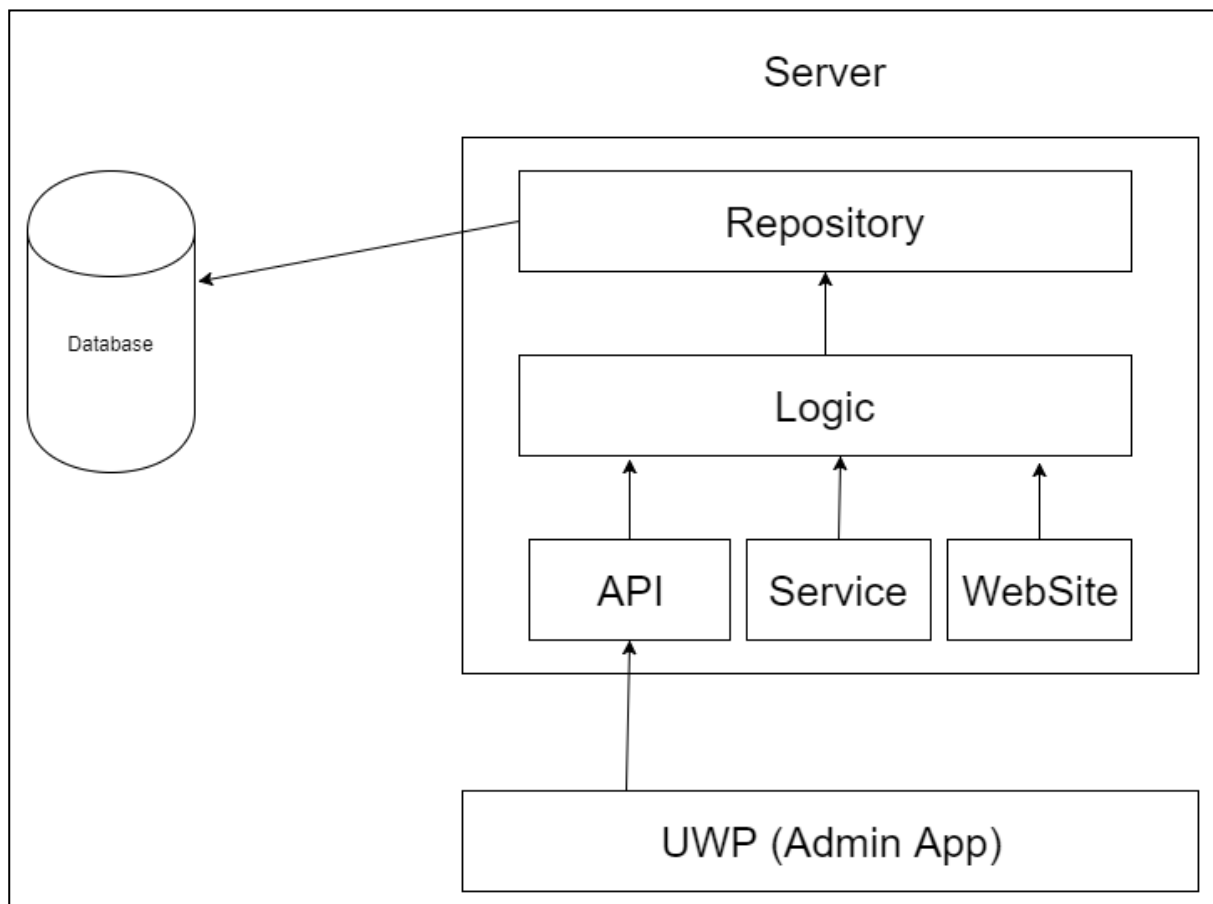


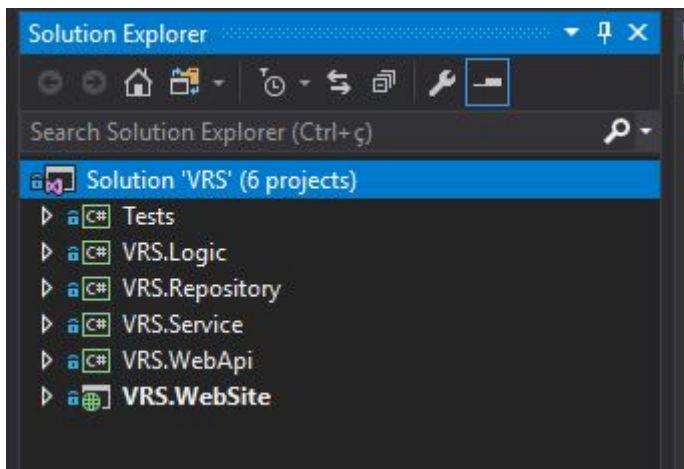
VRS Documentation

The Vehicle Renting System (VRS) is a system that allows the user to make a reservation for a vehicle to rent. The user uses a website to make the reservation and the admin can manage this information through an admin desktop app. The overall structure of the system is represented in the image below.

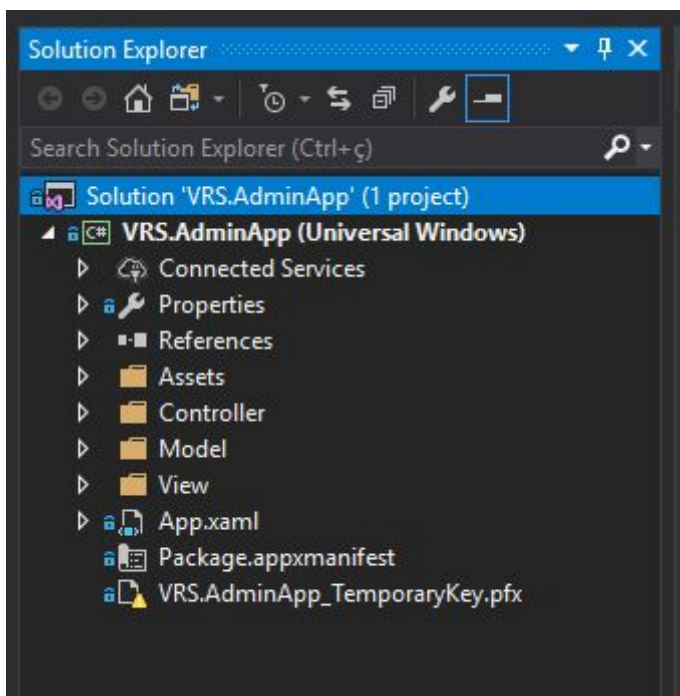


The development environment is divided in two solutions: **VRS** and **VRS.AdminApp**.

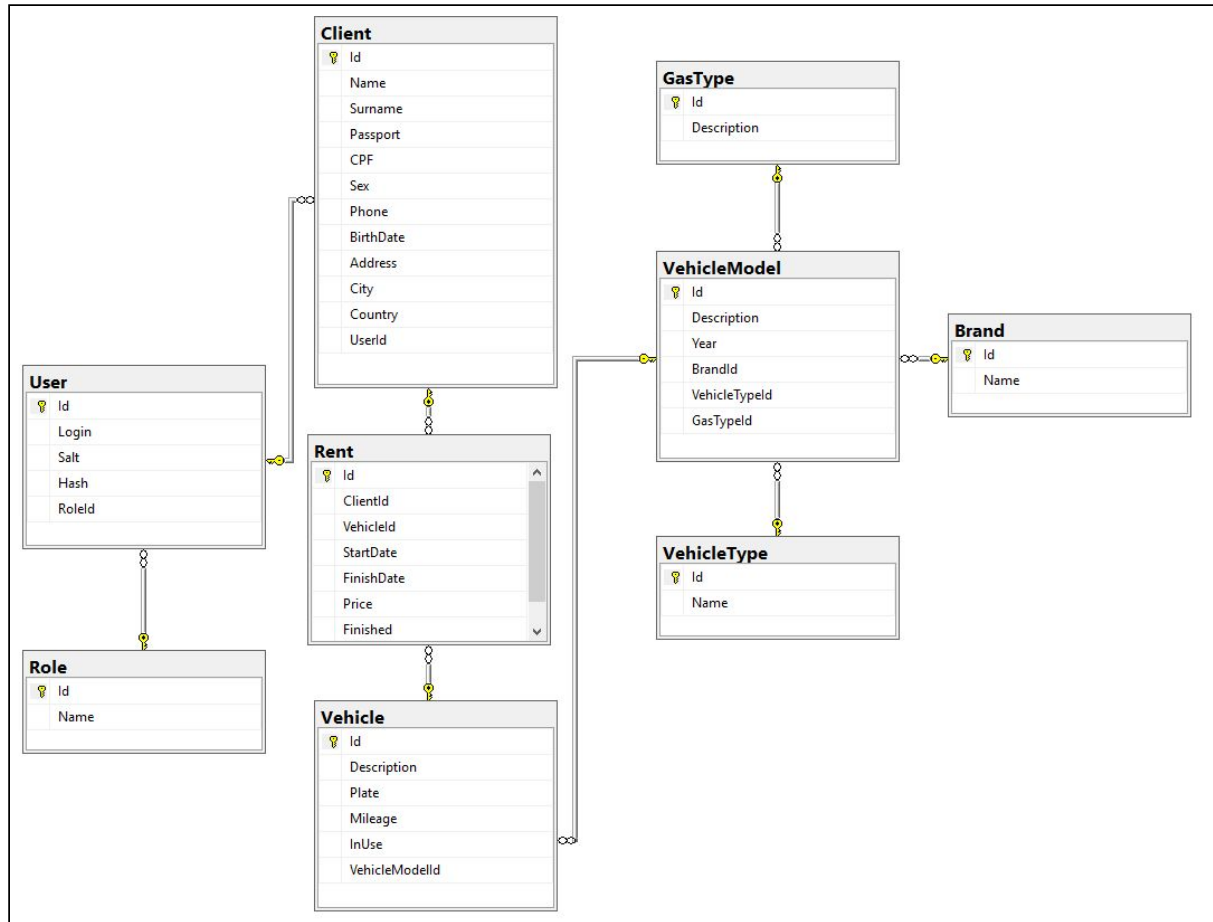
- **VRS:** Contains the visual studio's project for the website (VRS.Website), business code (VRS.Logic), repository and database code (VRS.Repository), background service (VRS.Service) and the web api (VRS.WebApi). There is also a console application project that is used to run and test some code snippet.



- **VRS.AdminApp:** Contain a single project for the admin desktop app.



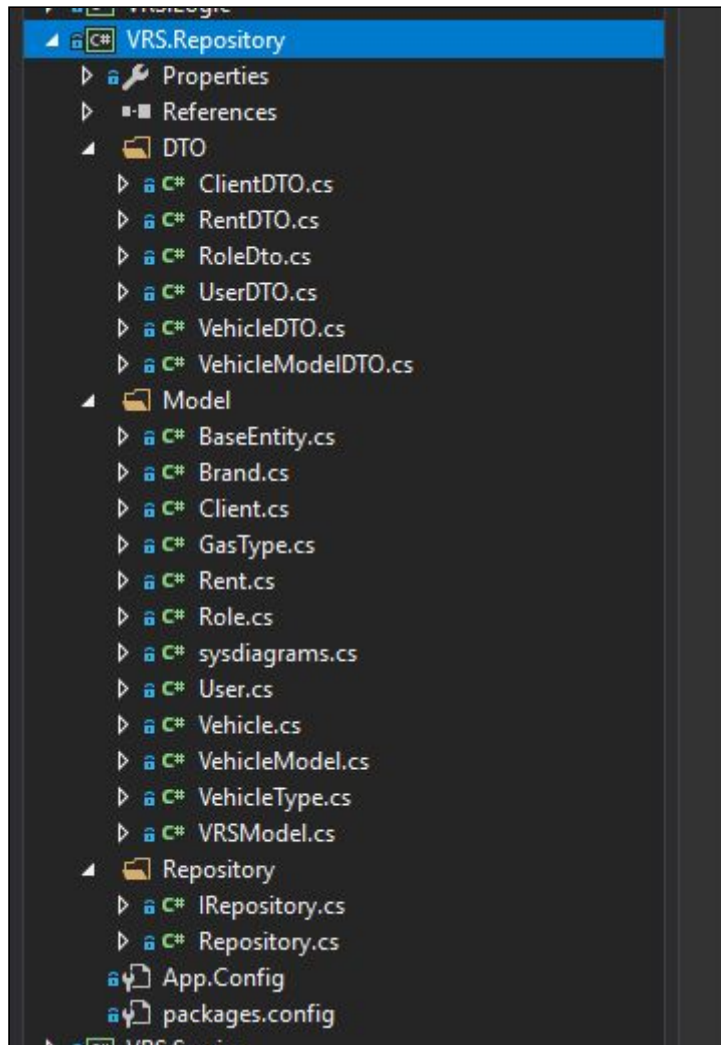
The system uses a relational database (Microsoft SQL Server) to store all data and the database diagram is the one below.



Inside the main folder (“.\VehicleRentSystem\VehicleRentSite”) there are two .sql files, **database.sql** and **inserts.sql**. The first file is responsible for the creation of the tables and the second file has the commands to insert the initial data for the database. They should be executed before executing any of the projects.

VRS.Repository

This is a dll library project that is responsible for the actions involving any data manipulation in the database.



It contains three folders: **DTO**, **Model** and **Repository**:

- **DTO:** Here are the Data Transfer Objects. These classes are used in the connection between the main system with the admin app through the VRS.WebApi projects. More details of this will be given.
- **Model:** These are the classes that represents the database tables. All of these classes inherits from Base Entity which is a simple abstract class that has the Id property. All of new models should inherit from this abstract class.

```

namespace VRS.Model
{
    [DataContract]
    11 references | Matheus N. Nienow, 30 days ago | 1 author, 2 changes
    public abstract class BaseEntity
    {
        [DataMember]
        15 references | Matheus N. Nienow, 30 days ago | 1 author, 2 changes | 0 exceptions
        public int Id { get; set; }
    }
}

```

- **Repository:** This folder contains the Repository class, which implements the IRepository interface. This is a generic class that works with all children of BaseEntity, it contains the main methods needed by the system.

```

namespace VRS.Model.Repository
{
    6 references | Matheus N. Nienow, 33 days ago | 1 author, 2 changes
    public interface IRepository<T>
    {
        5 references | Matheus N. Nienow, 33 days ago | 1 author, 1 change | 0 exceptions
        int Insert(T entity);
        2 references | Matheus N. Nienow, 33 days ago | 1 author, 1 change | 0 exceptions
        int Insert(ICollection<T> entities);
        1 reference | Matheus N. Nienow, 33 days ago | 1 author, 1 change | 0 exceptions
        int Delete(T entity);
        1 reference | Matheus N. Nienow, 33 days ago | 1 author, 1 change | 0 exceptions
        int Delete(ICollection<T> entities);
        4 references | Matheus N. Nienow, 33 days ago | 1 author, 1 change | 0 exceptions
        int Update(T entity);

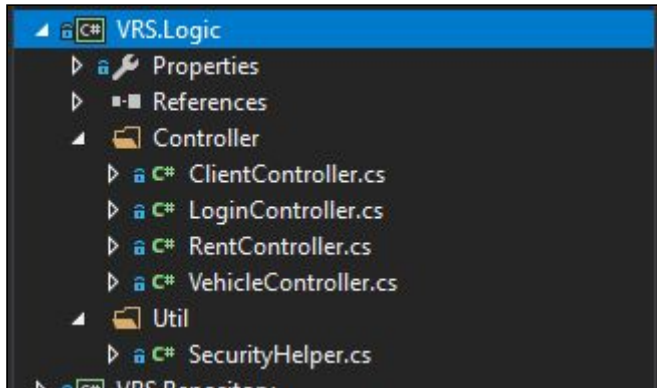
        7 references | Matheus N. Nienow, 33 days ago | 1 author, 1 change | 0 exceptions
        IQueryable<T> SearchFor(Expression<Func<T, bool>> predicate);
        6 references | Matheus N. Nienow, 33 days ago | 1 author, 1 change | 0 exceptions
        IQueryable<T> GetAll();
        2 references | Matheus N. Nienow, 33 days ago | 1 author, 1 change | 0 exceptions
        IQueryable<T> GetAll(params Expression<Func<T, object>>[] navigationProperties);
        5 references | Matheus N. Nienow, 33 days ago | 1 author, 1 change | 0 exceptions
        T GetById(int id);
        1 reference | Matheus N. Nienow, 33 days ago | 1 author, 1 change | 0 exceptions
        void Dispose();

        1 reference | Matheus N. Nienow, 33 days ago | 1 author, 2 changes | 0 exceptions
        IQueryable<T> GetItems(Expression<Func<T, bool>> predicate, params string[] navigationProperties);
        2 references | Matheus N. Nienow, 33 days ago | 1 author, 2 changes | 0 exceptions
        IQueryable<T> GetItems(Expression<Func<T, bool>> predicate, params Expression<Func<T, object>>[] navigationProperties);
        1 reference | Matheus N. Nienow, 33 days ago | 1 author, 1 change | 0 exceptions
        IQueryable<T> Include(Expression<Func<T, object>> predicate);
    }
}

```

VRS.Logic

This is a dll library project that holds all the business logic of the system. Every access to the repository should pass through the controllers of this project.



This project contains two folders: **Controller** and **Util**.

- **Controller:** contains all the controllers for the entities.
- **Util:** contain the SecurityHelper class, which is responsible for the methods used in the hashing of passwords. This class is used both to create a user and to verify a login.

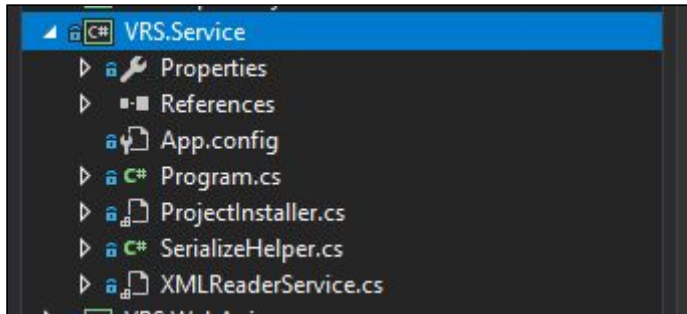
```
namespace VRS.Logic.Util
{
    3 references | Matheus N. Nienow, 33 days ago | 1 author, 1 change
    public class SecurityHelper
    {
        2 references | Matheus N. Nienow, 33 days ago | 1 author, 1 change | 0 exceptions
        public static byte[] HashPassword(string password, byte[] salt)...

        1 reference | Matheus N. Nienow, 33 days ago | 1 author, 1 change | 0 exceptions
        public static byte[] GenerateSalt()...

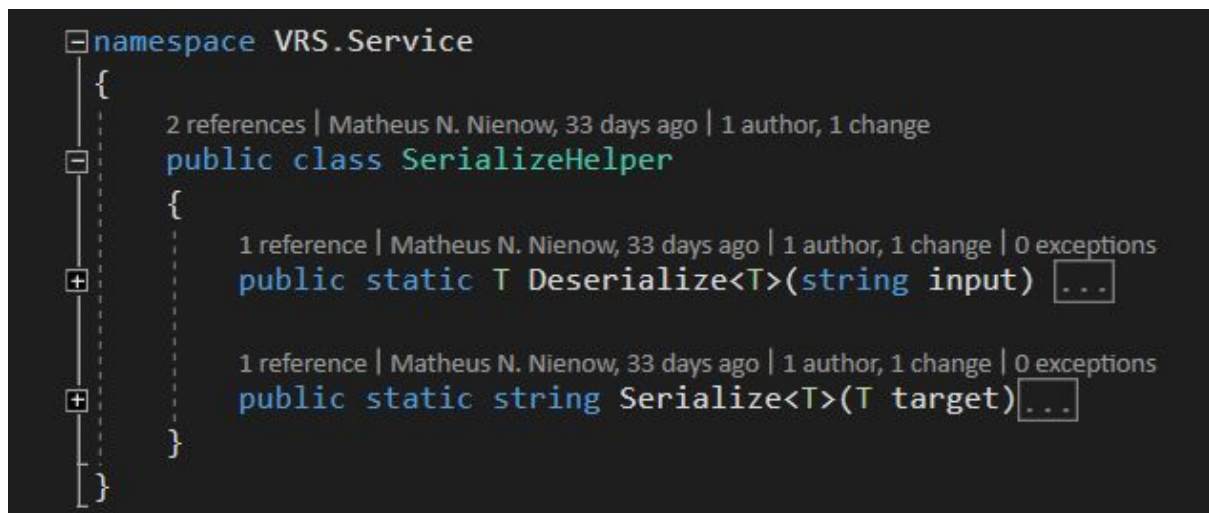
        1 reference | Matheus N. Nienow, 33 days ago | 1 author, 1 change | 0 exceptions
        private static byte[] CreateSalt(int size)...
    }
}
```

VRS.Service

This is a windows service project that is responsible for the background service that reads XML files and inserts data in the database.



This service keeps looking the file "C:\temp\VehicleModelSource.xml" for new vehicle models to insert in the database. There is a class named SerializeHelper, which generic contain methods to serialize and deserialize object. The XML file is simply a list of VehicleModel serialized using this helper.



In the future this project should be extended to accept files for all tables in the database, right now it's only working the VehicleModel.

VRS.WebApi

This is a simple WCF Service Application project that is responsible to expose the access to the business logic to the admin app. The project has the IService interface and Service class implementation for the business actions as well as a Result class

```
namespace VRS.WebApi
{
    [ServiceContract]
    1 reference | Matheus Navarro Nienow, 1 hour ago | 2 authors, 4 changes
    public interface IService
    {
        [OperationContract]
        1 reference | Matheus N. Nienow, 30 days ago | 1 author, 1 change | 0 exceptions
        UserDTO VerifyUser(string username, string password);

        [OperationContract]
        1 reference | Matheus Navarro Nienow, 1 day ago | 2 authors, 2 changes | 0 exceptions
        IEnumerable<RentDTO> GetRents();

        [OperationContract]
        1 reference | Matheus Navarro Nienow, 1 day ago | 2 authors, 2 changes | 0 exceptions
        IEnumerable<UserDTO> GetUsers();

        [OperationContract]
        1 reference | Matheus Navarro Nienow, 1 day ago | 1 author, 1 change | 0 exceptions
        IEnumerable<ClientDTO> GetClients();

        [OperationContract]
        1 reference | Matheus Navarro Nienow, 1 hour ago | 1 author, 1 change | 0 exceptions
        IEnumerable<ClientDTO> GetClientsForName(string clientName);

        [OperationContract]
        1 reference | Matheus Navarro Nienow, 1 day ago | 1 author, 1 change | 0 exceptions
        IEnumerable<VehicleDTO> GetVehicles();

        [OperationContract]
        1 reference | Matheus Navarro Nienow, 22 hours ago | 1 author, 1 change | 0 exceptions
        Result FinishRent(int id);

        [OperationContract]
        1 reference | Matheus Navarro Nienow, 1 hour ago | 1 author, 1 change | 0 exceptions
        Result CreateRent(RentDTO rent);
    }
}
```



```

namespace VRS.WebApi
{
    [DataContract]
    10 references | Matheus Navarro Nienow, 22 hours ago | 1 author, 1 change
    public class Result
    {
        [DataMember]
        1 reference | Matheus Navarro Nienow, 22 hours ago | 1 author, 1 change | 0 exceptions
        public bool Successful { get; set; }
        [DataMember]
        1 reference | Matheus Navarro Nienow, 22 hours ago | 1 author, 1 change | 0 exceptions
        public string Message { get; set; }

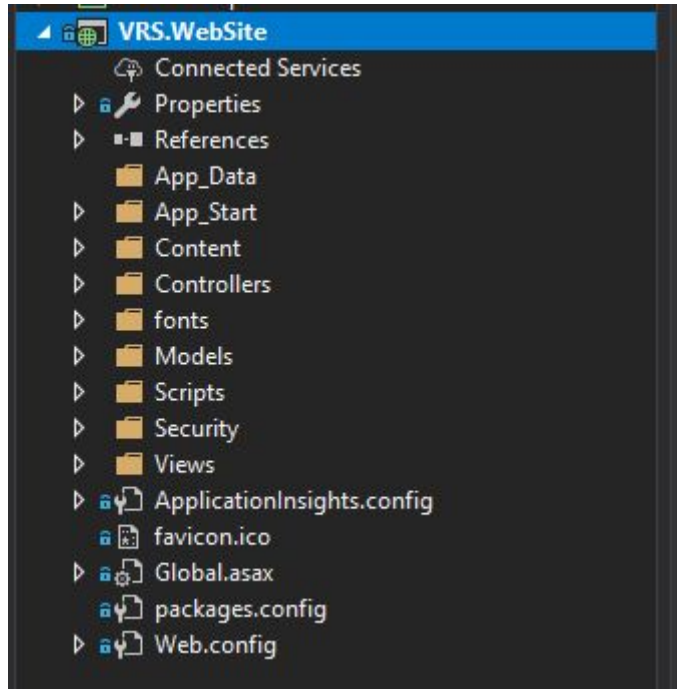
        5 references | Matheus Navarro Nienow, 22 hours ago | 1 author, 1 change | 0 exceptions
        public Result(bool successful, string message)
        {
            Successful = successful;
            Message = message;
        }
    }
}

```

Before sending the data to the admin app, all models should use the method “ToDto()” that transforms itself in the data transfer object model. This avoids the circular reference error and some others.

VRS.WebSite

This is a ASP.NET project that is responsible for the website used by the end user of the business. Here the client can register, see their rents, vehicles and create new rents.



The website has custom authorization that looks for the Role property of the User class. The controllers of this project should only communicate with the controllers of the VRS.Logic project, they shouldn't communicate directly to the VRS.Repository project.

After executing the project, the login screen of the website will show up:

VRSRegisterLog in

Login

Username

Password

Login

© 2017 - VRS

You should create a user first by clicking on the “Register” option in the top bar:

VRSRegisterLog in

Register

Username

Password

Name

Surname

Sex

Sex ▼

Phone

City

BirthDate

Register

© 2017 - VRS

Observation: right now there is no option to create an administrator user through the system. You should create a user through the website and change the Role in the database using the management studio. The creation of the user has to be done through the system because it hashes the password of the user, otherwise the login won't work.

After the login you will be taken to the home screen, which is a list of your rents:

VRSMy RentsVehiclesHello matheus!Log off

My Rents

[New rental](#)

Vehicle	Start	End	Price	Finished
Tesla Model S 2017	2017-12-04	2017-12-06	200 EUR	<input checked="" type="checkbox"/>
Tesla Model 3 2017	2017-12-04	2017-12-10	600 EUR	<input checked="" type="checkbox"/>
Volvo S60 2017	2017-12-04	2017-12-10	600 EUR	<input checked="" type="checkbox"/>
Volvo XC90 2017	2017-12-04	2017-12-16	1200 EUR	<input type="checkbox"/>
Audi Q2 2017	2017-12-04	2017-12-16	1200 EUR	<input checked="" type="checkbox"/>
Tesla Model S 2017	2017-12-04	2017-12-26	2200 EUR	<input type="checkbox"/>
Tesla Model 3 2017	2017-12-06	2018-01-06	3100 EUR	<input type="checkbox"/>

© 2017 - VRS

In this screen is possible to see all of your rents, whether it is finished or not. Using the “New rental” button it is possible to create new rental in the next screen:

VRSMy RentsVehicles

Hello matheus!Log off

New Rental

VehicleId

Tesla Model X 2017

▼

Start

End

Rent

[Back to List](#)

© 2017 - VRS

It is also possible to see a list of the vehicles clicking in the “Vehicles” option on the top bar:

Vehicles

Description	Model	Brand	Year	Mileage	In Use	
Tesla Model S 2017	Model S	Tesla	2017	0 Km	<input checked="" type="checkbox"/>	Details Rent
Tesla Model X 2017	Model X	Tesla	2017	0 Km	<input type="checkbox"/>	Details Rent
Tesla Model 3 2017	Model 3	Tesla	2017	0 Km	<input checked="" type="checkbox"/>	Details Rent
Volvo V40 2017	V40	Volvo	2017	0 Km	<input type="checkbox"/>	Details Rent
Volvo V60 2017	V60	Volvo	2017	0 Km	<input type="checkbox"/>	Details Rent
Volvo V90 2017	V90	Volvo	2017	0 Km	<input type="checkbox"/>	Details Rent
Volvo S60 2017	S60	Volvo	2017	0 Km	<input type="checkbox"/>	Details Rent
Volvo S90 2017	S90	Volvo	2017	0 Km	<input type="checkbox"/>	Details Rent
Volvo XC60 2017	XC60	Volvo	2017	0 Km	<input type="checkbox"/>	Details Rent
Volvo XC90 2017	XC90	Volvo	2017	0 Km	<input checked="" type="checkbox"/>	Details Rent
Audi A7 2017	A7	Audi	2017	0 Km	<input type="checkbox"/>	Details Rent
Audi A6 2017	A6	Audi	2017	0 Km	<input type="checkbox"/>	Details Rent
Audi A8 2017	A8	Audi	2017	0 Km	<input type="checkbox"/>	Details Rent
Audi Q2 2017	Q2	Audi	2017	0 Km	<input type="checkbox"/>	Details Rent
Audi Q3 2017	Q3	Audi	2017	0 Km	<input type="checkbox"/>	Details Rent
Audi Q5 2017	Q5	Audi	2017	0 Km	<input type="checkbox"/>	Details Rent

In each vehicle there are two options: Details and Rent. Clicking in Details you will be able to see more detailed information about the vehicle, and clicking in Rent you will be taken to the create rent page with the vehicle already selected:

VRSMy RentsVehicles

Hello matheus!Log off

Vehicle Model Details

Brand

Gas Type

Vehicle Type

Model

Year

Tesla
Electric
Sedan
Model S
2017

[Back to List](#)

© 2017 - VRS

VRSMy RentsVehicles

Hello matheus!Log off

Rent

Vehicle

Tesla Model X 2017

Start

End

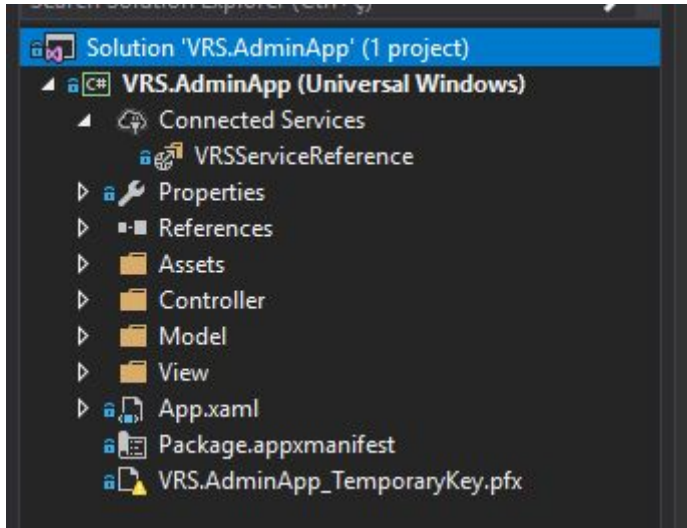
Rent

[Back to List](#)

© 2017 - VRS

VRS.AdminApp

This is a universal windows application project that is responsible for the administration management of the system.



This project has a reference to the VRS.WebApi's service and gets all of the data through this service. It uses the same login as the website but just users with the admin role (id = 1) will have access granted.

The service's methods returns DTO models that are transformed in regular models in the controllers, this should be done for every action of the controller that uses the service.

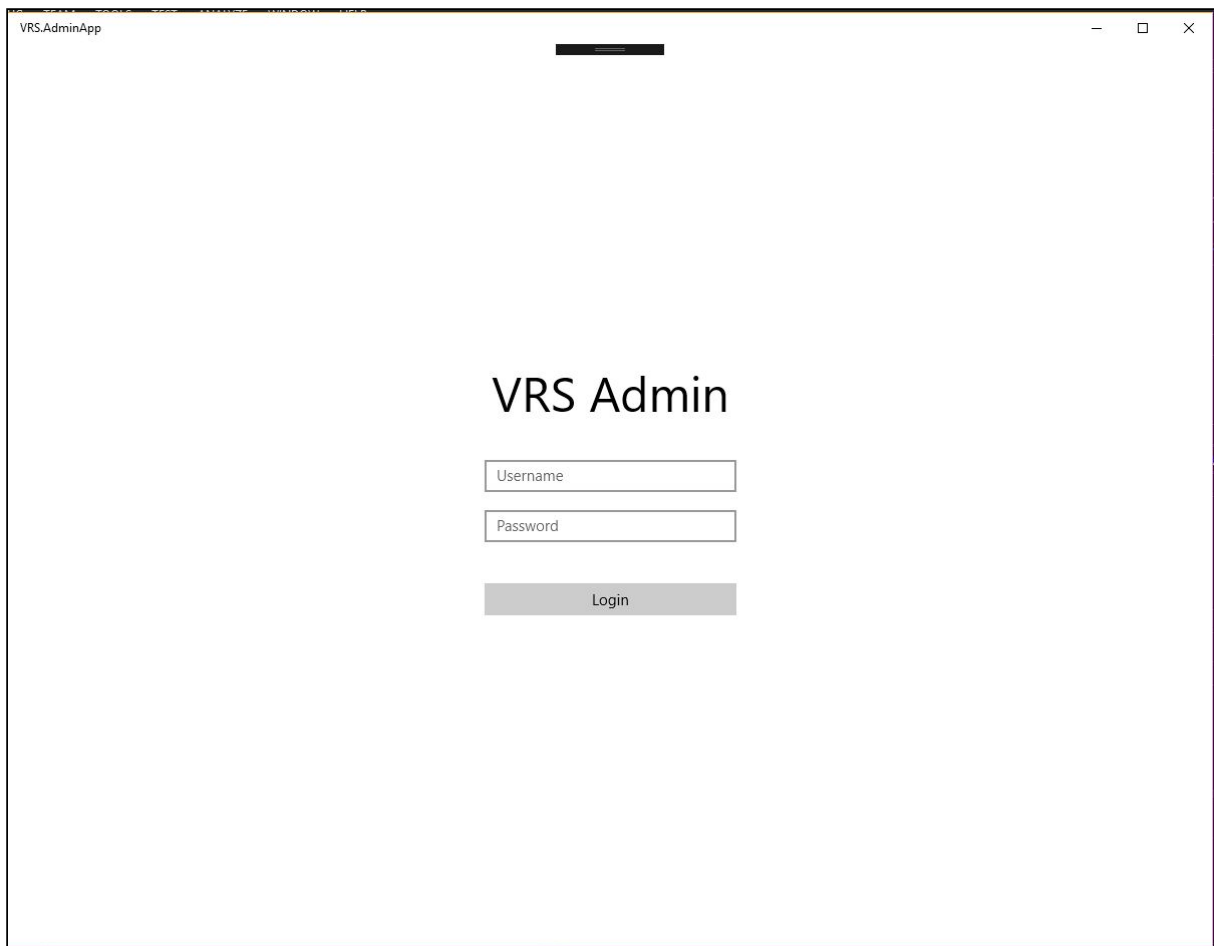
```
1 reference | Matheus Navarro Nienow, 1 day ago | 1 author, 1 change
async internal Task<IEnumerable<Vehicle>> GetVehicles()
{
    var result = await service.GetVehiclesAsync();
    return result.Select(x => new Vehicle(x));
}
```

All the models should have a constructor that receives the DTO model to facilitate the transformation, like in the image above.

Also, all controllers should inherit from the abstract class BaseController, since it has a protected service instance reference.

To executed this app, first the VRS.WebApi should be running. After that, you have to make sure that the service reference in the admin app is pointing to the same address as the VRS.WebApi.

Once executed, the app will open the login page:

A screenshot of a web application window titled "VRS.AdminApp". The window has a standard Windows-style title bar with minimize, maximize, and close buttons. The main content area is white and features a centered login form. The form consists of a large heading "VRS Admin", followed by two input fields labeled "Username" and "Password", and a "Login" button below them. The "Login" button is a solid gray rectangle with white text. The overall layout is clean and minimalist.

After the access is granted the app will take you to the home screen, which holds a list of the rents, users, vehicles and clients of the system:

VRS.AdminApp

ADMIN APP

Rents

Clients

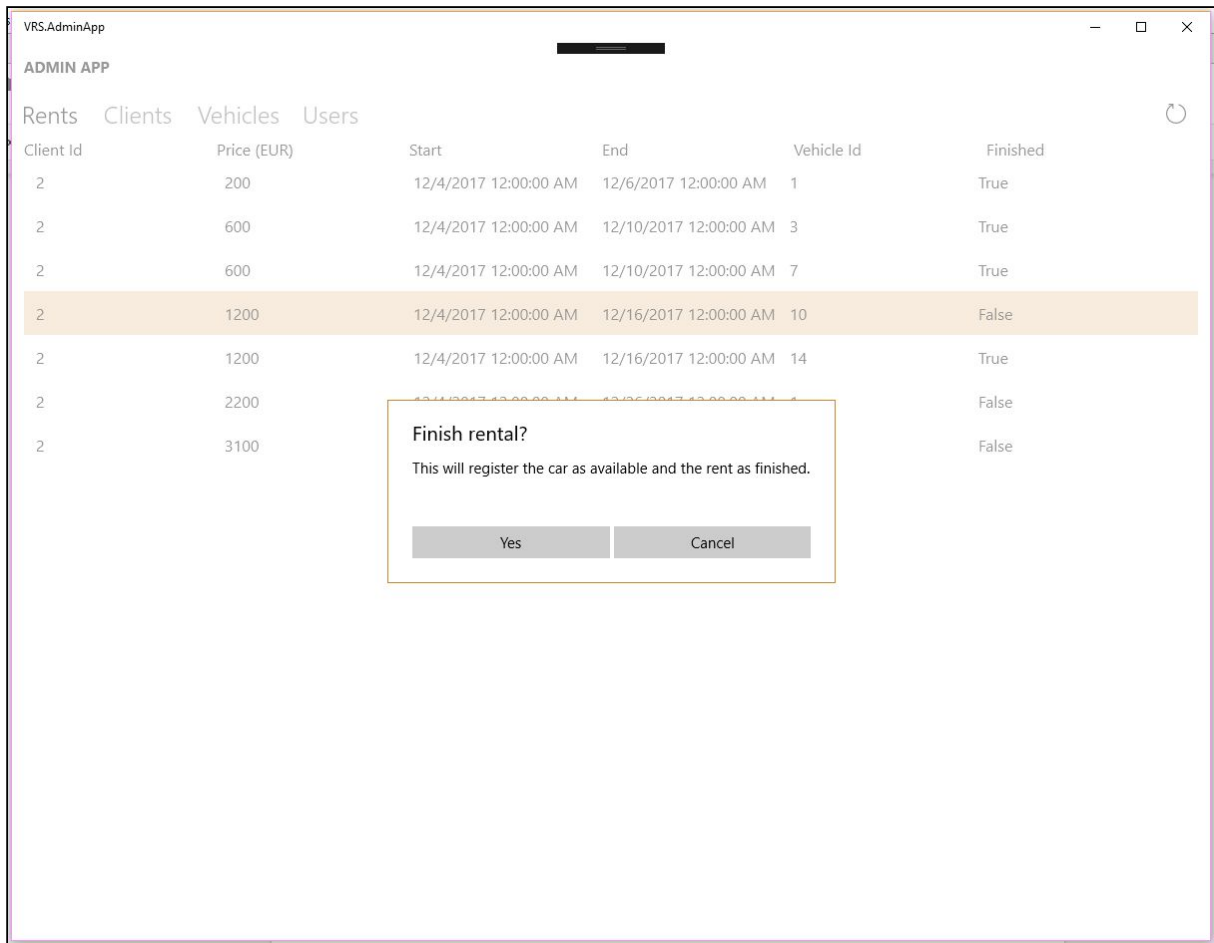
Vehicles

Users

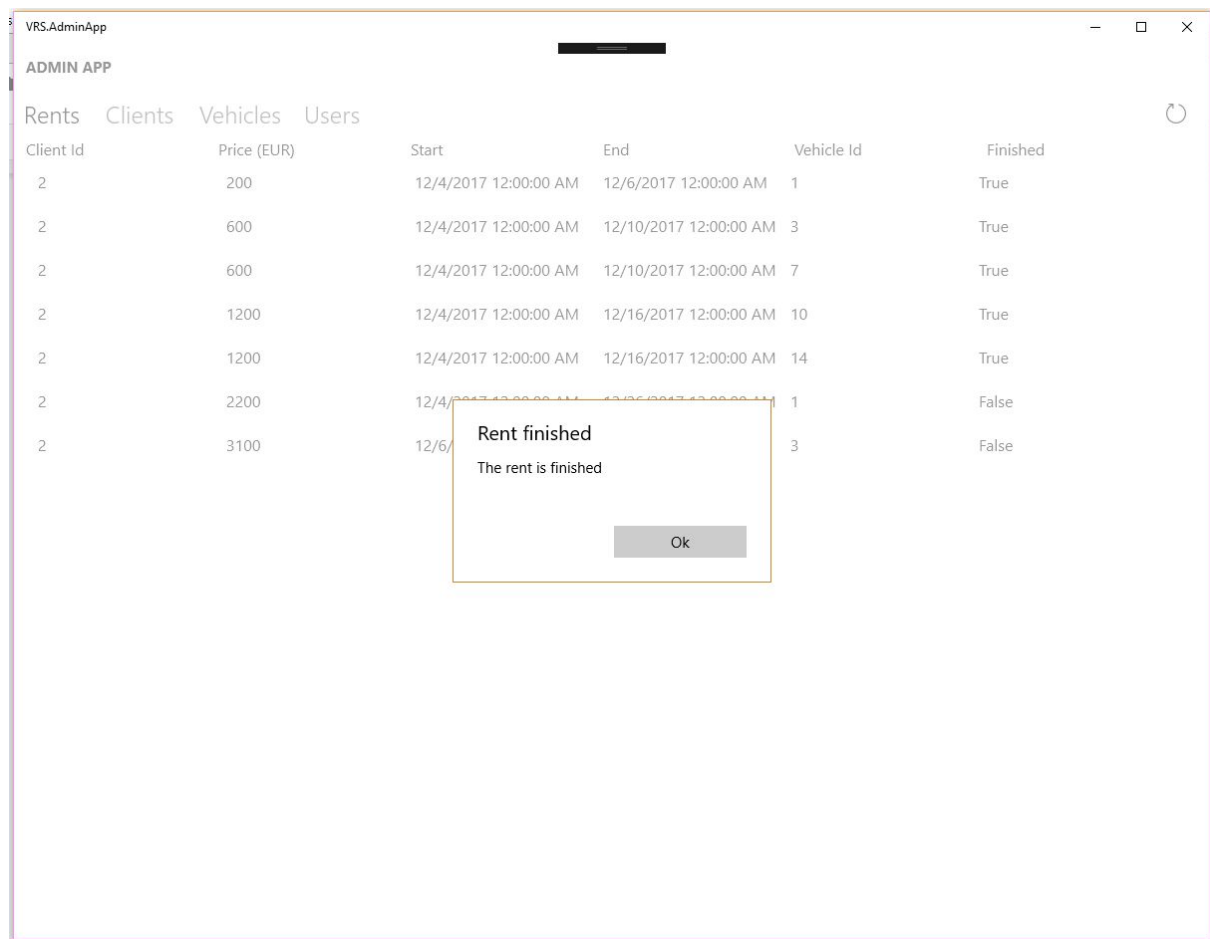
↻

Client Id	Price (EUR)	Start	End	Vehicle Id	Finished
2	200	12/4/2017 12:00:00 AM	12/6/2017 12:00:00 AM	1	True
2	600	12/4/2017 12:00:00 AM	12/10/2017 12:00:00 AM	3	True
2	600	12/4/2017 12:00:00 AM	12/10/2017 12:00:00 AM	7	True
2	1200	12/4/2017 12:00:00 AM	12/16/2017 12:00:00 AM	10	False
2	1200	12/4/2017 12:00:00 AM	12/16/2017 12:00:00 AM	14	True
2	2200	12/4/2017 12:00:00 AM	12/26/2017 12:00:00 AM	1	False
2	3100	12/6/2017 12:00:00 AM	1/6/2018 12:00:00 AM	3	False

Each one of these tabs have the refresh button to update the information on the screen. In the rents list it is possible to click in the rents that are not finished to finish them:



After clicking in the “Yes” button, the rent will be registered as finished, the car will be registered as available in the database and the list is updated:



In the vehicle list, it is possible to click in a vehicle that is not in use to create a new rental:

VRS.AdminApp								
ADMIN APP								
Rents	Clients	Vehicles	Users					↺
Id	Description	Plate	Mileage (Km)	In use		Vehicle Model		
1	Tesla Model S 2017	AAA1111	0	True		1		
2	Tesla Model X 2017	AAA1112	0	False		2		
3	Tesla Model 3 2017	AAA1113	0	True		3		
4	Volvo V40 2017	AAA1114	0	False		4		
5	Volvo V60 2017	AAA1115	0	False		5		
6	Volvo V90 2017	AAA1116	0	False		6		
7	Volvo S60 2017	AAA1117	0	False		7		
8	Volvo S90 2017	AAA1118	0	False		8		
9	Volvo XC60 2017	AAA1119	0	False		9		
10	Volvo XC90 2017	AAA1120	0	False		10		
11	Audi A7 2017	AAA1121	0	False		11		
12	Audi A6 2017	AAA1122	0	False		12		
13	Audi A8 2017	AAA1123	0	False		13		
14	Audi Q2 2017	AAA1124	0	False		14		
15	Audi Q3 2017	AAA1125	0	False		15		
16	Audi Q5 2017	AAA1126	0	False		16		

The screenshot shows a web application window titled 'VRS.AdminApp'. The main heading is 'New rent'. Below it is the text 'AAA1112'. There is a search section with a text input labeled 'Client name' and a 'Search' button. Below the search is a 'Client:' dropdown menu. The 'Start' date is set to 'December 6 2017' and the 'End' date is also set to 'December 6 2017'. There is a 'Price:' label followed by a large grey button labeled 'Create rent'.

In this page you can search for a client using the “Search” button, it will show the results in the combo right below it. After creating the rent it will return to the home screen.

Final considerations

All of the code is located can be found on these repositories:

<https://github.com/matheusnienow/VehicleRentSystem>

<https://github.com/matheusnienow/VRS.AdminApp>

The admin app should be updated to include the creation of users, both clients and administrators. It also should be able to filter the list. The layout of the app should be improved.

The website needs an layout overhaul and some fix in some descriptions.

There is a problem with the structure of the project. The VRS.WebApi project should be contained in a different solution. Right now it's not possible to executed both the WebSite and the WebApi at the same time. I've tried to open two instance of Visual Studio in the same project, but apparently both projects shares a resource and this causes the WebApi to close once the WebSite is started.

Matheus Navarro Nienow & Shengkai Zang