/11/2017 - Prototype Complete

15/12/2017 - Architecture Document 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 Modelencapsulates 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**

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

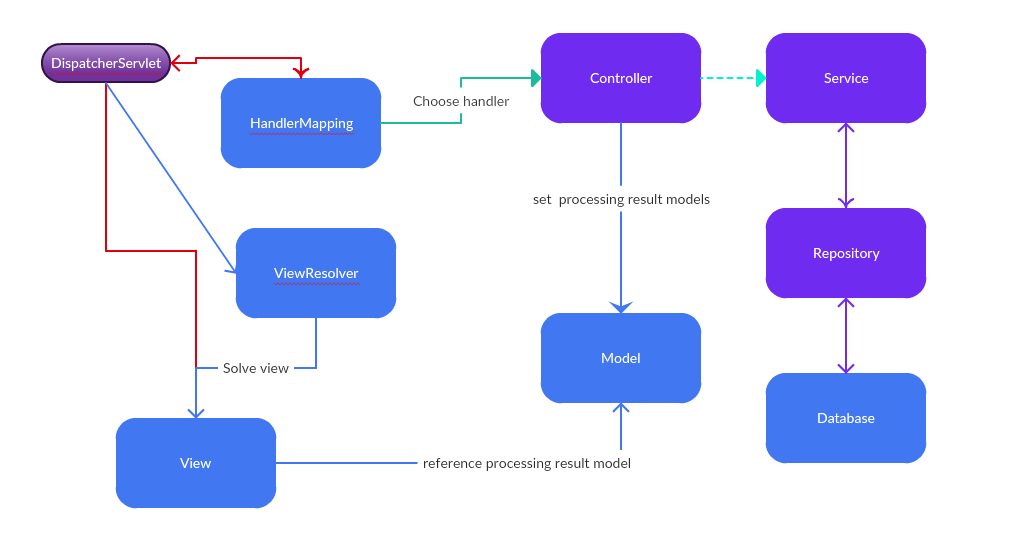


Figure 1 Hierarchy components

**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.

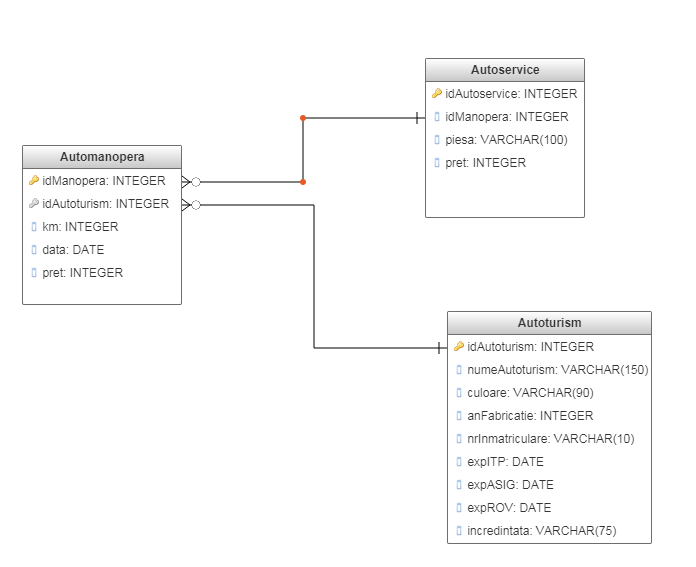


Figure 2 Database schema

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)
* km 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 Autoservice 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 fields for expiration dates 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

**Application functionality**

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.).

**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

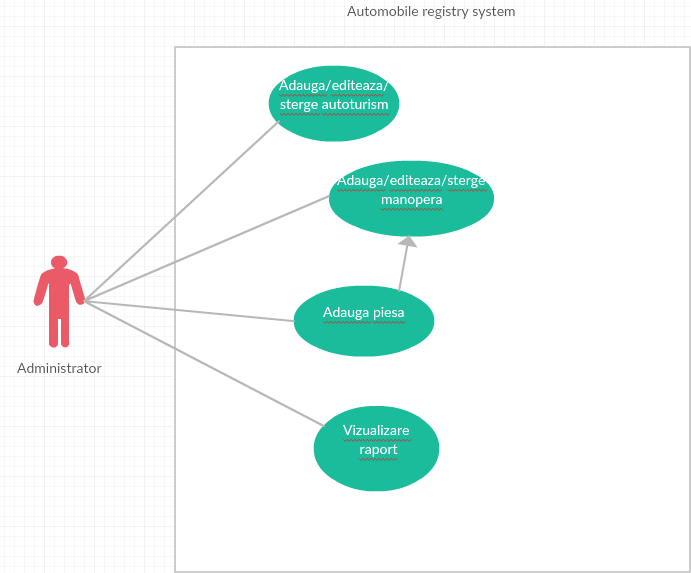


Figure 3 Diagram of use cases

1. **Adding manopera**

|  |  |
| --- | --- |
| **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 |

1. **Adding piesa**

|  |  |
| --- | --- |
| **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 |

1. **Adding autoturism**

|  |  |
| --- | --- |
| **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 |

1. **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 |

1. **Editing manopera**

|  |  |
| --- | --- |
| **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 Edit button on manopera row * Modify information * Click on Update Manopera |

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.

**Class diagrams**

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

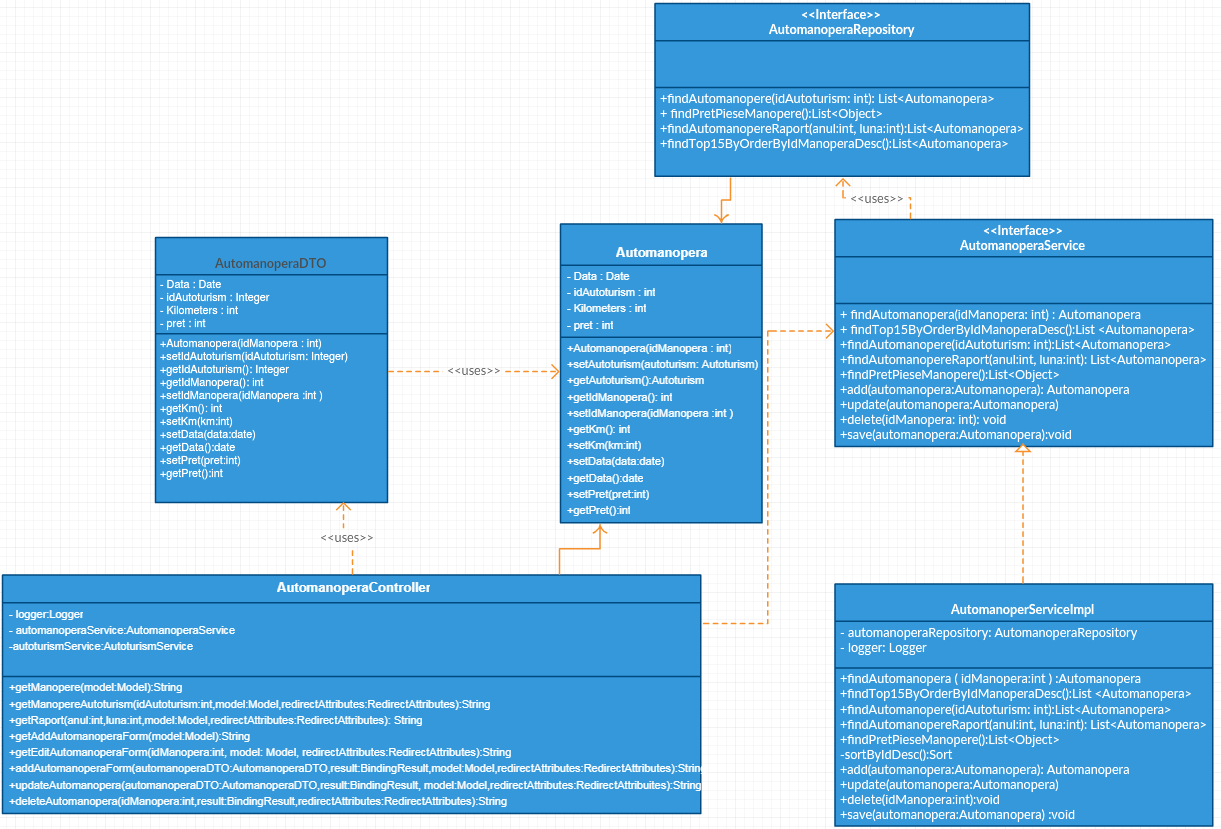


Figure 4 Automanopera model and controller diagram

The AutomanoperaDTO object is the implementation of the Transfer Object pattern which helps pass multiple attributes at once to the server.

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.

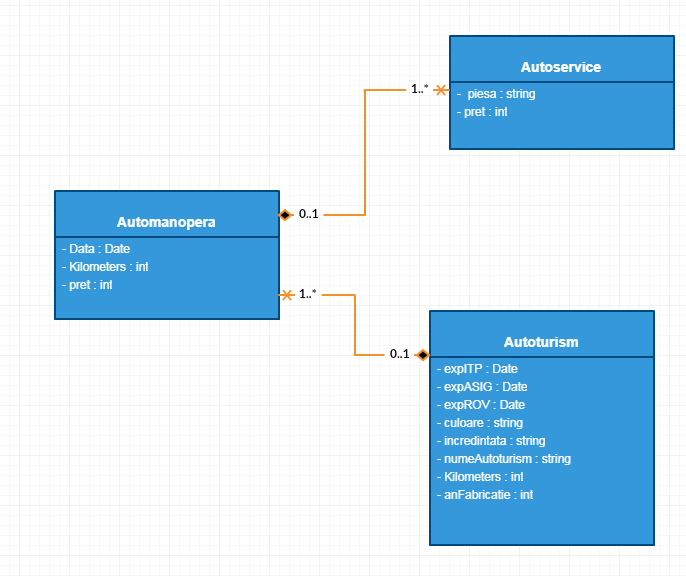


Figure 5 Entity relationship diagram

**Sequence diagrams**

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

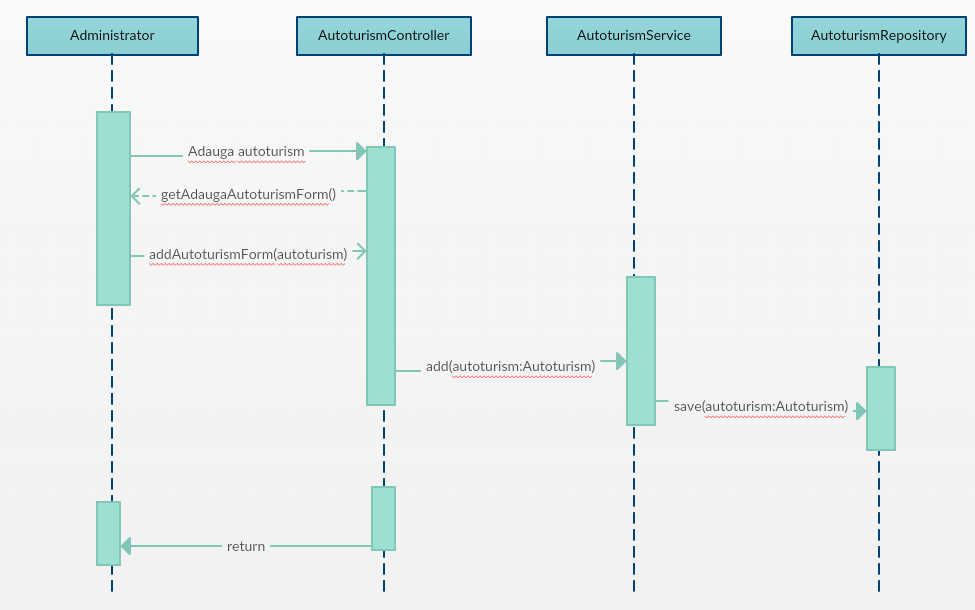


Figure 6 Adding car sequence diagram

In the diagram from figure 6 we can see what methods are called when a vehicle is added to the system.

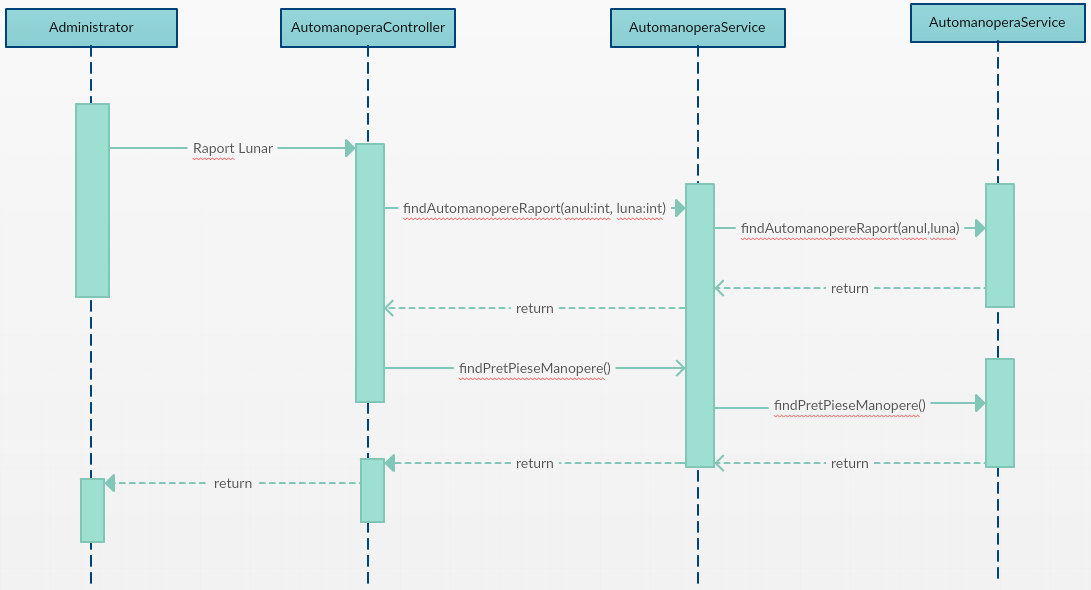


Figure 7 Viewing monthly report sequence diagram

In the diagram above(figure 7) 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>

1. Bootstrap - 4.01.2018

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