UNIVERSITATEA DIN CRAIOVA
FACULTATEA DE AUTOMATICĂ, CALCULATOARE ȘI ELECTRONICĂ

DEPARTAMENTUL DE CALCULATOARE ȘI TEHNOLOGIA INFORMAȚIEI

PROIECT DE DIPLOMĂ

Anghel Paul David

COORDONATOR ȘTIINȚIFIC

Dr. Ing. Marius Brezovan

Ing. Cătalin Sbora

September 2021

CRAIOVA

UNIVERSITATEA DIN CRAIOVA
FACULTATEA DE AUTOMATICĂ, CALCULATOARE ȘI ELECTRONICĂ

DEPARTAMENTUL DE CALCULATOARE ȘI TEHNOLOGIA INFORMAȚIEI

Cars Trading Website

Anghel Paul David

COORDONATOR ȘTIINȚIFIC

Dr. Ing. Marius Brezovan

Ing. Cătalin Sbora

September 2021

CRAIOVA

"Wisdom is not a product of schooling but of the lifelong attempt to acquire it."

Albert Einstein.

DECLARAȚIE DE ORIGINALITATE

Subsemnatul Anghel Paul-David, student la specializarea Calculatoare în limba Engleză din cadrul Facultății de Automatică, Calculatoare și Electronică a Universității din Craiova, certific prin prezenta că am luat la cunoştinţă de cele prezentate mai jos şi că îmi asum, în acest context, originalitatea proiectului meu de licenţă:

- cu titlul: Cars Trading Website,
- coordonată de Dr. Ing. Marius Brezovan şi Ing. Cătalin Sbora,
- prezentată în sesiunea Septembrie 2021.

La elaborarea proiectului de licenţă, se consideră plagiat una dintre următoarele acţiuni:

- reproducerea exactă a cuvintelor unui alt autor, dintr-o altă lucrare, în limba română sau prin traducere dintr-o altă limbă, dacă se omit ghilimele şi referinţa precisă,
- redarea cu alte cuvinte, reformularea prin cuvinte proprii sau rezumarea ideilor din alte lucrări, dacă nu se indică sursa bibliografică,
- prezentarea unor date experimentale obţinute sau a unor aplicaţii realizate de alţi autori fără menţionarea corectă a acestor surse,
- însuşirea totală sau parţială a unei lucrări în care regulile de mai sus sunt respectate, dar care are alt autor.

Pentru evitarea acestor situaţii neplăcute se recomandă:

- plasarea între ghilimele a citatelor directe şi indicarea referinţei într-o listă corespunzătoare la sfărşitul lucrării,
- indicarea în text a reformulării unei idei, opinii sau teorii şi corespunzător în lista de referinţe a sursei originale de la care s-a făcut preluarea,
- precizarea sursei de la care s-au preluat date experimentale, descrieri tehnice, figuri, imagini, statistici, tabele et caetera,
- precizarea referinţelor poate fi omisă dacă se folosesc informaţii sau teorii arhicunoscute, a căror paternitate este unanim cunoscută și acceptată.

Data, Semnătura candidatului,

02.09.2021

UNIVERSITATEA DIN CRAIOVA
Facultatea de Automatică, Calculatoare şi Electronică

Departamentul de Calculatoare şi Tehnologia Informației

Aprobat la data de
…………………
Şef de departament,
Prof. dr. ing.
Marius BREZOVAN

PROIECTUL DE DIPLOMĂ

Numele şi prenumele studentului/-ei:	Anghel Paul-David
Enunțul temei:	Cars Trading Website
Datele de pornire:	Create a car trading website similar to Carzz.ro
Conținutul proiectului:	The project is implementing a car trading website with detailed functional requirements and user friendly experience, also the research made and the documentation needed in order to describe every step of the planning, designing, implementing, conclusion and what it can be improved.
Material grafic obligatoriu:	Project documentation, Power Point presentation, source code
Consultații:	Periodice
Conducătorul ştiințific (titlul, nume şi prenume, semnătura):	Dr. Ing. Marius Brezovan şi Ing. Cătalin Sbora
Data eliberării temei:	01.10.2020
Termenul estimat de predare a proiectului:	03.09.2021
Data predării proiectului de către student şi semnătura acestuia:	

UNIVERSITATEA DIN CRAIOVA
Facultatea de Automatică, Calculatoare şi Electronică

Departamentul de Calculatoare și Tehnologia Informației

REFERATUL CONDUCĂTORULUI ȘTIINȚIFIC

Numele şi prenumele candidatului/-ei:	
Specializarea:	Calculatoare în limba Engleză
Titlul proiectului:	*Image-based Steganography Applications using Watermarking Techniques*
Locația în care s-a realizat practica de documentare (se bifează una sau mai multe din opțiunile din dreapta):	În facultate ☐ În producție ☐ În cercetare ☐ Altă locație: [*se detaliază*]

În urma analizei lucrării candidatului au fost constatate următoarele:

Nivelul documentării		Insuficient ☐	Satisfăcător ☐	Bine ☐	Foarte bine ☐
Tipul proiectului		Cercetare ☐	Proiectare ☐	Realizare practică ☐	Altul [*se detaliază*]
Aparatul matematic utilizat		Simplu ☐	Mediu ☐	Complex ☐	Absent ☐
Utilitate		Contract de cercetare ☐	Cercetare internă ☐	Utilare ☐	Altul [*se detaliază*]
Redactarea lucrării		Insuficient ☐	Satisfăcător ☐	Bine ☐	Foarte bine ☐
Partea grafică, desene		Insuficientă ☐	Satisfăcătoare ☐	Bună ☐	Foarte bună ☐
Realizarea practică	Contribuția autorului	Insuficientă ☐	Satisfăcătoare ☐	Mare ☐	Foarte mare ☐
	Complexitatea temei	Simplă ☐	Medie ☐	Mare ☐	Complexă ☐
	Analiza cerințelor	Insuficient ☐	Satisfăcător ☐	Bine ☐	Foarte bine ☐

	Arhitectura	Simplă ☐	Medie ☐	Mare ☐	Complexă ☐
	Întocmirea specificațiilor funcționale	Insuficientă ☐	Satisfăcătoare ☐	Bună ☐	Foarte bună ☐
	Implementarea	Insuficientă ☐	Satisfăcătoare ☐	Bună ☐	Foarte bună ☐
	Testarea	Insuficientă ☐	Satisfăcătoare ☐	Bună ☐	Foarte bună ☐
	Funcționarea	Da ☐	Parțială ☐	Nu ☐	
Rezultate experimentale		Experiment propriu ☐		Preluare din bibliografie ☐	
Bibliografie		Cărți	Reviste	Articole	Referințe web
Comentarii și observații					

În concluzie, se propune:

ADMITEREA PROIECTULUI	RESPINGEREA PROIECTULUI
☐	☐

Data,

Semnătura conducătorului științific,

PROJECT SUMMARY

The project name is Cars Trading Website, that is implementing a platform where car buyers can search the car they want suited on their needs by navigating through a lot of cars announcements with detailed descriptions, images and seller information. The seller is represented by the person that creates the announcements and can track his announcemnets performences through the view count, how many buyers contacted him through the integrated messaging system.

The project is made using ASP .NET core technology developed by Microsoft and running on Windows. The backend language is C# and as frontend frameworks I used HTML, CSS, Bootstrap, razor pages. As design pattern I used MVC that means model view controller, where the model is represented by the main element of the pattern that manages the data and logic, the view is represented by the user interface where the data or information is displayed to the user. And the controller is taking the inputs that are transformed in commands to the model and view.

Another pattern used in the application is Repository pattern that consists in abstracting the data access. This consists in reducing the code duplication and having a base repository with standard methods that are suited for other entities such as CRUD actions (Create, Read, Update, Delete), so when we are creating another repository we already have these methods implemented and of course there can be added some additional methods depending on what features are needed to implement in the repository. And the repositories are called in the services, and the services are called in the controller.

As IDE for the project I used Visual Studio 2019 because is stable and its autocomplete component named IntelliSense is really great, and is offering support for a various number of platforms. It is easy to navigate between classes, project files, and is easy to install additional packages via Nuget package manager, also it can be integrated with TFS or GIT.

I used SQL server to host the database, that stores and retrieves information requested by the app. The database I go for is LocalDB that can be managed inside the visual studio 2019 accessing it really fast. As an alternative it can be used SQL Server Management Studio that is an integrated environment for monitoring and manage SQL Server instances and also can make interogations on the database to test the potential outcome of the queries before implementing them in the code.

The application has three main roles: Admin, Moderator and Buyer/Seller. The admin is represented by the person that administrates the website having more rights then a normal user, he can review announcements, ban users, post on newsletter, add a new category in the website, chat with other users, delete announcements.

The moderator is basically a user with some of the admin rights, for example he can post news, can delete news, and also the main purpose of the moderator is to review announcements to make the admin life more easier.

The application user can be a seller and also the buyer, because every authenticated user can post announcements, view all the approved ones, contact other users on the platform using the integrated chat system, can update profile information such as password, phone number, email. Every announcement is added in a specific category to make the filtering more efficient.

The guest is represented by the person who doesn't have an account yet, so the actions are limited to only view the announcement list and apply a lot of filters to find the best match. If the guest tries to see detailed information about a specific announcement, then he/she is redirected to the login page to become an authenticated user in order to access the advanced features of the websiste.

Other main functionality of the website is the recommender system that is starting to display announcements in a section called recommended announcements being present on the homepage of the application. The recommender system is taking in considerations the user desires and in what cars he/she is interested based on the filtering that a user is applying.

I chose the name of the application to be „Racooter" by constructing it from two words „Racoon" and „scooter", so my website motto is „How do you call a racoon on a scooter? - Racooter". The scooter is representing the niche where the website is placed: with vehicles, motorcycles, scooters and car parts.

Key terms: *cars trading, messages integration, recommender system, cars announcements, newsletter, number of views,* filters*, review announcements, moto, auto, car parts*

REZUMAT PROIECT

Proiectul este intitulat Cars Trading Website, care implementează o platformă în care persoanele interesate in achizitia unei mașini, pot gasi o mulțime de anunțuri despre vehicule, mașini, motociclete, cat si piese auto. Fiecare anunt contine o descriere detaliata, cu imagini, specificatii, categoria din care face parte și informații despre vânzător. Vânzătorul este reprezentat de persoana care creează anunțurile și își poate urmări performanțele anunțurilor prin numărul de vizualizări, cât si prin numarul de cumpărători care l-au contactat prin intermediul sistemului de mesagerie integrat.

Utilizatorii pot face cautari cat si filtra anunturile in functie de mai multi parametrii, care vor expune aplicatiei dorintele si interesele utilizatorului ca apoi aplicatia sa inceapa a recomanda anunturi personalizate pe cererile si dorintele acestuia, prin sistemul de recomandare integrat.

Proiectul este realizat in tehnologia ASP.NET Core, dezvoltata de Microsoft si ruleaza pe windows. Limbajul de programare in care este scris backend-ul aplicatiei este C#, iar pentru frontend a fost folosit HTML, CSS, JavaScript, RazorPages si Bootstrap.

Ca si design pattern am folosit MVC (model view controller), in care modelul reprezinta forma, sablonul datelor care salveaza datele extrase din baza de date, View-ul reprezinta interfata unde datele si informatiile corespunzatoare sunt afisate utilizatorului, iar Controller-ul preia datele de intrare ale utilizatorului si le transforma in comenzi catre Model si View.

Un alt pattern folosit este Repository Pattern, care consta in abstractizarea accesului la date. Acest lucru va rezulta in reducerea codului duplicat si obtinerea unui repository generic cu metode standard care sunt valabile pentru toate entitatile, cum ar fi actiunile CRUD (create, update, delete). Deci cand creeam un alt repository o sa avem aceste metode deja implementate si bineinteles pot fi adaugate si alte metode, in functie de functionalitatea care trebuie implementata. Repository-ul este apelat in serviciu, iar mai departe acesta este apelat in controller.

Ca si IDE pentru dezvoltarea aplicatiei, am folosit Visual Studio 2019 pentru ca este foarte stabil si contine o componenta solida de autocompletare numita IntelliSense, dar si ca suporta o multime de platforme si tehnologii. Navigarea prin proiect se face cu usurinta si este destul de bogat in extensii si pachete care pot fi instalate din Nuget package, dar si poate fi integrat cu TFS sau GIT.

Am folosit SQL Server pentru baza de date, care salveaza si extrage informatile necesare pentru aplicatie. Am ales LocalDB deoarece poate fii gestionata chiar din Visual Studio, accessand baza de date foarte rapid si eficient. Ca alternativa se poate folosi SQL Server Management Studio (SSMS) care este un mediu integrat pentru a monitoriza instantele SQL Server, cat si pentru a realiza

interogari pe baza de date pentru a testa dinainte rezultatul interogarilor inainte de implemntarea lor in cod.

Aplicatia contine trei roluri principale: Admin, Moderator si Cumparator/Vanzator (utilizator principal). Admin-ul este reprezentat de persoana care administreaza site-ul avand mai multe atributii si functionalitati decat un user normal. In atributiile acestuia intra: poate sa revizuiasca anunturile inainte de a fi afisate pe site, poate bloca utilizatorii din a mai posta anunturi, adauga noi categorii, poate vorbi cu alti utilizatori prin intermediul sistemului de mesagerie integrat, poate sterge anunturi, posteaza pe pagina de noutati, poate vizualiza cat si modifica conturile tuturor utilizatorilor platformei.

Moderatorul este practic un utilizator cu unele dintre drepturile admin-ului cum ar fi: revizuirea anunturilor, postarea de noutati cat si stergerea lor, astfel face o parte din treaba admin-ului crescand eficienta si timpul de asteptare al utilizatorilor pentru a fi aprobate anunturile postate.

Utilizatorul principal al aplicatiei poate fi atat cumparator cat si vanzator, pentru ca fiecare utilizator logat poate posta anunturi, poate vizualiza anunturile aprobate, poate contacta alti utilizatori ai platformei prin mesageria integrata, poate sa isi modifice informatiile despre nume, numar de telefon email, cat si parola. Poate filtra anunturile dupa diferite criterii cum ar fii: Marca, Model, Capacitatea motorului, Pret, anul fabricatiei etc.

De asemenea utilizatorul va primi intr-o sectiune speciala din homepage, anunuturi recomandate in functie de cautarile si filtrarea rezultatelor, reprezentand interesele si preferintele acestuia.

Un rol secundar este reprezentat de Guest, fiind persoana care inca nu detine un cont, astfel este limitat la cateva actiuni de baza cum ar fi: poate vedea toate anunturile, poate aplica filtrele, in rest este restrictionat si in caz ca incearca sa acceseze detalii despre un anumit anunt, este redirectionat catre pagina de login pentru a se loga sau crea un cont, pentru a debloca functionalitatile avansate ale platformei.

Am ales ca site-ul sa se numeasca „Racooter”, prin constructia a doua cuvinte: „Racoon” care inseamna raton si „Scooter” care inseamna scuter. Astfel site-ul are ca si motto: „Cum numesti un raton pe un scuter? - Racooter”. Iar scuterul reprezinta domeniul si nisa de activitate a site-ului web fiind dominata de vehicule, motociclete, scutere si piese de masini.

TABLE OF CONTENTS

FIGURE LIST

1 INTRODUCTION

1.1 Purpose

The main purpose of the project is to create an online website that is allowing users to trade cars more efficient, by connecting buyers and sellers on a platform. The interface is user friendly and intuitive that will motivate a lot of users to use the application because is really easy to search an announcements that a user is interested in due to applying a variety of filters that contains keywords and predefined values, for example: the category they are searching for in order to display only results based on their needs and interests.

A second purpose is to make the communication more efficient, so the website has a built in chat, that is allowing every user to just connect and message every seller on the platform and ask more details about the car he/she is interested. The user can receive support almost instantly just by messaging the admin or moderator of the website to find out more information about the application and how the things are working.

A third purpose of the application is to have a great and well maintained community, that can be achieved by giving a set of base rules that the users must follow, otherwise they can be banned from posting announcements or updating the existing ones that a specific user owns. And more, a simple buyer can report anytime a seller based on a misinformation, false, or misleading info that was communicated through an announcement or in the private chat. In this way the admin can see the reported users and analyze their activity on the platform and can lead to permanent or temporar banning. In plus, the admin can post in the newsletter different guidelines or common issues that users are encountered and what they can do in that specific situation.

Another purpose is to get familiar with users needs and interests in order to provide to them custom recommended announcements based on their filtering that will lead to more connctions between buyers and sellers. This will fill the main objective of the application and generate more sales as well.

On the other hand, when someone wants to buy a car, the user experience is really important because buying a vehicle can be stressfull and hard to get matches on what exactly the user is searching for. But using an online platform the process is simplified and he/she does not need to make unnecessary trips to different dealerships because with some clicks can see a list of announcements from different sellers. Also there are multiple contact information, through phone call, direct message, email, or to go in the location specified in the announcement if is near enough, because also detailed information about the location where the user can see and test the car are displayed on each announcement.

1.2 Motivation

The motivation behind the development of the Racooter – Cars Trading Website is to assist and improve the experience of a user when buying a new car, providing a fresh online platform that can handle a lot of cars announcements with very detailed information. Because a user is interested to see all the sides and small details when buying a car and in a lot of cases just providing more characteristics of the product it will help to narrow down the searches and find what is the most suited car, avoiding unnecessary trips to the seller or phone calls just to get the extra missing information.

Another motivation in choosing this theme is that I like everything about cars and the little details, mechanical stuffs, thats why I implemented in the application a lot of filters by make, model, category, mileage, price, year, power, announcement title and a lot of information on the announcement details page that will let the user know if a specific car is full option or which individual options it has or not. And other important thing on such platform is to inform the user if the price is negotiable or is fixed, to know from the start how much the seller is willing to lower the price, this giving a smooth comunication because both of them are on the same page and know what to expect.

On the other hand, implementing a complete trading website can be quite challenging and I learned a lot of things on the way, specially on the messaging feature and recommender system that must work great in order to give the best user experience. The first version of the message feature was an offline chat, meaning that the receiver will see the message only if refreshes the page, but I was motivated to make it more like an online chat where the users are seeing the messages almost instantly. So I tried to keep the things real and simulate as much as possible how would be in a real environment and think of possible solutions that will improve the site performance and user experience.

1.3 Chapters resume

Below will be displayed the project chapters resume and a short description at what we should expect in each chapter, in order to have an overview of the project.

Chapter 1: Introduction – Here will be presented the purpose of the application, represented by the reasons for the application was created and what benefits can add to the table. In total are described five important reasons that strongly sustain the development of this project.
Other point discussed in this chapter is the motivation, that is referring to the inspiration and the drive of implementation such a challenging ideea.

Chapter 2: Research on the topic – In this chapter are presented the technology used to build the project, and also other tools like Drawio. Other point touched in this chapter is represented by the comparisions between other online auto platform that already exists and are described the main features of them that can be present in this project as well.

Chapter 3: Project requirements and specifications – In this section are described the functional requirements and also the diagrams such as use case diagram, activity, class, data access models and also a description of the application architecture.

Chapter 4: Project implementation – Here will be presented the project structure to have an overall ideea how the project files are structured, also the graphical user interface with screenshots from the applications. In this chapter are described in detail all the features implemented in the project and also with examples.

Chapter 5: Conclusions and improvements – In this section are described some conclusions after implementing and researching the project, and also future improvements that will make the application even better.

Chapter 6: Bibliography – In this chapter are listed all the materials that contributed in obtaining the needed information for the research and development of the application.

Chapter 7: Web references – Here are placed all the web references related to the project.

2 RESEARCH ON THE TOPIC

2.1 Visual Studio IDE

„Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It is used to develop computer programs, as well as websites, web apps, web services and mobile apps. Visual Studio uses Microsoft software development platforms such as Windows API, Windows Forms, Windows Presentation Foundation, Windows Store and Microsoft Silverlight. It can produce both native code and managed code.”[1]

Some of the main features of Microsoft Visual Studio that are making it a perfect fit for developing the solution are:

- **Code editor –** „it includes a code editor that supports syntax highlighting and code completion using IntelliSense for variables, functions, methods, loops, and LINQ queries. IntelliSense is supported for the included languages, as well as for XML, Cascading Style Sheets, and JavaScript when developing web sites and web applications. Autocomplete suggestions appear in a modeless list box over the code editor window, in proximity of the editing cursor.”[1]

- **Debugger –** „it includes a debugger that works both as a source-level debugger and as a machine-level debugger. It works with both managed code as well as native code and can be used for debugging applications written in any language supported by Visual Studio. In addition, it can also attach to running processes, monitor, and debug those processes.”[1]

- **Designer –** is integrating a lot of visual designers to assist the developer in creating a great application depending on the purpose, needs and the platform on which will be implemented. The main designers that are integrated in Visual Studio are the following: Windows Forms Designer, WPF Designer, Web designer, Class Designer, Data Designer.

- **Other –** some of the other main tools integrated in the IDE are the following:

  - Solution Explorer, where the user can see all the code files, libraries, dependencies and other ressources used in running and building the application
  - Team Explorer, where the user can integrate Azure DevOps features with version control integration that will keep track of pending changes in the local repository, also basic actions can be done as push, fetch, pull, make a pull request, create a branch, commit and also it is great that can be seen the modifications made in each file to know what is changed from the last version.

- Data Eplorer, the main purpose is to manage databases of SQL Server insances. It can generate queries to make interogations on the database and also create, update or delete other tables, columns, or dropping the database.
- Server Explorer, its main purpose is to manage database connections.

I chose this IDE because I worked with it in the past and I really enjoy the user experience and the variety of possibilities and technologies that can be integrated with the environment being the best choice for me.

Having the option to access the database directly from the IDE is a nice feature through the Server Explorer, in this way navigating more easily in the database and checking if the entries were saved correctly without any other issues.

The version that I used is Microsoft Visual Studio Community Edition 2019 because is free and has all the features I need, this is a strong aspect in choosing Visual Studio as IDE.

2.2 .NET Framework

„The .NET Framework is a software framework developed by Microsoft that runs primarily on Microsoft Windows. It includes a large class library called Framework Class Library (FCL) and provides language interoperability across several programming languages. Programs written for .NET Framework execute in a software environment named the Common Language Runtime (CLR). The CLR is an application virtual machine that provides services such as security, memory management, and exception handling. As such, computer code written using .NET Framework is called managed code. FCL and CLR together constitute the .NET Framework.

FCL provides the user interface, data access, database connectivity, cryptography, web application development, numeric algorithms, and network communications. Programmers produce software by combining their source code with .NET Framework and other libraries. The framework is intended to be used by most new applications created for the Windows platform."[2]

Some of the advantages in using this technology that empowered my decision in using this stack are:

- **Object oriented** – „.NET is based on Object-Oriented Programming Module. OOP is a development model that involves breaking down software into easily manageable smaller pieces. OOP compartmentalizes data into data fields and describes objects behavior through the declaration of classes. Object-Oriented Programming Module simplifies by making the code manageable, respond to recurring issues and easier to test. It eliminates necessary programming and hence less coding for developers. In addition to this, .NET makes it possible to reuse components and code, thereby saves time and cost of development." [3]

- **Cross-platform design** – „.NET Core is a cross-platform, which means it allows the code to run on Windows, Linux and OS X. .NET core, unlike the original .NET framework has a fully open-source code which ensures that a wide engineering community can contribute to its development. If you are writing the code in F#, C# or Visual Basic, your code will run each of the compatible operating systems. This allows companies to reach an extensive variety of platforms staying within the .NET ecosystem. At the same time, with the cross-platform design, it also becomes possible for the .NET community to share their large pool of engineering skillsets." [3]

- **Easy Maintenance** – „One of the crucial and most advantageous features of .NET Core is flexible deployment. It can be installed as a part of the application you are developing as well as separately. The modular design allows including all the dependencies that you need. Moreover, the deployment with .NET is as easy as copying a folder. [3]

  Another benefit is that you can have more than one .NET Core versions running side by side on the same machine. Hence, making it easy to cover different projects and seamlessly perform a deployment task." [3]

- **Large community** – A lot of peoples and companies uses .NET technology so it is really great documented by Microsoft and provides support to almost any encountered issue because someone already been there and solved the problem. There are a lot of forums about .NET and one is provided by Microsoft to make sure that their users are content and the problems are solved efficiently.

- **Cost effective** – When choosing a framework, the costs of running the project on a specific technology are an important aspect. .NET is often used with visual studio code that is pretty small, extensible, free and constantly updated. And the infrastructure of .NET is providing flexibility in choosing a supplier for SLA (Service level agreement). And there is the posibility to host the project in the cloud.[3]

My application is implemented using ASP.NET Core[4] that is a modular framework that can run on the entire .Net Framework on Windows but also on the cross-platform .NET Core. With the ASP.NET Core can be built modern web apps and services that can be deployed on the cloud and of course are running on .NET Core being highly efficient and optimized by Microsoft.

2.3 C# programming language

„C# is a general-purpose, multi-paradigm programming language encompassing strong typing, lexically scoped, imperative, declarative, functional, generic, object-oriented (class-based), and component-oriented programming disciplines. It was developed around 2000 by Microsoft as part of its .NET initiative and later approved as an international standard by Ecma (ECMA-334) in 2002 and ISO (ISO/IEC 23270) in 2003. Mono is the name of the free and open-source project to develop a compiler and runtime for the language. C# is one of the programming languages designed for the Common Language Infrastructure (CLI).”[5]

The main features of C# programming languages that makes it a great fit on my application are:

- **Object oriented** – the application that I want to implement must be object oriented in order to organize the program around objects and data, instead of actions and logic. The main benefits of using an object oriented programming language is the integration of these three principles: Encapsulation, Inheritance and Polymorphism[6].

The Encapsulation is basically wrapping up data members and methods togheter, into a single unit often represented by a class. In this way the the internal characteristics of an object is hidden, this will prevent unauthorized users to access the behaviour of the abstraction of a certain object.[JS19]

The Inheritance concept is occuring when a class includes some properties of another class. This will prevent code duplication in order to have a more cleaner project, and is really usefull evan when a change is made on the base class, because automatically the derived class will inherit the new properties and methods added in the parrent class.

The Polymorphism principle is basically „one name, many forms”[7] A method that can have different behaviours. An example of polymorphism use is the overloading technique, being a static polymorphism: a method has the same name but different signatures, meaning that will have distinct parameters. This is allowing to give extra functionalities and other behaviour then the previous implementation of the same method, just by changing the signature.

- **Modern programming language** – is very up to date and really strong for building huge, safe and scalable applications. Its also easy to maintain, also it has a garbage collector that automatically reclaims memory that was filled by unused objects. The exeption handling is on point as well, providing an exetensible solution for detecting errors and recovery.

- **Scalable** – C# is automatic scalable programming language that can handle increasing the load without affecting the performance.

- **Many libraries** – C# is providing a lot of libraries that are making the implementaion significantly quicker. Some of them that are integrated in .NET: AutoMapper is used to map one object to another, Autofac is a container that manages the dependencies between classes, FluentValidation is used for building validation rules with the help of lambda expressions.[8]

2.4 Design patterns

The main design patterns used in the ASP.NET application are MVC (Model View Controller) and Repository Pattern that will be described below.

MVC Pattern[9] is basically dividing the application in three main sections: Models, Views and Controllers. This particular pattern assists the developer to obtain separation of concerns: a guiding principle in assuming that the software application should be divided based on the type of work it performs. User requests are linked to a controller that will work closely with the Model to perform the requested action or to retrieve interogation results. The controller is selecting the View to display back to the user the information, and fills it with the needed Model data.

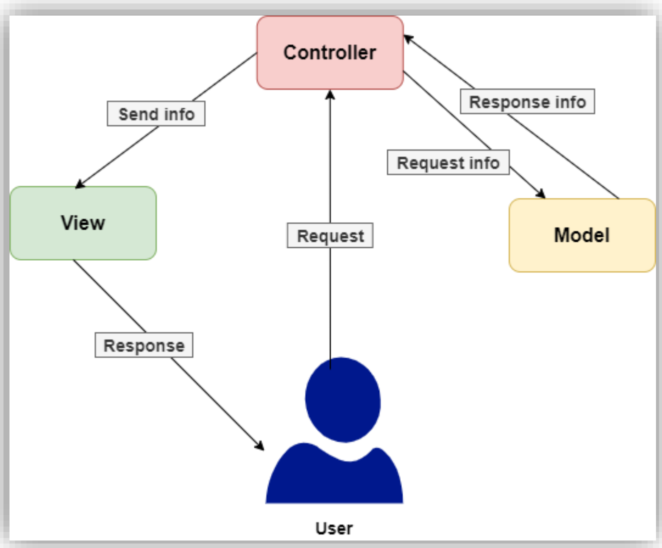


Figure 1. MVC pattern scheme

In the presented scheme can be observed the flow and the references between the components that are building the MVC pattern. This will allow the application to scale in terms of complexity because it's more efficient to manage components that have only one purpose.[EG94]

The Controller and the View are dependent on the Model, but the model itself does not depend on any of them, leading to the main advantages of the separation allowing the Model to be updated, tested independent of the visual interface that is often changing.

The Model main responsability is to encapsulate the bussiness logic with any implementation for persisting the state of the application. There can be used ViewModels that will contain the data needed to display on the view. The ViewModels instances are generated and populated with data by the controller.

The View main responsability is to display the content in the user interface. They are using the Razor view engine to integrate C# code in the HTML, the logic in the view should be minimal and is related to displaying the content.

The Controller main responsabilities are to handle user requests, work with the model and select which view to present. It’s send back the response to the user through the View, basically is controlling how the application should behave to a specific request.

Repository Pattern[10] is a technique for abstracting data access, that is responsable with storing and retreieving data. Repositories are basically classes that encapsulate the logic needed to access data sources, by focusing on common data access functionality, offering a great maintenance because is independent from the infrastructure that used to access the database from the model layer.

In the Repository pattern there should be created a repository for each aggregate, so the way of updating the database will be made through the repository, having a one to one relationship link with the aggregate. A repository is allowing the population of data in memory that is retrieved from the database using entities, then in memory they can be changed according to the use and purpose of them, after are persisted back to the database using transactions.

Below is an example scheme of the flow used by Repository pattern in order that a user post an announcement using the repositories. As we can observe there are individual repositories for each aggregate, in our case user aggreagate, announcement aggregate that has a child specification and also a value object Images where will be stored the link to the pictures of the product from the announcement, this will be the domain model layer. The User aggregate depends on the User repository and Announcement aggregate depends on Announcement repository interfaces, that are implemented in the Infrastructure persistence layer by the referenced repositories that are depending on the UnitOfWork, that is accessing the tables form the Data tier layer.

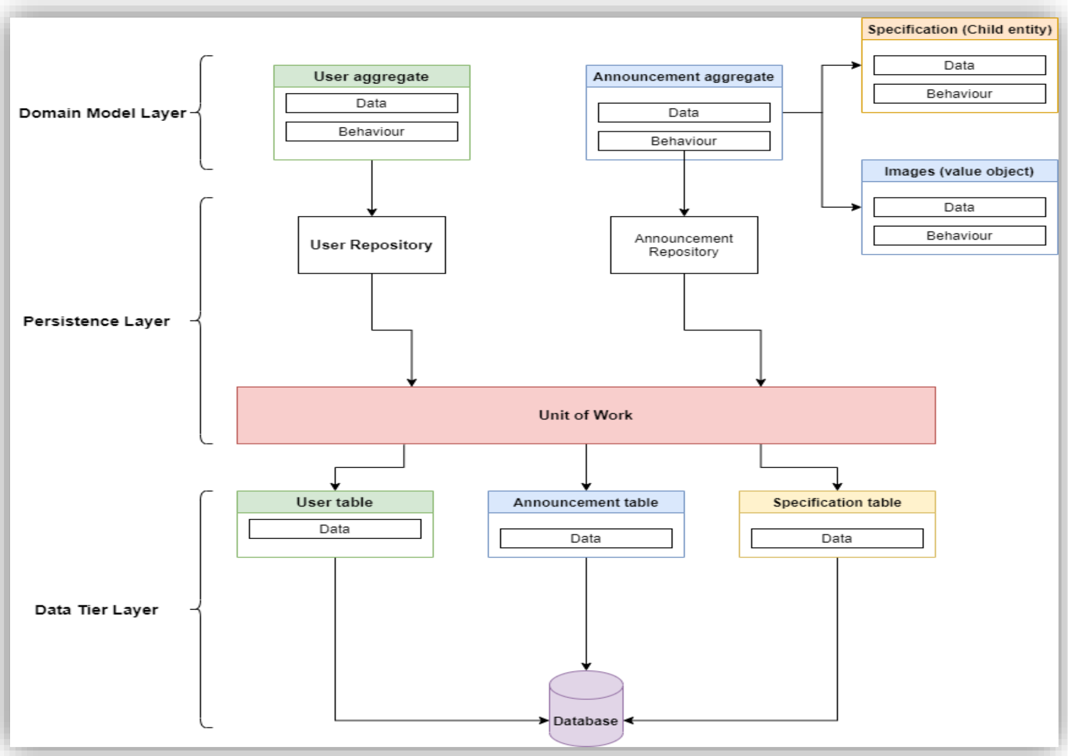


Figure 2. Relationships in Repository Pattern

2.5 Frontend technologies

2.5.1 HTML

„Hypertext Markup Language (HTML) is the standard markup language for documents designed to be displayed in a web browser. It can be assisted by technologies such as Cascading Style Sheets (CSS) and scripting languages such as JavaScript.

HTML[TF18] elements are the building blocks of HTML pages. With HTML constructs, images and other objects such as interactive forms may be embedded into the rendered page. HTML provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes and other items. HTML elements are delineated by tags, written using angle brackets”[11][12]

Figure 3. Example of HTML document

2.5.2 CSS

„Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language such as HTML. CSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript.

CSS is designed to enable the separation of presentation and content, including layout, colors, and fonts. This separation can improve content accessibility, provide more flexibility and control in the specification of presentation characteristics, enable multiple web pages to share formatting by specifying the relevant CSS in a separate .css file which reduces complexity and repetition”[13]

Below is an example of integrating CSS with HTML that will enhance the style of the head and paragraph elements, the CSS[14] code is placed between the style tags. Other ways to integrate CSS in an HTML is by placing the style inside the element, this method is called inline CSS, and other method is to make a reference to the CSS file using the link tags and href attribute where the path will be placed.

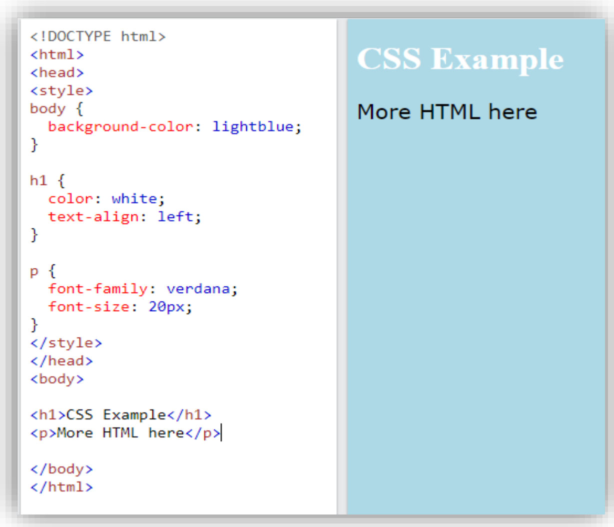


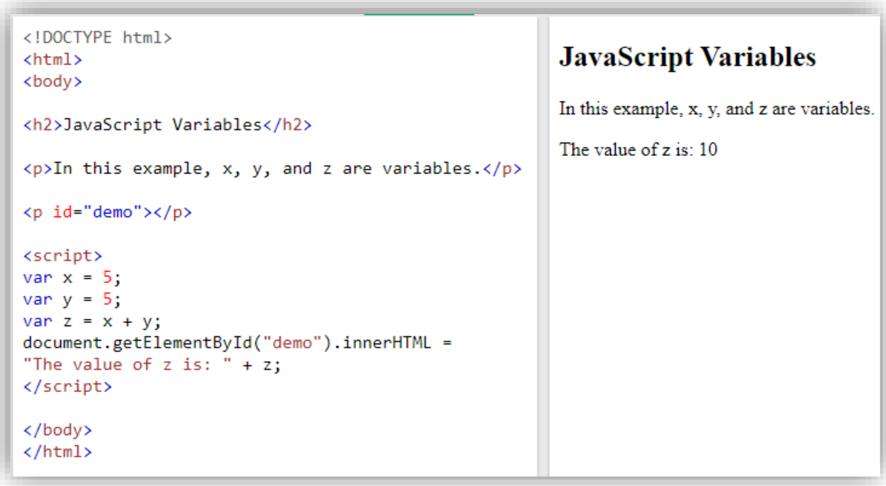
Figure 4. Example of HTML and CSS

2.5.3 Javascript

„JavaScript, often abbreviated as JS, is a programming language that conforms to the ECMAScript specification. JavaScript is high-level, often just-in-time compiled, and multi-paradigm. It has curly-bracket syntax, dynamic typing, prototype-based object-orientation, and first-class functions.

Alongside HTML and CSS, JavaScript is one of the core technologies of the World Wide Web. Over 97% of websites use it client-side for web page behavior, often incorporating third-party libraries.

Most web browsers have a dedicated JavaScript engine to execute the code on the user's device."[15][16]



```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Variables</h2>

<p>In this example, x, y, and z are variables.</p>

<p id="demo"></p>

<script>
var x = 5;
var y = 5;
var z = x + y;
document.getElementById("demo").innerHTML =
"The value of z is: " + z;
</script>

</body>
</html>
```


Figure 5. Example of JavaScript

2.5.4 Bootstrap

"Bootstrap is a free and open-source CSS framework directed at responsive, mobile-first front-end web development. It contains CSS- and JavaScript-based design templates for typography, forms, buttons, navigation, and other interface components.

Bootstrap is a HTML, CSS & JS Library that focuses on simplifying the development of informative web pages. The primary purpose of adding it to a web project is to apply Bootstrap's choices of color, size, font and layout to that project. As such, the primary factor is whether the developers in charge find those choices to their liking. Once added to a project, Bootstrap provides basic style definitions for all HTML elements. The result is a uniform appearance for prose, tables and form elements across web browsers. In addition, developers can take advantage of CSS classes defined in Bootstrap to further customize the appearance of their contents. For example, Bootstrap has provisioned for light- and dark-colored tables, page headings, more prominent pull quotes, and text with a highlight."[17][18]

I chose to use Bootstrap because is open source and has a nice set of predefined templates and UI components that are responsive, some of them are: Tables, Forms, Buttons, Glyphicons, Dropdowns, Navigation, Pagination, Labels, Alerts, Tabs, Modals, Carousels and many more.
The grid system is great and helped me a lot with allignment of the announcements in bootstrap cards that makes the display more efficient and intuitive, giving a great user experience.

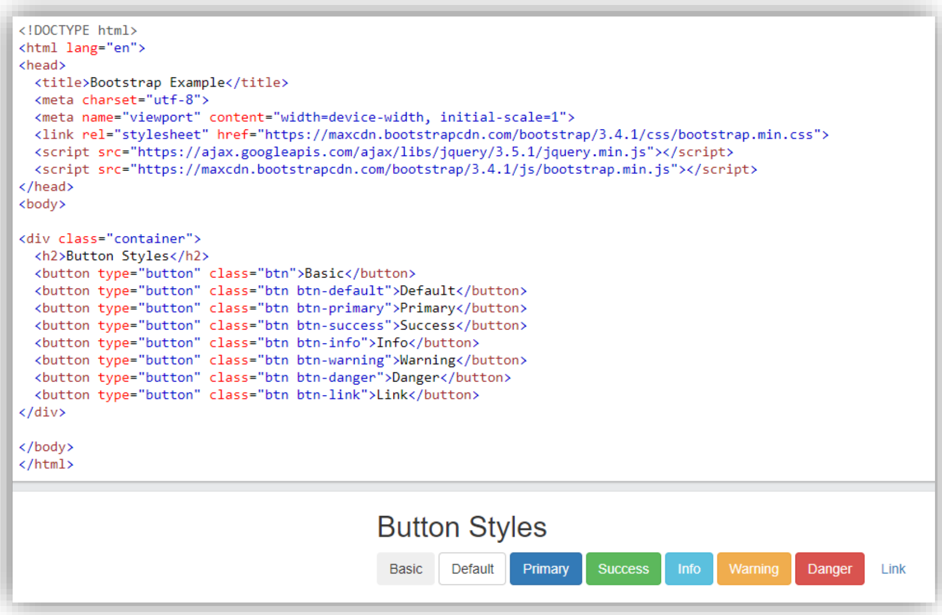


Figure 6. Example of Bootstrap buttons [17]

2.6 Microsoft SQL Server and SSMS

„Microsoft SQL Server is a relational database management system developed by Microsoft. As a database server, it is a software product with the primary function of storing and retrieving data as requested by other software applications which may run either on the same computer or on another computer across a network.

Microsoft markets at least a dozen different editions of Microsoft SQL Server, aimed at different audiences and for workloads ranging from small single-machine applications to large Internet-facing applications with many concurrent users."[19]

„SQL Server Management Studio (SSMS) is a software application first launched with Microsoft SQL Server 2005 that is used for configuring, managing, and administering all components within Microsoft SQL Server. It is the successor to the Enterprise Manager in SQL 2000 or before. The tool includes both script editors and graphical tools which work with objects and features of the server.

A central feature of SSMS is the Object Explorer, which allows the user to browse, select, and act upon any of the objects within the server.„[20]

I used Microsoft SQL Server because is easy to install, with only a quick setup executable everything was set and I could run the SQL Server instance on localDB without additional configurations from the command line. It has a user firendly instalation GUI that comes with a lot of instructions and how to run it properly after the instalation is complete. Other important aspect is represented by the automatic updates routine that is significantly reducing the maintenance cost and keeps the database up to date with the latest features and security options.

Another aspect is the enhaced performance due to built in data compression and encryption techniques, providing high performances. Also the security is important in choosing the platform that will host the database, SQL Server is very secure with enhanced encryption algorithms that will make it impossible to crack the security layers, significanty reducing the risk of attacks.

I chose SSMS (SQL Server Management Studio) because first of all is free and easy to use, being an important tool for creating and modify databases. Other main features that SSMS has are: adding database objects with tables, views and stored procedures, testing database objects with the help of some external tools, query database in order to get results and check if we get the correct results.

There can be made restore and maintenance plans on the database, also importing and exporting data is made simple and efficient, and can be restored or access backups of the database to have an extra assurance of safety against data loss or corruption.[21]

2.7 Draw.io

"Diagrams.net/draw.io is an open-source technology stack for building diagramming applications, and the world's most widely used browser-based end-user diagramming software.
Diagrams.net is a trademark and draw.io is a registered trademark of JGraph Ltd. JGraph Ltd is a company registered in England that develops and owns the software, runs the diagrams.net and draw.io sites and owns the diagrams.net and draw.io brands." [22]

The main reasons I chose this diagramming software because is free and offers a variety of diagrams and templates at the fingertips that are really useful in implementing a software project.

Some of the diagrams that can be made with Drawio are: Class diagram, Flowchart, Entity Relationship diagram, Kanban boards, Sequence Diagram, Activity diagram, Class diagram, Use Case diagram, Presentation pages diagram, UML diagrams and many other customizable diagrams.

The diagrams can be exported in various formats that will be a great fit in many environments, for instance it can be exported as an image with formats like: PNG, JPEG, SVG but also formats like: PDF, HTML, XML, URL and if I need to edit the diagram later in the editor then I should save it with the extension .drawio that will allow opening it anytime with the Drawio tool for further editing.

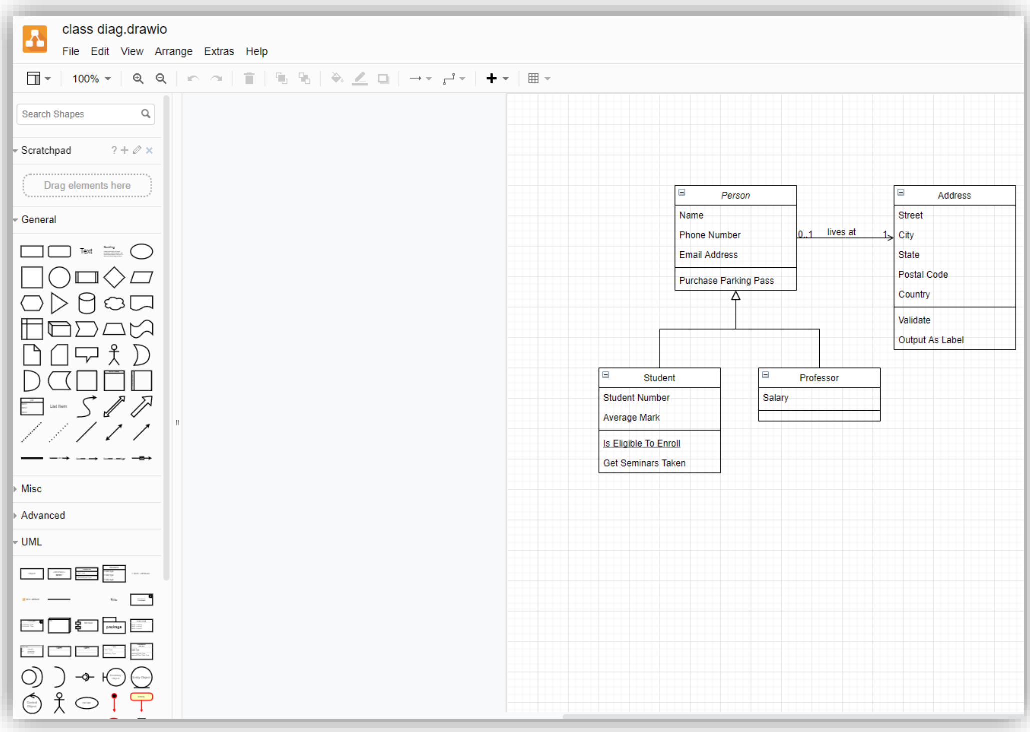


Figure 7. Drawio workspace editor

Here is an example of class diagrams modelation inside the Drawio editor that contains a lot of features and intuitive drag and drop actions in order to have the best user experience, I really like this tool a lot, and definitely making diagrams is more easy now.

2.8 GitHub

„GitHub, is a provider of Internet hosting for software development and version control using Git. It offers the distributed version control and source code management (SCM) functionality of Git, plus its own features. It provides access control and several collaboration features such as bug tracking, feature requests, task management, continuous integration and wikis for every project."[23]

The main features that recommend GitHub[24] to be one of the best version control hosting provider are:

- **Well documented** – GitHub has a really great documentation in the help section that has a ton of articles related to all sort of problems, issues and also improvements that has something to do with git.

- **Projects Showcase** – it has all the required tools to showcase the work and projects that I work on, being an advantage when applying to jobs. It is well structured and can make the repositories public or invite through link other users that should have access to that repository. Also there can be added some pinned and starred projects, that will definitely are more highlighted to a visitor.

- Follow changes across versions – often at one project are working multiple individuals and is important to keep tracking the changes that everybody does, GitHub is following all the changes and also there is a version history where can be made rollbacks to previous versions of the project, in this way the history versions are not lost when making a new one.

- Integration options – GitHub can be integrated with platforms such as Google Cloud or Amazon

- Open source – the projects and applications uploaded on GitHub are open source if are set as public, and can contribute to developing and learning technical skills just by checking other peoples work and sometimes just a great example can clear the things up.

- Markdown – this feature is allowing the user to access a simple text editor to write well structured documents, being a way to style text on the web. The user is in control of the visuals, formatting words, adding images, creating lists and other useful functionalities. The Markdown feature is used on Gists, comments in the issues panel also in Pull Requests and files that has the extension *.md* or *.markdown*

2.9 Existing auto markets

2.9.1 Examples

There are a lot of cars trading platforms nowadays where users can buy and sell used cars, some of the top three best sites worldwide are:

- Cars.com[26] – this site is the best car trading platform that is a standard for all the other ones, because their car listing is very variate and widely, it has a huge inventory where sellers can even deposite their cars for sell there until is sold, or there is the opportunity to let all the hustle in finding a perfect buyer in to their hands. Also the announcement a user makes on this webistes can be listed on their other domains like Auto.com, PickupTrucks.com, NewCars.com The site gained popularity among car buyers and sellers that made it a bit harder to stand out from the crowd and be noticed by the buyers with a free plan, but there are paid plans that are helping sellers with selling their cars.

- AutoTrader[27] **–** this site is known for advanced search tools that are making buyers life a lot easier in finding the right car for them at the correct price, evan if they are looking for a specific model or option trim, is a really high chance to be there. Sellers are enjoying this website as well because have feature that can guarantee the money-back that is working on the principle: if they are not making any sales, then no charge is applied. There is also the possibility to sell directly the car to a dealership or to

the site, making the process to sell the car evan faster, but of course they are buying it for less money.
- eBay Motors[28] **–** it gained popularity on the way, because in the beginning peoples were sceptically in buying or selling cars on eBay, but this is actually a great ideea especially if the seller wants to find local buyers to avoid the transport costs. Local listing are free and are selling pretty fast due to eBay popularity and high organic traffic. A downside in using this platform is that the buyer bids are non binding, meaning that he/she can abandon the bidding process at any moment.

In Romania the best websites for selling second hand cars that inspired me in choosing this theme on the project are:

- Autovit – this platform is dedicated not only to sell second hand cars but also new cars directly from a dealership. It is bounded with the biggest announcements website in Romania: Olx. There are numerous filters that can be applied to find the best car, but also there is a comparing system that shows the buyer the real value of the car depending on other similar listings. There is also the posibility to make a buy in leasing using safe and trust worthy facilities. A mobile version of the webiste is available to make it more accessible to all kind of internet users.
- Carzz.ro – is a great option when buying or selling a car, because it has many filtering options, large images, detailed information about the seller and the car and the support is good. There is also a recommender engine that is recommending cars announcements based on the user searches, announcements added in the favourite list that is making easier to link the buyer and seller. There is also a mobile version of the website to redirect more traffic to it.
I like this website a lot due to its simple but effective design and layout that made me to create my project around it.

2.9.2 Main features

All these websites for selling online used cars have a lot of common features that are making them the best platforms in their niche. So we will take a look in detail and analyze the main functionalities that a good car trading platform should have:

- many filters – the purpose of the site is to match buyers and seller and this can be achieved also with the filtering option that will narrow down the listed results in order to find the best suited configuration. Having a variaty of filters can increase the chance in finding more quicker what the buyer is looking for having a better user experience and the platform will generate more sales.
- user firendly interface – this is an important aspect because the interface is the connection with the user, so it should be intuitive and with all the information they need to feel confident in using the platform.
- great car listing layout – the main feature is to have a list with all of the approved announcements that

will be a lot, so the layout and elements alignments should be done correctly with responsive designs.

- **many details about the car** – the buyer wants to know as much as possible about the car that he/she is willing to buy, so by having a lot of details about the option trim, make, model, horsepower and other aspects is really crucial, and will avoid unnecessary seller contacts to just ask some basic questions about the car that could be on the website and the focus will be more on more important things like buying/selling.

- **seller contact information** – this is important because what is the use of a good car announcement without seller information, so is best to have multiple ways to contact the seller: via email, phone number, message.

- **message system -** communication is the key in making transactions, and having an integrated quick chat system is really great and can become in handy when the buyer wants some more details before making a phone call or reservation.

- **recommender engine** – is a system that is choosing special announcements for a particular user based on his searches, filtering, favourite list, representing the user interests and preferences. Recommending this type of announcements, the chance in connecting a buyer and seller is significantly higher, increasing the platform sales, traffic and popularity.

- **blog** – the website traffic can be increased significantly by integrating a blog where are posted car related articles, but also information about the website can be posted with guidelines and maybe future updates.

- **announcement views** – it is really important for the seller to know how the announcement is performing, and by having the posibility to track the number of views they get is really great. This will help at improving the overall website announcements because the sellers will see what is really working or not, just by the number of visitors they get.

- **good support** – is an important aspect because on any platform there can have difficulties especially when it's about transactions, or account problems, issues with a seller, scam announcements and so on. There is the possibility to contact a supervisor of the website like an admin or moderator that the users can address.

- **report announcements system** – everybody can post some announcements, but there should be the feature to report any of the announcements or the users if are providing false or scam information, and a supervisor should investigate the activity of that user that could lead in permanent ban.

- **admin approval system** – after an announcement is posted, it should not be posted immediately in the car listing, because can have false, incomplete or inappropiate information that should be restricted and not shown on the webiste to not alterate the user experience.

- **banning system** – almost every online market has banning system implemented that will restrict a user from some of the rights and features or permanently blocking the access in their account.

- **more categories** – the industry is growing and should cover a lot of categories and niches, to have the announcements more grouped, and when a new one is created to force the seller in fitting its announcement in a category, so there are not announcements without a category. This will play an important role also in filtering the results, the buyer will definitely want to filter the results by a category to make the search more efficient.

3 PROJECT REQUIREMENTS AND SPECIFICATIONS

3.1 Functional requirements

In this section I will elaborate in detail the application functionalities based on the four main roles present in the project: Admin, Buyer/Seller, Moderator and Guest.

The application will restrict access to different areas and features based on the current logged in user. In this way will be prevented unauthorized access to the admin panel of the application, and having the possibility to give more rights and features only to the website supervisors.

In case that a user tries to access a feature beyond it's current role rights, a proper message will appear that will signal the user that he/she is not allowed to access that side of the application. But precautionary measures are already taken, that consists in hiding from the unauthorized users the buttons, tables, dropdown options and other UI elements in the graphical user interface that lead to other advanced features only for supervisors. In our case the supervisors are represented by the Admin and Moderator role that have access to additional functionalities helping them in monitoring, managing normal users activity and also the website normal behaviour.

Buyer/Seller (main application user)

This is represented by the person that registered on the website an account and had a successfully login. This role is automatically generated when a new account is created, being the main user of the platform. The functionalities related to this role are presented below:

- **Post announcements** – the user can post an announcement on the website that is basically a form that requires multiple choices and user input boxes that will store in the database the fresh announcement. The seller must provide many information about the car that he/she wants to sell like: title, category in which the product will be placed, the price in EUR currency, the availability date, meaning the fact that maybe the car is not ready yet for testing by a potential buyer but at that specific date everything can be arranged and available. Other information provided when adding and announcement should be the location where the buyer can meet up in person and investigate the car but also is used to estimate the distance between the seller and a potential buyer. Announcement description plays a big role in successfully selling a car, here is a section where the user is free to write details about the car, why he/she is selling it and some sentences that could attract a potential buyer.

  When dealing with online markets for second hand products, the buyer is interested if the price is negotiable or not, so the application should have a dropdown where the seller can sellect if the price is negotiable or not, also when listing the announcement if the price is negotiable then it should appear near the price in green colored text and if the price is fixed then near the price should appear the fixed text in color red. In this way there is no confusion created and the buyer knows what to expect and both of them are on the same page.

Another important section in adding an announcement should be the specifications, basically here will be other sets of user inputs and dropdown options but are referring to the characteristics of the vehicle like: Make, Model, Year, Mileage, Power (HP), Body type – here should be a dropdown where the seller can choose from cabrio, sedan, coupe, hatcback and SUV. Other fields are Number of doors, engine size, emissions, color, a dropdown where the seller is choosing if the car for sale is full option or not and another one where is selected the fuel type: diesel, petrol, hybrid, GPL, electric.

There should be also some checkboxes that if are checked is meaning that the car has that option being diplayed on listing as yes, otherwise will be displayed as no. Option trims like: ABS, warranty, cruise control, full electric windows, vented seats, heated steering wheel, ESP, log history, dual zone climate, heated seats, electric mirrors, and if had accident or not.

When creating an announcement there should be the possibilites to add multiple images to allow the seller to post pictures with the described car to complement all the details he/she provided.

The seller should not be allowed to post an announcements without completing essential information about the vehicle for sale, so there should be mandatory fields like: Title, category, price, location, is negotiable or not, description, make, model, year, mileage, power and also the adding images are mandatory because we dont't want announcements without images. This will prevent creating confusion or to waste buyers time or memory performances by having incomplete and useless announcements.

- **Delete announcement** – the application user can have the possibility to delete any of his announcement anytime, with the reason that he/she sold the car listed or just doesn't want to sell it anymore. The user can delete only his announcements from the announcement panel, even if are not yet listed in the home page of the application, meaning that the announcement was not approved or is still in pending review from the admin.
- **Update announcement** – the seller can update any of his announcements anytime, providing more information or changing the title, description, price and other key factors that could lead in a potential sale.

- **Chat system** – The application should have a messaging system between users that can be accessed by the main application user, admin and moderator. It will consists in having a dedicated section called Inbox where every user can see in the left a list with users that he/she communicate or linked displaying essention information about the recipient for instance the email. And in the right side to appear the conversation messages between the logged in user and a recipient, that will change depending on which conversation is selected.

  The messaging system is an online chat meaning that the messages between two users should be received almost instantaneous without the need of any user to refresh the conversation or the whole page. On each message sent and received should be a timestamp that represents in hour, minutes, date and year when the message was sent.

  Another feature on the chat system is the possibility to send the messages from the message box just by hitting the enter key, but also there should be a button that can send messages when clicked.
- **Update account information** – the user should be able to update the account information anytime to keep updated the account with recent data. There should be the possibility to update also the password and phone number.
- **Track posted announcements** – each user has an announcement panel where he/she can track the announcement performance and reach, having detailed information about each announcement he/she posted. Essential information are the created date, announcement title, category, the status that will inform the user if that specific announcement was approved or is still in pending. If an announcement status is „Pending Approval" is meaning that the admin or the moderator did not approved yet the announcement or just simply did not checked that announcement yet, but if the status is changed in „Approved" is meaning that one of the admin or moderator approved the announcement and now is displayed on the homepage along with other approved announcements form other sellers.

  When a new announcement is created, automatically will have the status set to „Pending Approval" and wont be shown on the homepage, waiting for the admin or moderator approval.

  On the announcement panel are present also three main action that a user can perform on announcements: see a preview of the announcement, update the posted announcement or delete the announcement without further questions.
- **See all the announcements listing** – every user is able to see all the approved announcement in the homepage of the application displayed as bootstrap cards on well arranged grids.

The car listing on the homepage should provide enough information for the user before evan clicking on it, regarding the price, date when that specific announcement was created, seller full name, category on which the vehicle is taking part of, and also the number of views on each announcement card. There should be also the possibility to contact the seller directly through a button.

If the user decides to click on a particular announcement then will be redirected on another page with car details where are basically all the information completed by the seller when he/she created the announcement. The pictures are enlarged and with the feature that the user can navigate freely through the images carrousel paying attention to every detail. Furthermore there should be also the seller contact information where the user can contact that particular seller through many means of communication like: email, phone number, or to start chatting using the built in messaging system from the application.

- **Filtering results –** the application user is able to filter results depending on various criteria like: Title, category, price, model, make, year, mileage and power. These are enough information to customize the announcement listing based on user needs and preferences.
- **Contact the admin –** There should be the posibility that a user can contact the admin directly in the app using the chat system and address him all the issues that he/she encounters.
- **Report other users** – On each announcement details in the seller contact area should be the option to report a user due to false or scam announcement posting, activity, or insulting on the private chat, so is essential for any user to be aware that there is an option to report another user.
- **Newsletter –** When the website is opened, the user should be redirected to the newsletter, that is basically a list with fancy displayed news that contains information about the website, guidelines, rules, hints and other car related stuffs. The user can see all the news in descending order depending on the posted date, in this way the application is making sure that the users are always reading latest news and updates by constantly popping out the fresh ones.
- **Recommended announcements** – Authenticated users should receive recommended announcements based on their interests, that are expressed when they apply filters, so the application should collect filter and search parameters in order to start displaying similar announcements in their homepage in a special section called „Recommended for You".
  All of the recommended announcements are customized for each user and only him can see the recommended announcements.

Admin

This is represented by the supervisor of the platform, that has additional features and rights that is helping the webiste to have a better community and increase the performance and efficiency of others features and functionalities that are designed to assist normal users in finding the best car for them.

The main functionalities that an admin role has access to are presented below:

- **Review announcements –** when a new announcement is made, is not displayed on the homepage because it should wait for the admin or moderator approval. So the admin is receiving all the fresh created announcements in the announcement panel and can investigate them in detail and see if there are some improvements left to made, or maybe are incomplete ir just wrong. There should be the possibility that the admin can see a preview of the announcement before is listed, with all the details, images and seller contact information.
- **Delete announcements –** the admin has access to perform deleting of any of the announcements on the website, the ones that are already approved and announcement that are still waiting the approval. When an announcement is deleted will disappear from the home page and also from the seller announcement panel.
- **Display all users –** it has the possibility to see all the platform users account with information like: full name, email, is reported, the person that reported it, if is blocked or not and also it has some actions on the users presented below.
- **Ban users –** is the feature represented by the admin right to block a certain user from posting new announcements on the page, a proper message will notify the banned user that he/she is not allowed to create announcements anymore and is asked to contact the admin for further details.
- **Delete users –** another feature from the users panel is the deletion of an user account, then the user in cause will not be able to login using those credentials because the account was permanently deleted due to guidelines violation.
- **Change users display name –** the admin is able to update user display name if is inappopiate or is not in conformity with platform rules and guidelines. Modifying the name of the user, will be updated as well on all of the announcements of that particular user, that are displayed in the car list on the home page.
- **Message sellers –** it’s obvious that the admin should be able to message any user on the platform to keep a good communication and to make sure everything is going smooth as expected.
- **Create new category –** this is a nice feature because the admin can create new categories that will be able to be used by the sellers when creating or updating an announcement. By adding

multiple categories on the way is needed in order to cover all the niches on the online auto market industry.

- **Post news in newsletter –** there is the possibility to post news on the front page of the website that everybody will see them, can be related to platform rules and guidelines, potential solutions to common issues, informative articles and so on. The adding of a new post consists in a Title, Sub-title, and content where the admin/moderator can write freely and then will be saved and displayed. There is also a timestamp on each news post that will mark the date and time when was created.
- **Delete news –** it's great to have the feature to delete some news easily and efficient using an intuitive trashcan icon that is hidden from normal users.
- **Quick search –** on the announcement panel there should be search box where the admin can make a quicksearch on announcements that are waiting the approval. There should be the search box also in the adding of a new category to quickly navigate and find the already listed categories. The search should find matches in any of the columns of the announcements table or categories table, in this way the search is really accurate and very usefull when having more items.

Moderator

This role is represented by the second supervisor that has less accessibility features then the admin but is accelerating the website flow by doing some of the admin work to not get agglomerated and lead to low user experience and can affect platform performance.

Below are described the essential features that are accessed by the moderator:

- **Review announcements** – similar to admin also the moderator can review announcements, making the process faster and efficient that will lead to higher work load and less waiting time for announcement approval, created by the seller.
- **Post on newsletter –** it can become in handy to be able to post news because the admin can be busy sometimes and the application users should be kept informed at all time wahat updates will come soon or other important aspects and guidelines**.**
- **Delete news –** if there are too many news or becoming irrelevant then is great to delete these announcements, and sometimes could be too many and having a second role that could do that is helpful.
- **Contact the admin –** the moderator can use the chat system to contact the admin as well, maybe he/she need some advices in managing the website activity, in reviewing an

announcement, or what content it should be posted next. So the possibility that the moderator can contact the admin is really usefull.

- **Apply filters**
- **See car listing**
- **Contact sellers**
- **Update profile information**

Guest

This role is represented by the person that didn't logged in successfully, and has restricted access on the website. If the guest is trying to access authenticated user features, he/she will be redirected to the login page in order to authenticate in a valid account to change in the role of buyer/seller.

The main features that a guest can still access are:

- See all car announcements
- Apply filters
- See newsletter

3.2 Use case diagrams

A use case diagram [29] can illustrate in a more visual and practical way the functionalities and requirements mentioned in the previous sections. This diagram indicates all the actions and features an authenticated user, guest, also admin and moderator can do with the use of this application.

A use case diagram can summarize the details of the application user types, often knows as actors, that will interact with the app resulting a different behavior depending on the use cases. This are describing a function that a system will perform to accomplish current user's goal and purpose.

Use case diagrams are helpful before a project implementation starts to make sure that every user type that will use the system will have a well-defined path of actions that are representing a form of system requirements.

The use case diagram for the Buyer/Seller and the guest is represented below:

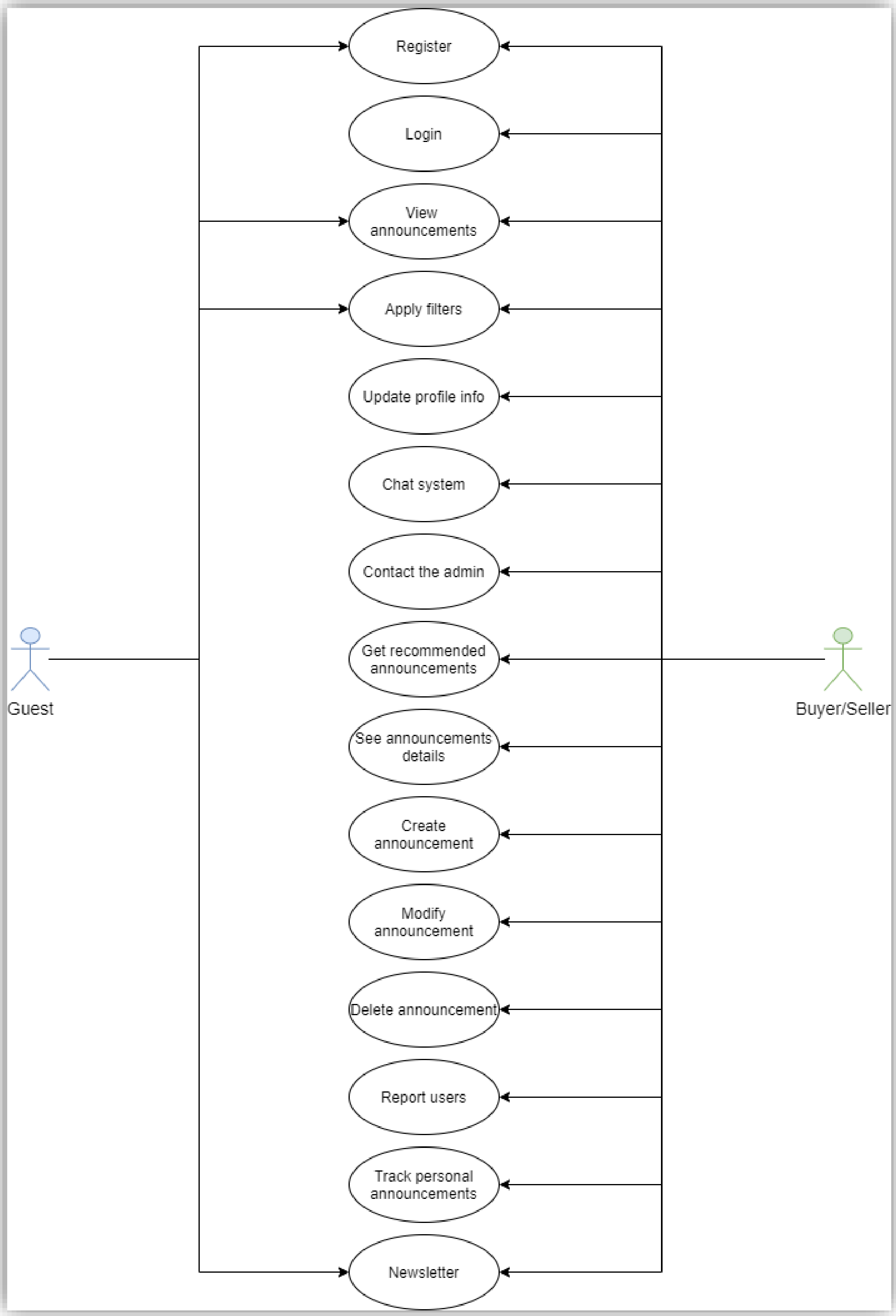


Figure 8. Use case diagram for guest and buyer/seller

The use case diagram for the Admin and Moderator is represented below:

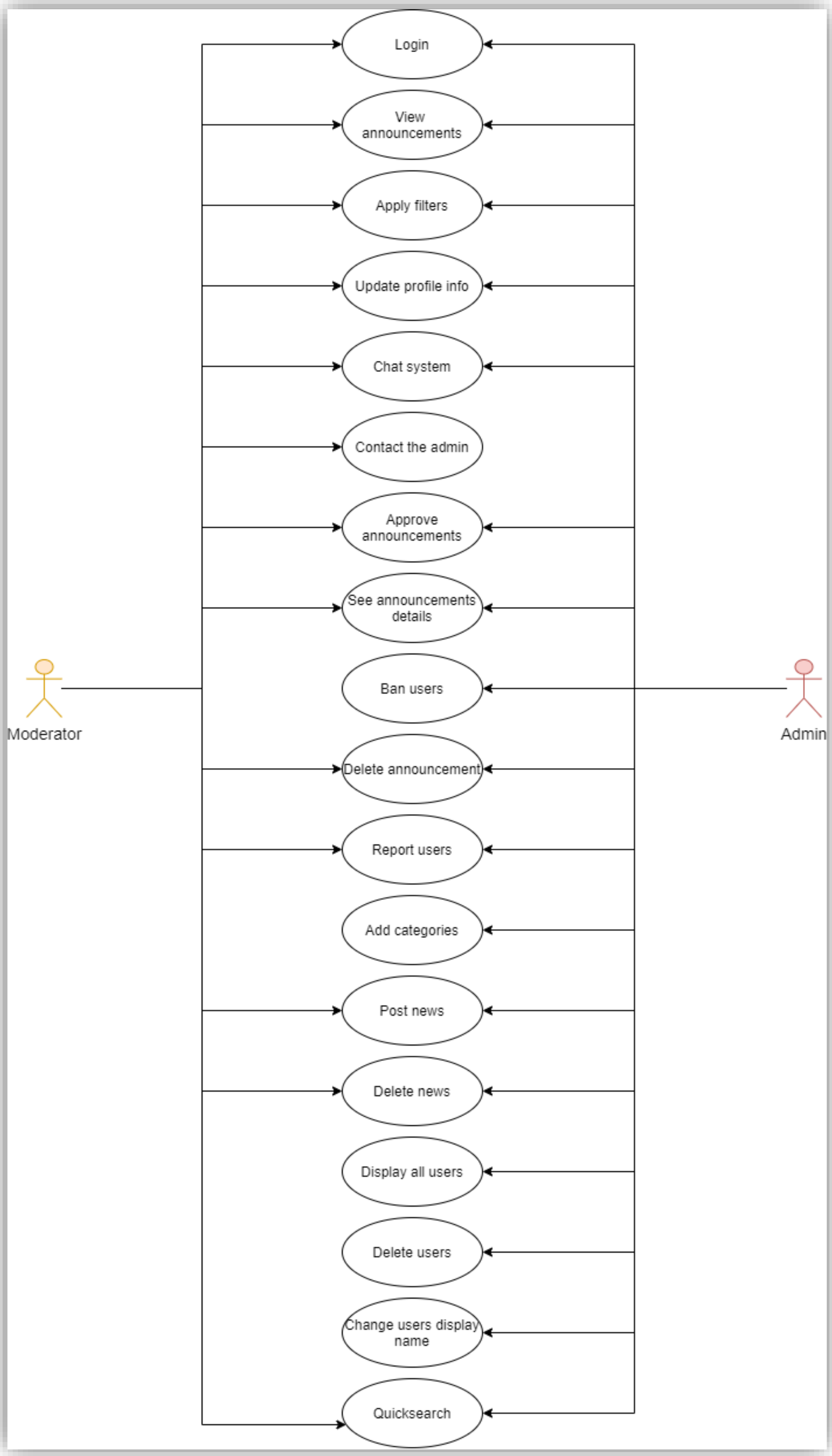


Figure 9. Use case diagram for admin and moderator

3.3 Activity diagrams

An activity diagram is representing a behavior diagram that describes each step and possible outcomes of an action. In a way these diagrams are similar to state machine diagrams, which are behavior diagrams as well, but activity diagrams are more focused on the user interaction with the application. Because these types of diagrams are similar admin and moderator, guest and buyer/seller, I will represent in the next figures the activity diagrams only for the application user (buyer/seller) and the admin, the other roles are just derivations of these two main roles.

Below is presented the activity diagram of the Buyer/Seller role:

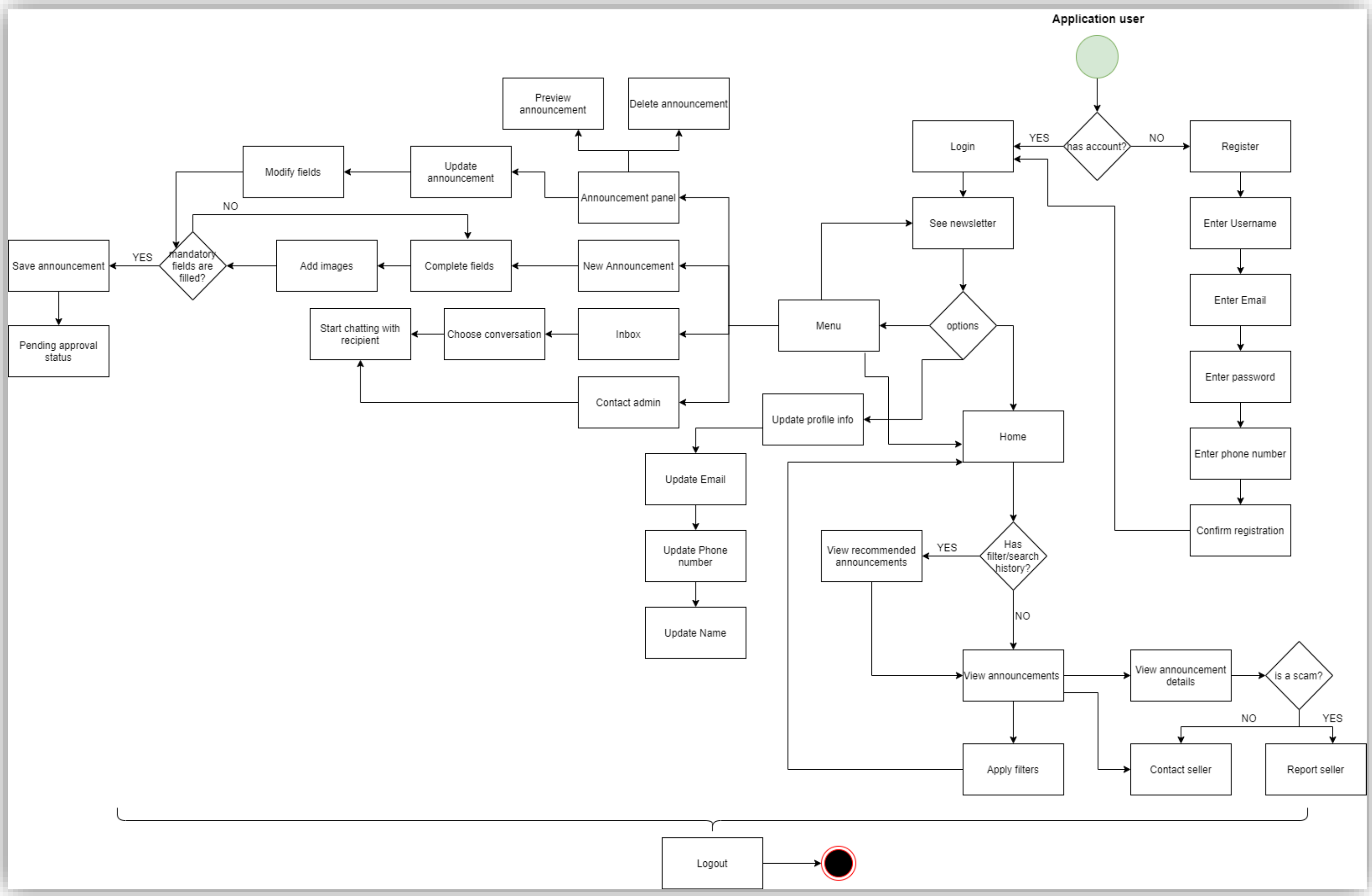


Figure 10. Activity diagram for application user

Below is presented the activity diagram of the Admin role:

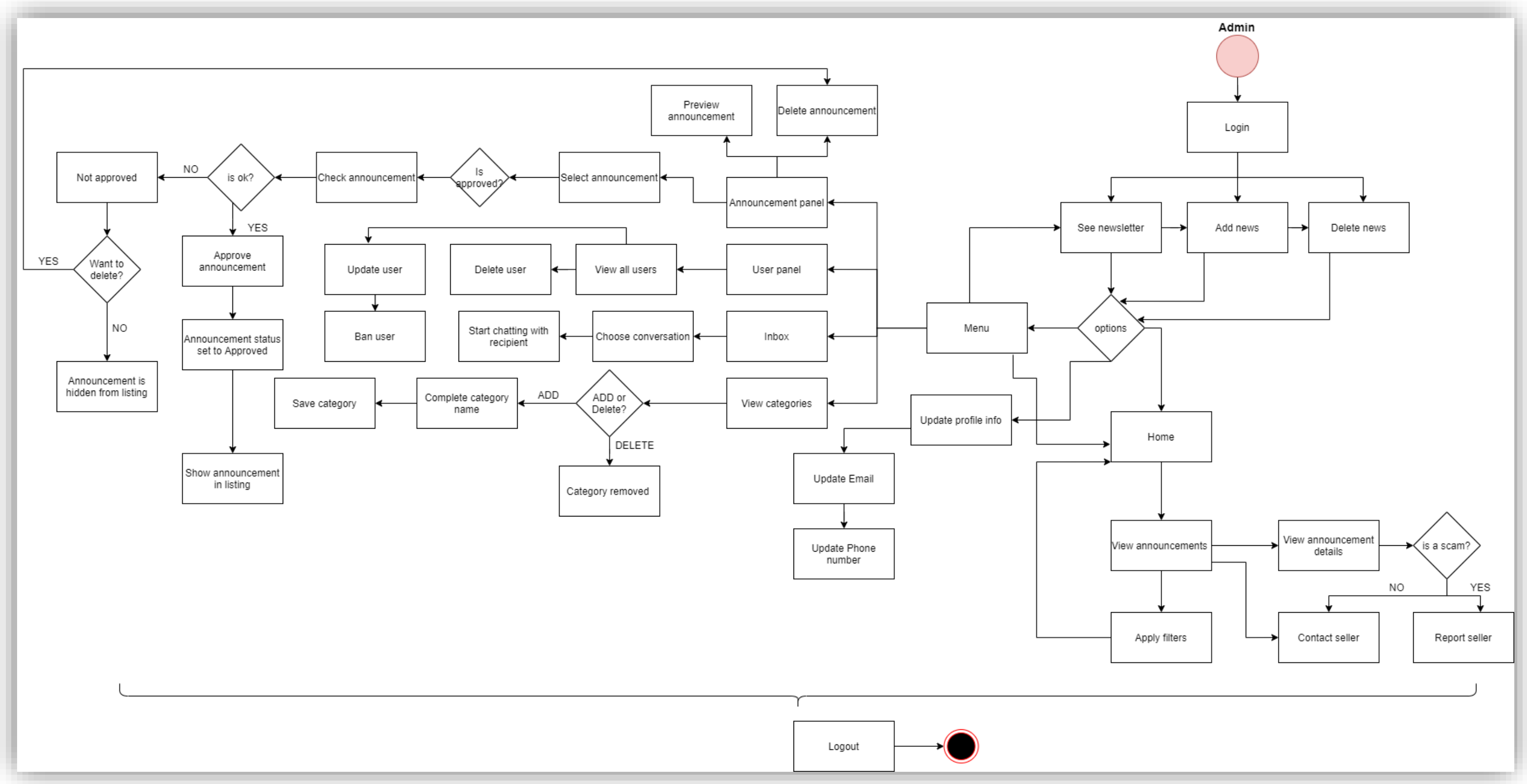


Figure 11. Activity diagram for Admin

3.4 Class Diagram

„The class diagram is a graphical notation that is used to visualize o object oriented systems. A class diagram in the Unified Modeling Language (UML) is a type of static structure diagram describes the structure of a system by showing:

- Classes
- Attributes
- Methods
- Relationship between objects"[30]

Class diagrams are really helpfull because are showing how the entity models should be implemented with the proper attributes, methods and relationship between objects that will clear some of the implementation doubts, directly from the designing stage. The relationship between classes are described by the following types: Assiociation, Inheritance, Realization, Dependency, Aggregation and Composition.

The class diagram for my application is looking like this:

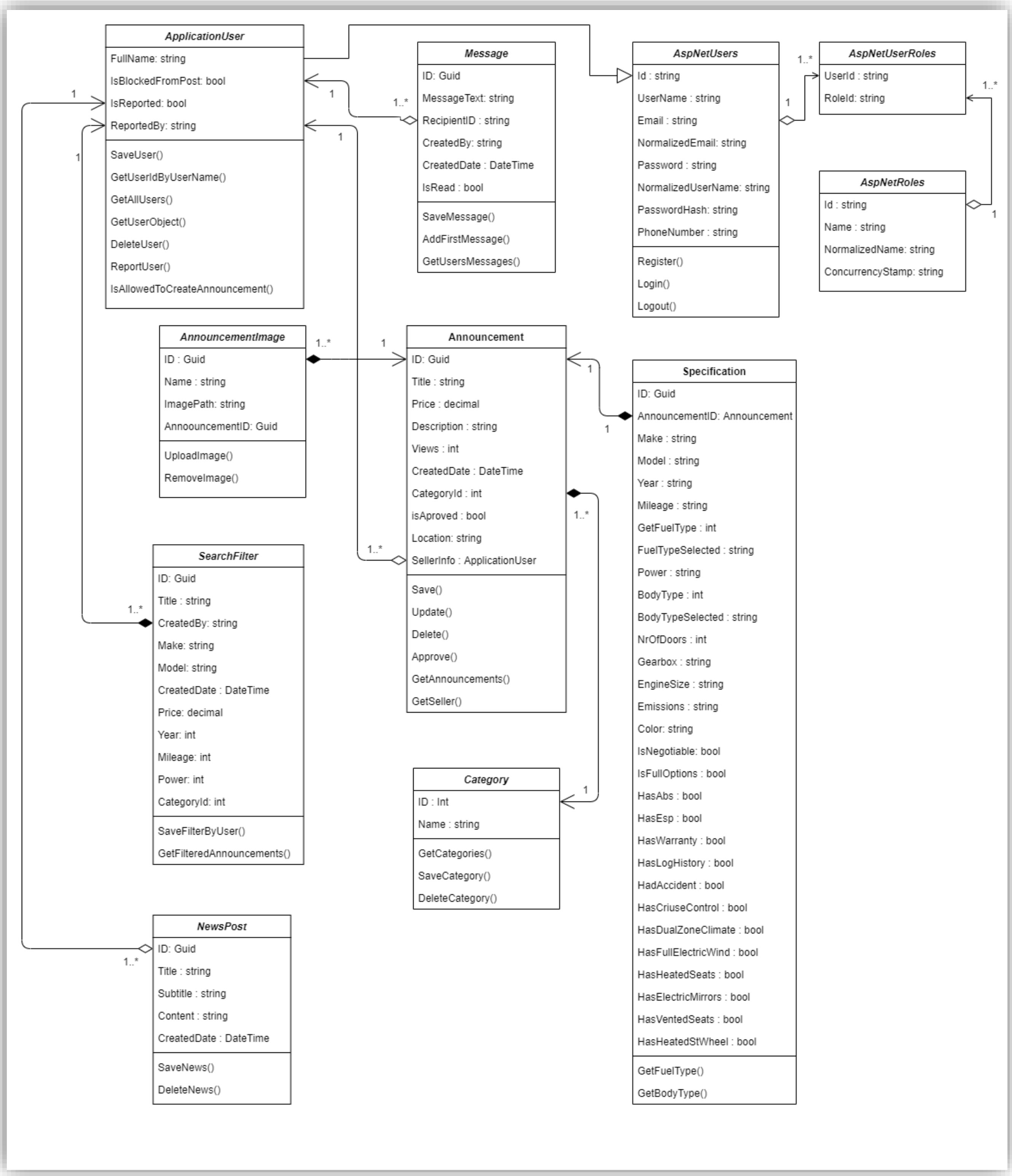


Figure 12. Class Diagram

3.5 Entity Relationship Diagram

The Entity Relationship Diagram (ERD) is showing the relationships pf entity sets that builds the database. An entity is basically an object with data and a set of entitites is a collection of similar entities. These objects can contain attributes that are defining its characteristics.[31]

Below can be seen the ERD for my application, in this way it is observed the database structure also:

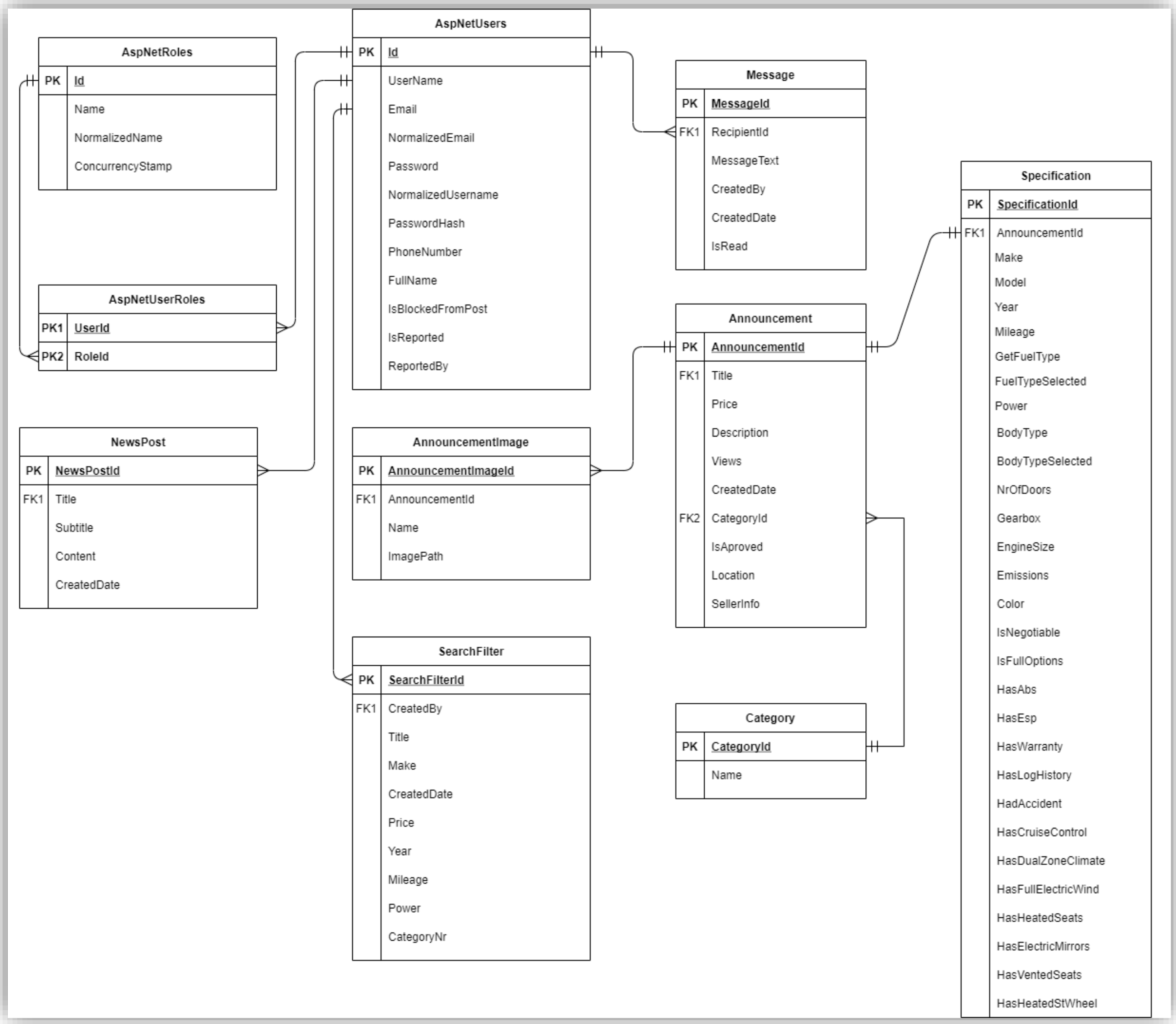


Figure 13. Entity Relationship Diagram

We can observe that the ApplicationUser class is not generated in a table because it inherits the parrent class IdentityUser and when the database will be generated the fields from ApplicationUser, will be created inside the AspNetUsers table, like an extension. In this way the ApplicationUser can benefit from all the rich features of ASP.NET Identity instance.

3.6 Application architecture

The project architecture consists in three main layers:

- **Data access layer:** is abstracting the database in order to have more flexibility over accessing and storing data efficiently and not impacting the bussiness logic of the application. So instead of retrieving an entire row from the database, it will return a model object with the corresponding properties that can be handled more safely.
- **Bussiness logic layer:** is basically working as a connection between Data Access layer and Presentation layer. All the knowledge passes through the Business Logic Tier before passing to the presentation Tier. Business Tier is that, the sum of Business Logic Layer, Data Access Layer and Value Object and also other components accustomed to be added at the business logic.
  The business logic uses the Repository pattern that allows better further modifications and an excellent application maintenance over time because the functionalities are modular, using interfaces to create the implementation more efficient.
- **Presentation layer:** this is the highest level of the application, it displays information sent form the other layers consisting in processed data shown in the front-end.
  The authentication system filters the Presentation tier counting on the role of the customer, there are four main roles: Admin, Buyer/Seller, Moderator and Guest, having different access type to the database and features. For example, only to the admin are displayed the possibilities to post announcements, delete users, review announcements.

Below will be displayed a diagram with all the application tiers and the relationship between them:

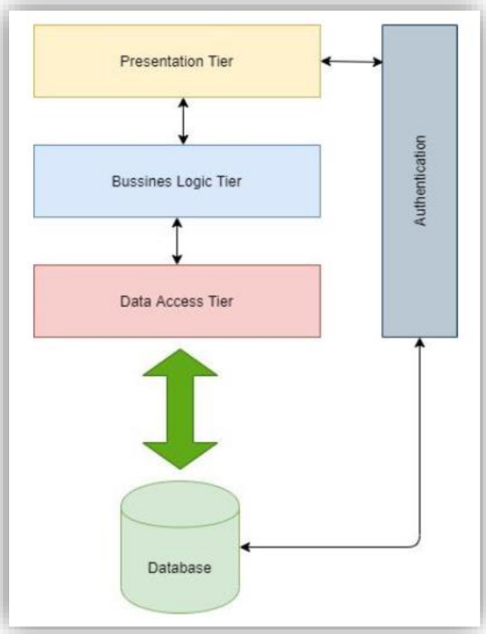


Figure 14. Application layers

4 PROJECT IMPLEMENTATION

The implementation of the application is following closely the functional requirements, design patterns, UML diagrams of the application presented in previous chapters and application architecture

I will begin with the project structure and a full flow schema to ilustrate in which manner the functionalities will be developed. Then will be described in detail the user interface implementation with screenshots and frontend code. And last but not least we will go in more depth to describe the application logic and development of all the features and functionalities described also with backend code snippets and screenshots.

4.1 Project structure and flow

Taking in consideration the three layered architecture, the Repository pattern and MVC design, I structured the project in three modules and another one that represents presentation tier with views and controllers[AF20]:

- **Racooter.BussinessLogic** – here will be placed in a separate folder all the services of the app that are implementing the reppositories, containing all the application logic
- **Racooter.DataAccess** – here will be placed multiple folders that will keep in order all the files related to accessing the database and its abstractization process.

  - *DbContext* – here will be present the database context of the application that will generate the actual database tables using code first approach

  - *DbInitializer* – I am using ASP.NET Identity for the authentification system so I will generate some predefined accounts with different roles assigned to them when seeding of the database occurs. So in this folder will be the file that can generate entries when the database is created.

  - *GenericRepository* – as I speciffied I am using repository pattern, so here will be placed the the base repository with main functionalities like CRUD actions that all the repositories will implement.

  - *Migrations* – here are automatically saved all the migrations are made on the database in order to have migration history where we can rollback to previous versions of the database generation.

  - *Models* – here are all the entities that will result in generating tables in the database

  - Repositories – in this folder will be found the repositories that implements the application logic and are using the generated correspondent objects from the DbContext, that can perform various actions on the data.

- UnitOfWork – this is used to unify multiple operations that are done many times, in just one single transaction. Meaning that for a specific application user action, all the transactions like insert, update, delete are done in a single transaction that doing multiple database transactions.[32]

- **Racooter.DataTransferObjects –** data transfer objects(DTO)[33] are used to transport data between the processes making the costs of calling the web server more efficient, transforming the usual multiple calls in a single one, storing the essential information that needs to be passed on to the next layer. DTO objects does not implement any behavior only storage, retreieval and serrialization/deserrialization.
- **RacooterCarTradingApp –** this is the main application that is implementing the view, controllers and setup options for the project.

  - *root* **–** the root will contain different folders like:

  CSS – here will lay the application styling files written with css.

  Js – here is present the site.js file that will contain all the Ajax requests to the api. controller on each action the users makes in the front-end.

  Images – here will be placed all the images from the announcements, also when adding new images is better to place it in this root folder.

  - *Areas* – here are placed all the pages related to the ASP.NET Identity and validation scripts.

  - *Controllers* **–** here will be placed the controllers for the views that will access the back-end server written with C#, for processing the data

  - *Views* **–** here will be created the front-end of the application using RazorPages

Below is represented a screenshot from Visual Studio, where can be seen the project structure as described above:

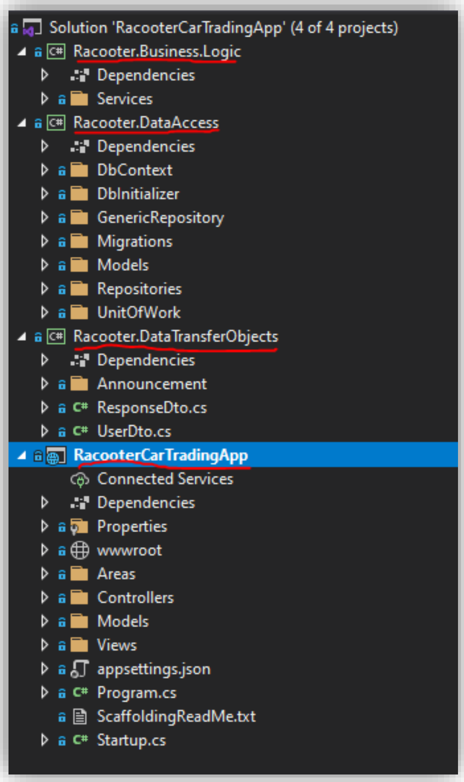


Figure 15. Project structure

So, a main flow of the application will be presented in the following schema, as an example is taken the action to ADD announcement. The schema will describe visually all the layers, processes and files containing methods that will complete the request of adding an announcement:

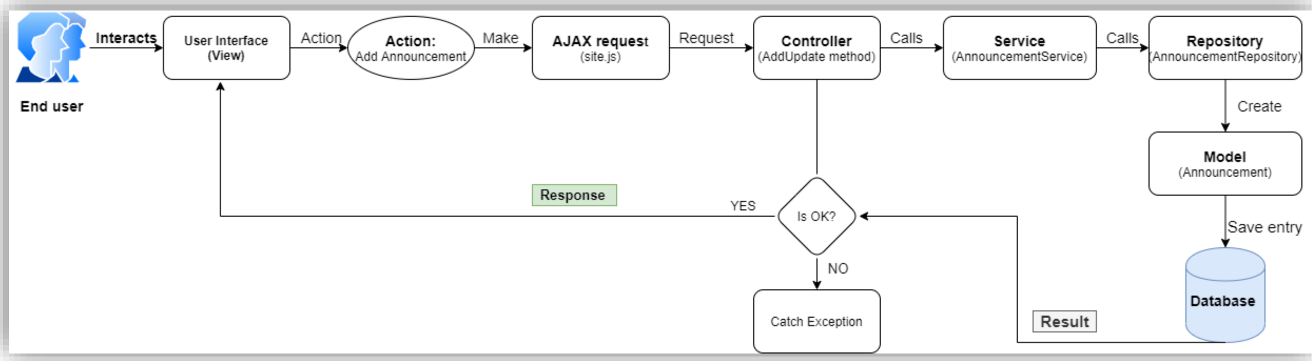


Figure 16. Application features Flow

The end user is interacting with the UI and can make different actions, we will take the example with the adding of a new announcement. After the user clicks the front-end element usually a button, then is called a javascript function inside site.js file where a request will be made to the controller endpoint using all the parameters and values captured form the user input.

The controller will take the request and call that particular method specified in the path, and will call the correspondent method from the service. After that, the service will call the repository where the base method is implemented, in our case the repository is called AnnouncementRepository and the method is AddUpdate.

Then the repository is creating a new instance of the referenced model, filling with all the parameters and values from the user, and it will try to save the entry in the database using the application DbContext.

After the request is done a result will come back to the controller that will analyze the result and if is ok then will send back the response in the view to the end user, otherwise it can throw some exceptions.

4.2 Graphical User Interface implementation

In this section will be presented all the front-end part of the application including screenshots, fron-end code, UI pages and other important aspects that are relevant for the GUI implementation like: field validation, elements layout, responsiveness and so on.[RB20]

- **Newsletter page**

This is the first page when the application starts and it contains a news list. There is the possibility to create a new one by completing the assigned fields: Title, Sub-title and content that are of type text. An icon with a trashcan is placed on each news that will represent the delete news button:

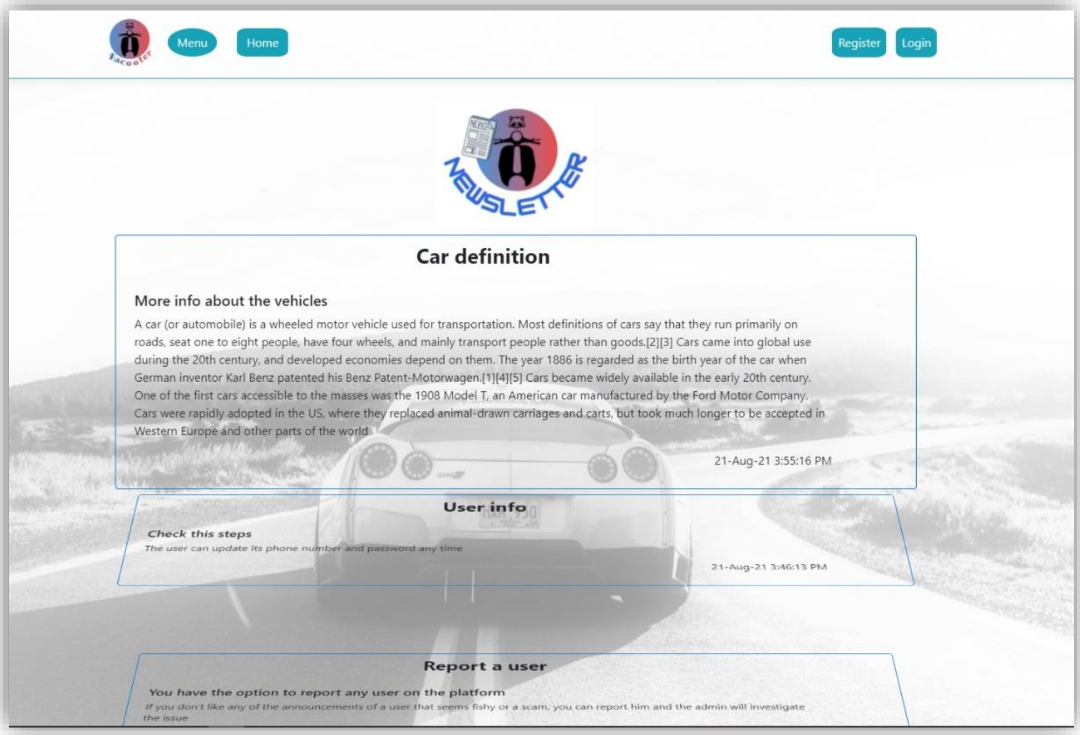


Figure 17. Newsletter page simple

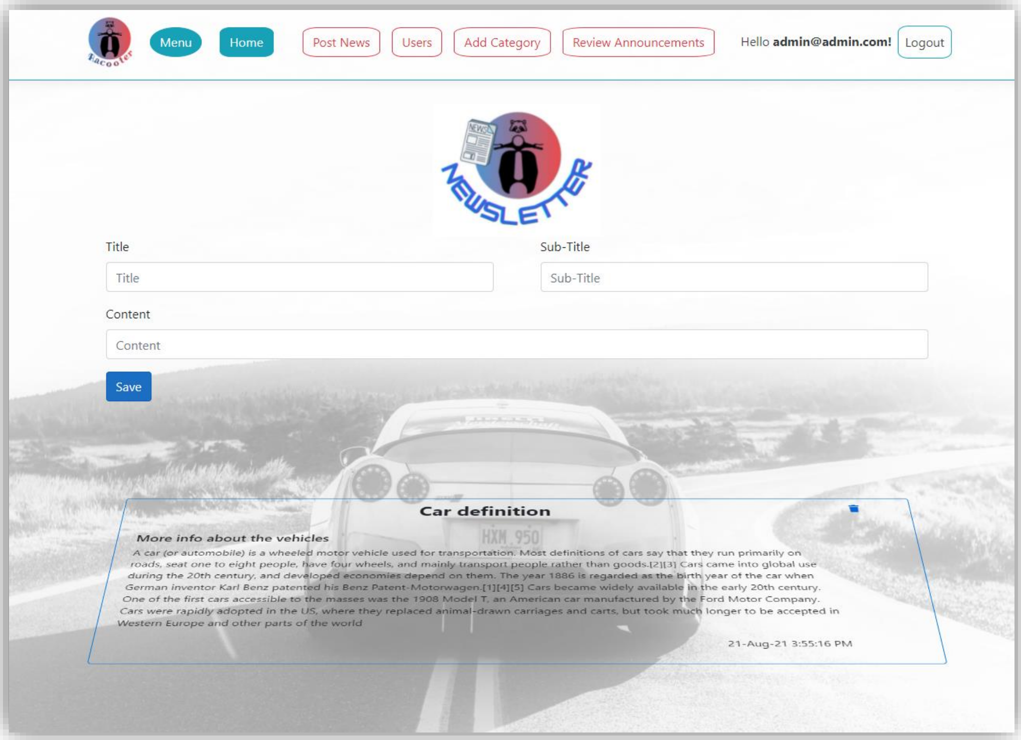


Figure 18. Newsletter page add news

Here are seen more elements because now we are logged in as admin and in the toolbar are placed shortcut buttons to the essential features of the current logged in user, in our case is the admin.

The user can hover the mouse over a news and will rotate to become perpendicular with the user field of view, this is made using css placed in the *NewsPosts.cshtml* representing the razor page for the news. A razor page is allowing integrating some C# code in the HTML view that will generate content for browsers dynamically.

Css of the news is placed between style tags and looks like this:

```
<style>
    .row.each-post {
        border: 1px solid #e5d6d6;
        margin-top: 10px;
        margin-bottom: 10px;
        padding: 10px;
        transform: perspective(2000px) translate3d(0px, -66px, 198px) rotateX(42deg) scale3d(0.86, 0.75, 1) translateY(50px);
        border-radius: 5px;
        will-change: transform;
        transition: 0.4s ease-in-out transform;
    }
    .row.each-post:hover {
            transform: scale3d(1, 1, 1);
    }
</style>
```

Figure 19. CSS of News page

Adding announcement form, that will check if the current logged user is admin or moderator in order to show the adding form and the delete button:

```
<div class="col-md-12">
    @if (User.IsInRole("Admin") || User.IsInRole("Moderator"))
    {
        <div class="row mb-5" id="NewsPostsFormContainer">
            <div class="form-group col-md-6">
                <label class="col-md-12">Title</label>
                <div class="col-md-12">
                    <input type="text" id="txtPostTitle" class="form-control" placeholder="Title" />
                </div>
            </div>

            <div class="form-group col-md-6">
                <label class="col-md-12">Sub-Title</label>
                <div class="col-md-12">
                    <input type="text" id="txtPostSubTitle" class="form-control" placeholder="Sub-Title" />
                </div>
            </div>

            <div class="form-group col-md-12">
                <label class="col-md-12">Content</label>
                <div class="col-md-12">
                    <input type="text" id="txtPostContent" class="form-control" placeholder="Content" />
                </div>
            </div>

            <div class="col-md-12 mb-5">
                <div class="col-md-12">
                    <a class="btn btn-primary text-white" onclick="App.SaveNewsPost()">Save</a>
                </div>
            </div>


        </div>
    }
```

Figure 20. Add news form

And the HTML for displaying all the news can be found below, it does not have any restrictions because can be accessed even by the Guest role. The data is loaded form the *NewsPost* model and displayed the correspondent items after iterating through the model:

```
<div class="col-md-12">

    @foreach (var item in Model)
    {
        <div class="row each-post" style="border-color: #0275d8">
            <div class="col-md-11">
                <h3 style="text-align-last: center">
                    <strong>@item.Title</strong>
                </h3><br />
                <h5>
                    @item.SubTitle
                </h5>
                <p>
                    @item.Content
                </p>
                <p style="text-align-last: right">
                    @item.CreatedDate
                </p>
            </div>
            <div class="col-md-1">
                @if (User.IsInRole("Admin") || User.IsInRole("Moderator"))
                {
                    <a href="/Announcements/DeleteNewsPost?id=@item.Id">
                        <i class="fa fa-trash"></i>
                    </a>
                }
            </div>
        </div>
    }

</div>
```

Figure 21. Display news elements

- **Home page**

This is the page where all the announcements are displayed in bootstrap cardsand also are present the filter elements. This page is composed from two views:

_Home.cshtml – where is placed the recommended announcements section and the normal announcements in another section, divided by bootstrap alert. Recommended announcements will be displayed using a yellow theme and the normal announcements are displayed in a green theme to be distinctive.

Home.cshtml – here is the view where the filter is present, that consist in having some option boxes, input fields and dropdowns where users will input their preferences in order to filter the announcements list.

We will start looking at the second view first and analyze how the filters are implemented from a front-end point of view. The filters on the page are looking like this:

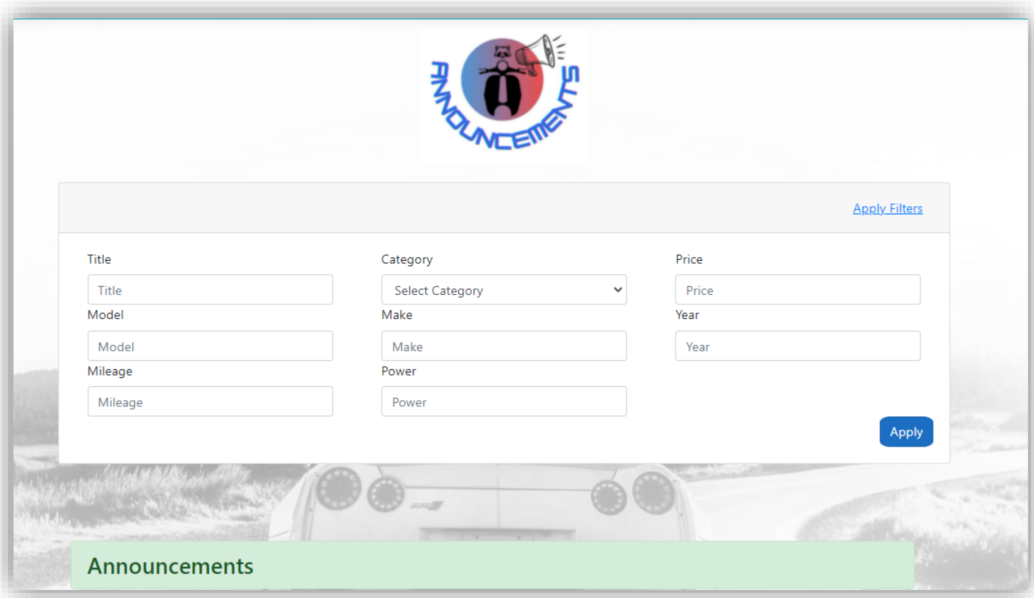


Figure 22. Filter elements

If the user presses the „Apply Filters" text, the filter element will retract back, collapsing the information, below we will see soon the announcements layout and integration present in the first view. For now let's take a look at the front-end code:

```
<div class="container">
    <img src="~/pictures/anouncement.jpg" style="height: 170px; display: block; margin-left: auto;margin-right: auto;" /><br />
    <div class="row mb-5">
        <div id="accordion" style="width:100%;">
            <div class="card">
                <div class="card-header" id="headingTwo">
                    <h5 class="mb-0 text-right">
                        <button class="btn btn-link collapsed" data-toggle="collapse" data-target="#FiltersContainerCollapse" aria-expanded="false" aria-controls="colla
                            Apply Filters
                        </button>
                    </h5>
                </div>
                <div id="FiltersContainerCollapse" class="collapse" aria-labelledby="headingTwo" data-parent="#accordion">
                    <div class="card-body">
                        <div class="row">
                            <div class="col-md-4">
                                <label class="col-md-12">Title</label>
                                <div class="col-md-12">
                                    <input type="text" id="ftrTitle" class="form-control" placeholder="Title" />
                                </div>
                            </div>
                            <div class="col-md-4">
                                <label class="col-md-12">Category</label>
                                <div class="col-md-12">
                                    <select id="ftrCategory" class="form-control">
                                        <option value="">Select Category</option>
                                        @foreach (var item in ViewBag.Categories)
                                        {
                                            <option value="@item.Id">@item.Name</option>
                                        }
                                    </select>
                                </div>
                            </div>

                            <div class="col-md-4">
                                <label class="col-md-12">Price</label>
                                <div class="col-md-12">
                                    <input type="number" id="ftrPrice" class="form-control" placeholder="Price" />
                                </div>
                            </div>

                            <div class="col-md-4">
                                <label class="col-md-12">Model</label>
                                <div class="col-md-12">
                                    <input type="text" id="ftrModel" class="form-control" placeholder="Model" />
                                </div>
```

Figure 23. Filter HTML 1

This is a part of *Home.cshtml* view where every input field element of the filter has an unique id, that will be used to retrieve the value from that particular element using javascript inside *site.js* file.

```
<div class="row" id="AnnouncementsContainer">

    <div class="col-md-12">
        <h3 class="text-center alert alert-warning">Please wait while loading data!!!</h3>
    </div>
</div>
```

Figure 24. Filter HTML 2

In figure 24 we can observe that the announcements after the filtering will appear dyinamic using an AJAX call to the controller api that will fill the view with filtered announcements.

In the following screenshot I placed the method on javascript that is retrieving all the values from the boxes and dropdown, sending as data to the *_Home* method inse *Announcements* controller.

If we get a positive result then will be appended the announcements in the *AnnoucementsContainer* div.

```
let handleLoadHomePageAnnouncements = ($that) => {

    let loaderHtml = '<div class="col-md-12"><h3 class="text-center alert alert-warning"> Please wait while loading data!!!</h3></div>';
    $('#AnnouncementsContainer').html(loaderHtml);
    var $parentDiv = $($that).parents('#FiltersContainerCollapse');

    var filterObj = {
        Title: $('#ftrTitle').val(),
        Category: $('#ftrCategory').val(),
        Price: $('#ftrPrice').val(),
        Model: $('#ftrModel').val(),
        Make: $('#ftrMake').val(),
        Year: $('#ftrYear').val(),
        Mileage: $('#ftrMileage').val(),
        Power: $('#ftrPower').val(),
        PageNumber: 1
    };

    $.ajax({
        type: "Get",
        url: "/Announcements/_Home",
        data: filterObj,
        success: function (response) {
            $('#AnnouncementsContainer').html(response);
        }
    })
}
```

Figure 25. Home page js

The other view that is forming the pages *_Home.cshtml* where all the announcements are displayed in two sections, one is related to the normal announcements and the other one is representing the recommended announcements for the logged in user.

An announcement card contains information about the title, price, created date, seller name, category, views count, a thumbnail of the first attached image and a button that will allow the buyer to contact the owner of a particular announcement.

```
    <div class="col-md-12 mb-5 mt-5">
        <h3 class="alert alert-success">
            Announcements
        </h3>
    </div>
    @foreach (var item in Model.Where(x => x.FilterColumn == null || x.FilterColumn == Char.MinValue).ToList())
    {
        <div class="col-md-3 mb-5">
            <div class="card" style="width: 100%;">
                <img class="card-img-top" style="height: 140px" src="@( item.ImagesPath.Count()==0?"/pictures/no-image-icon-23485.png" :item.ImagesPath.FirstOrDefault())" alt="Card image
                <div class="card-body" style="background-color: whitesmoke">
                    <h5 class="card-title">
                        <a href="/Announcements/Announcement?Id=@item.AnnouncementId">@item.Title</a>
                    </h5>
                    <p class="card-text"><strong>Price: </strong>@item.Price &euro;</p>
                    <p class="card-text"><strong>Date: </strong>@item.CreatedDate</p>
                    <p class="card-text"><strong>Seller: </strong>@item.UserName</p>
                    <p class="card-text"><strong>Category: </strong>@item.CategoryString</p>
                    <p class="fa fa-eye" style="padding-left: 40%"><strong>  @item.Views</strong></p><br />
                    @if (item.CreatedBy != ViewBag.CurrentUserId)
                    {
                        <a href="/Announcements/Messages?id=@item.CreatedBy" class="btn btn-success" style="display: block; margin-left: auto; margin-right: auto;">Contact Seller</a>
                    }
                </div>
            </div>
        </div>
    }

</div>
}



else
{
    <div class="col-md-12">
        <h3 class="text-center alert alert-danger">No data to show!!!</h3>
    </div>
}
```

Figure 26. Announcements HTML

In the front-end, the announcements will look like this:

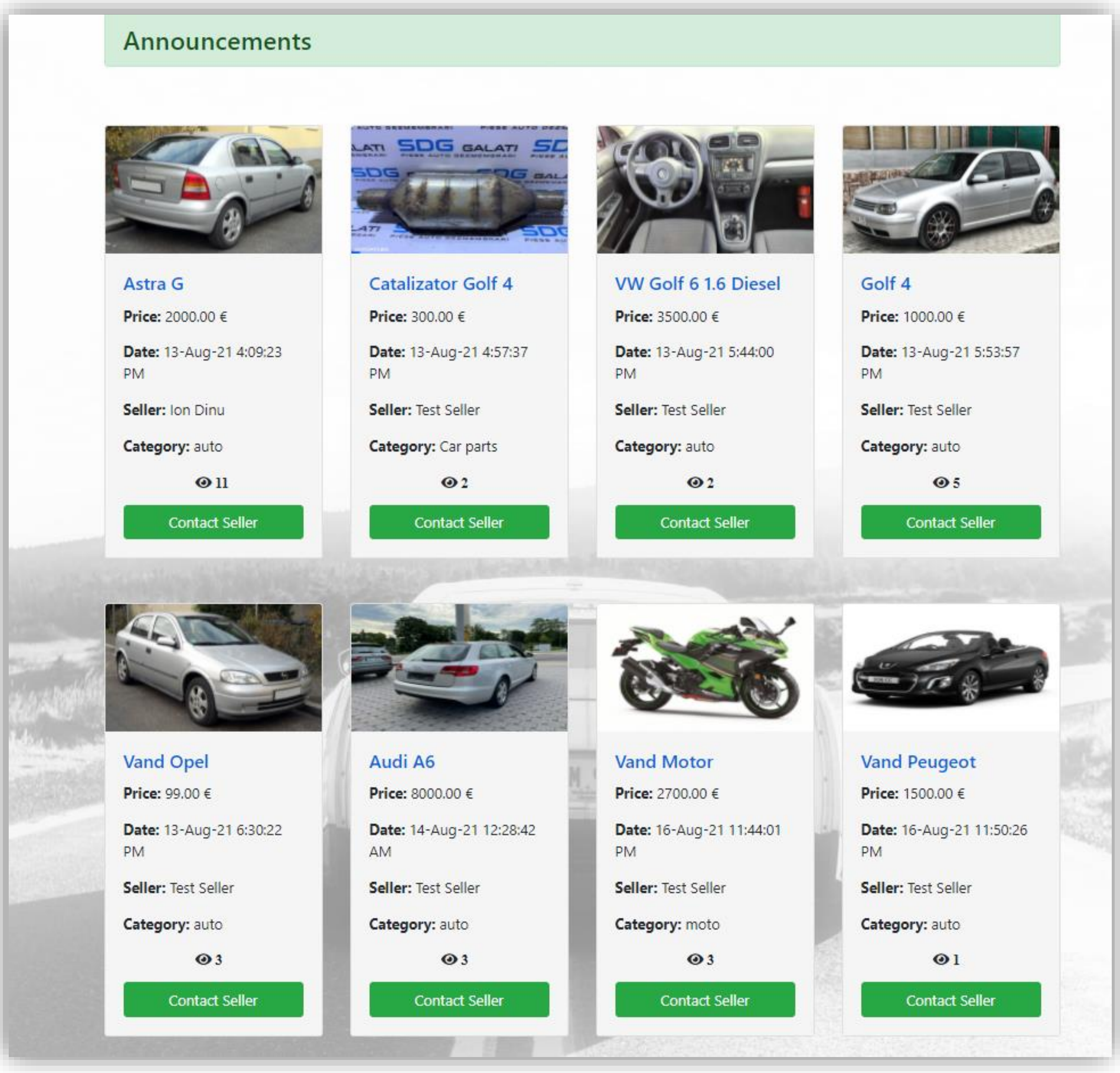


Figure 27. Announcements display

The recommended announcements section will look similar, only the alert will be yellow with the text: „Recommended for You" and the contact seller button will yellow as well to be more distinctive then the normal announcements and attract the user interest.

- **Announcement details page**

This represents the page of the announcement details, so if the user clicks an announcement card from the previous described page, then he/she will be redirected to a detailed page of that particular announcement where the buyer can see more information about the car.

On this page there is the possibility to see more contact information about the seller, but also a report seller button and a contact seller one that will contact directly the seller of that announcement.

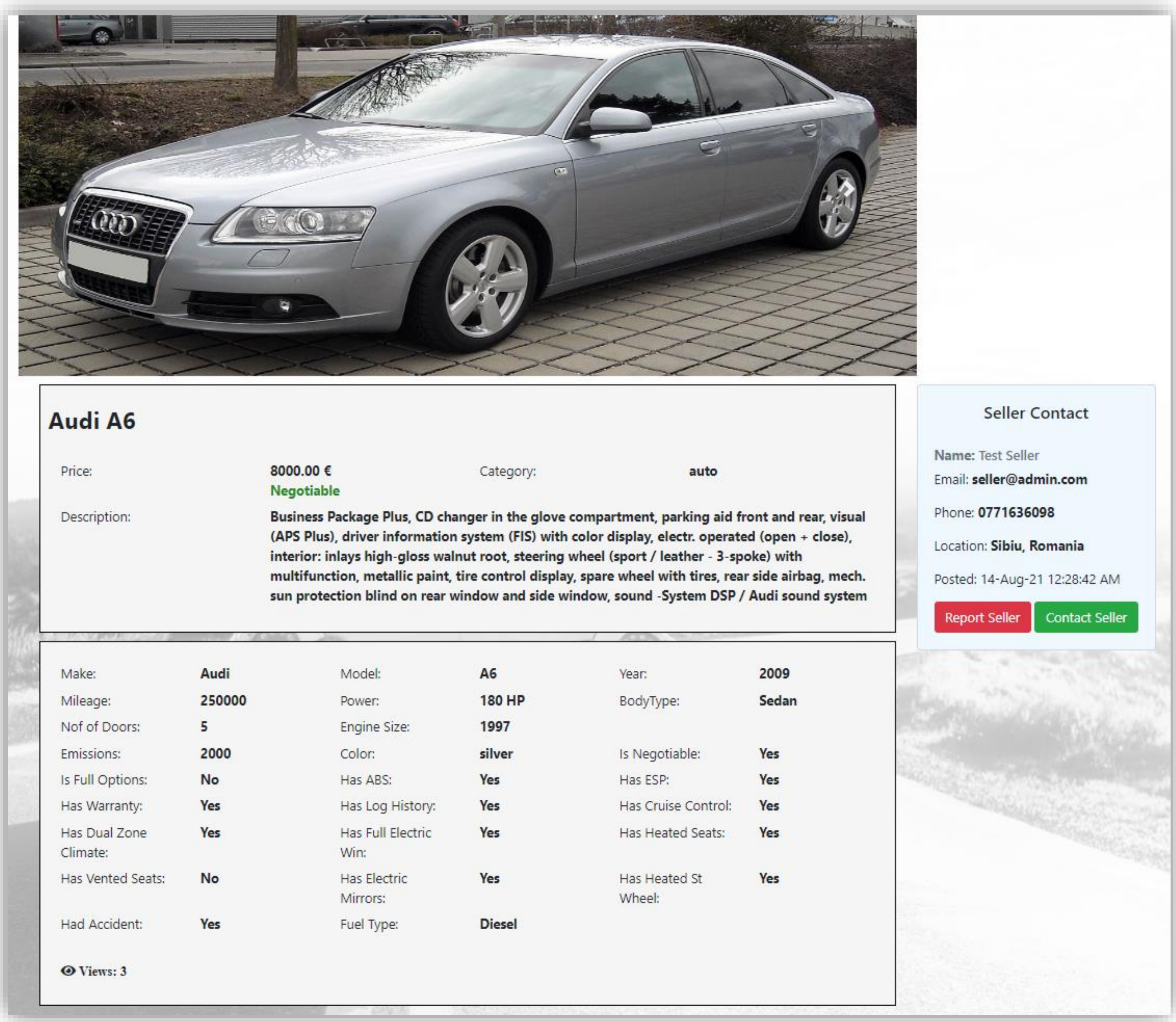


Figure 28. Announcement details page

For example we get this announcement and we can observe the ammount of information that is provided with an easy to read and intuitive layout. The price can be „Negotiable” or „Fixed”, and is displayed in green if is negotiable and in red if is fixed to attratc buyer attention on important details and not bother the seller with irrelevant questions.

This page is implemented in the *Announcement.cshtml* view, and I used a carousel as an image container, so the pictures are automatically sliding one by one in a loop, or can be manually changed by pressing the arrows from the left and right of the image, that are hard to see on the screenshot.

The announcement details are displayed using bootstrap cards, and bootstrap grid system using *row*, *col-md* class types. Below is a part of the front-end code for displaying the announcement info:

```html
<div style="padding: 25px;margin: 10px; border: 1px solid #000000; background-color: #F5F5F5;">
    <div class="row">
        <h3>
            <strong>@Model.Title</strong>
        </h3>
    </div><br />

    <div class="row">
        <label class="col-md-3">
            Price:
        </label>

        <label class="col-md-3">
            <strong>@Model.Price &euro;</strong><br />
            @if (Model.Specification.IsNegotiable)
            {
                <strong style="color: forestgreen"> Negotiable</strong>
            }
            else
            {
                <strong style="color: red"> Fixed</strong>
            }
        </label>

        <label class="col-md-3">
            Category:
        </label>
        <label class="col-md-3">
            <strong>@Model.CategoryString</strong>
        </label>
    </div>

    <div class="row">
        <label class="col-md-3">
            Description:
        </label>
        <label class="col-md-9">
            <strong>
                @Model.Description
            </strong>
        </label>
    </div>
</div>
<div style="padding: 25px;margin: 10px;border: 1px solid #000000; background-color: #F5F5F5">
    <div class="row">
        <label class="col-md-2">
            Make:
        </label>

        <label class="col-md-2">
            <strong>@Model.Specification.Make</strong>
```

Figure 29. Announcement details HTML 1

And here is the part where are displayed info about the seller:

```html
<div style="position:absolute; left: 100%">
    <div class="card" style="width: 18rem; background-color: #F0F8FF;">
        <div class="card-body">
            <h5 class="card-title" style="text-align: center">Seller Contact</h5><br />
            <h6 class="card-subtitle mb-2 text-muted"><strong>Name:</strong> @Model.UserName</h6>
            <p class="card-text">Email: <strong> @Model.UserEmail</strong></p>
            <p class="card-text">Phone: <strong> @Model.PhoneNumber</strong></p>
            <p class="card-text">Location: <strong> @Model.Location</strong></p>
            <p class="card-text">Posted: @Model.CreatedDate</p>
            @if (Model.CreatedBy != ViewBag.CurrentUserId)
            {
                <a class="btn btn-danger" style="color: white" onclick="App.ReportSeller('@Model.CreatedBy')">Report Seller</a>
                <a class="btn btn-success" href="/Announcements/Messages?id=@Model.CreatedBy">Contact Seller</a>
            }
        </div>
    </div>
</div>
```

Figure 30. Announcement details HTML 2

- **Add Announcement page**

This is the page where is present the adding form of the announcement, where the seller is providing all the required information about the car that he/she is willing to sale.

It contains a lot of user input choices and front end elements like: dropdowns, text boxes, number boxes, date dropdown, and checkbox for the specific options that a car has.

Some information are mandatory to be completed by the user, so additional checks will be made using js, and the user will be informed if other fields must be completed before saving the announcement. There will be three button, one for adding images, saving the announcement and to cancel the process:

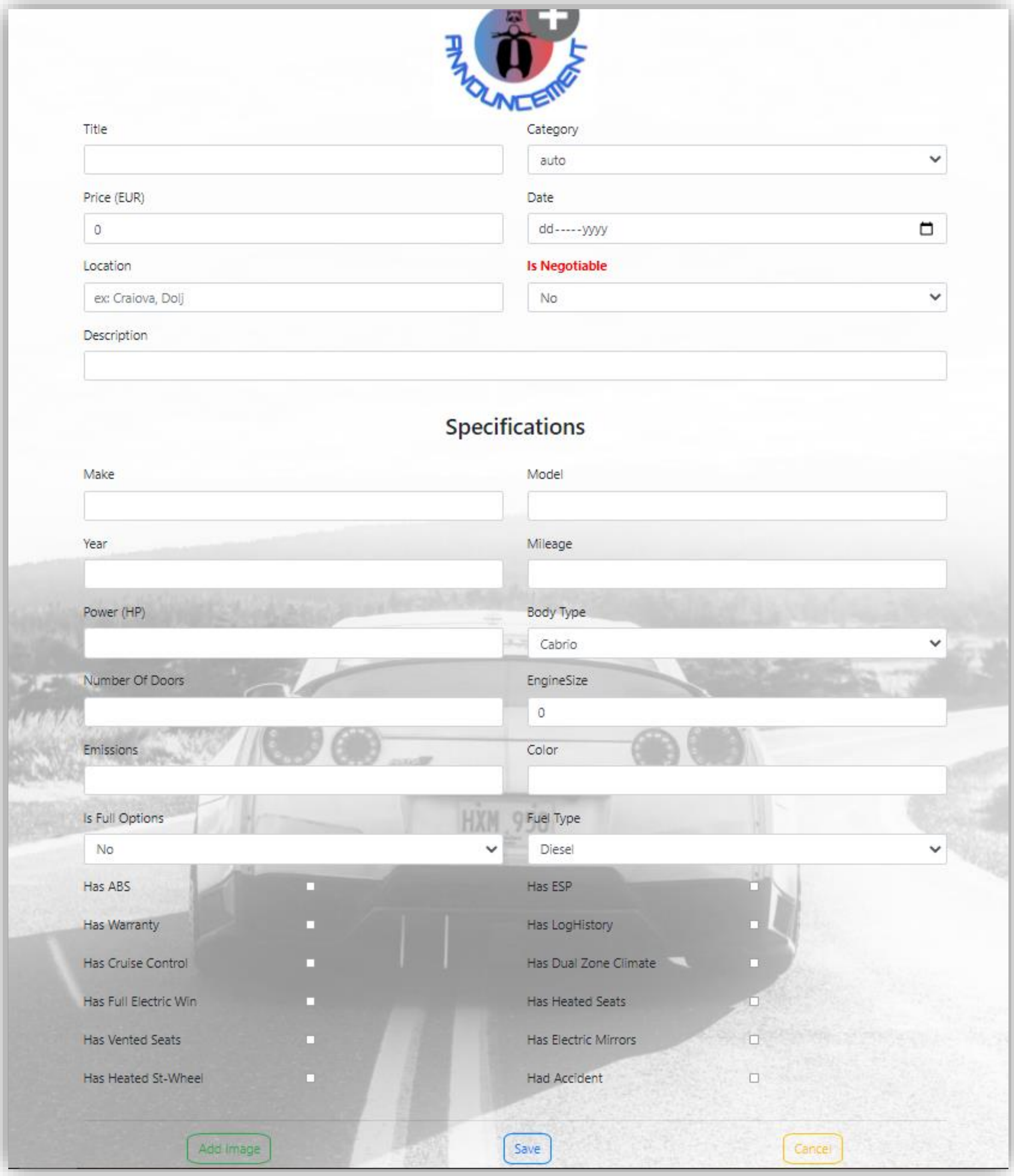


Figure 31. Add Announcement form

The add image button will allow the user to browse in the file explorer and choose an image, if the seller want to add multiple images, he can press again the „Add Image” button that will create another button for choosing an image that will trigger the file explorer window, below is a demonstration:

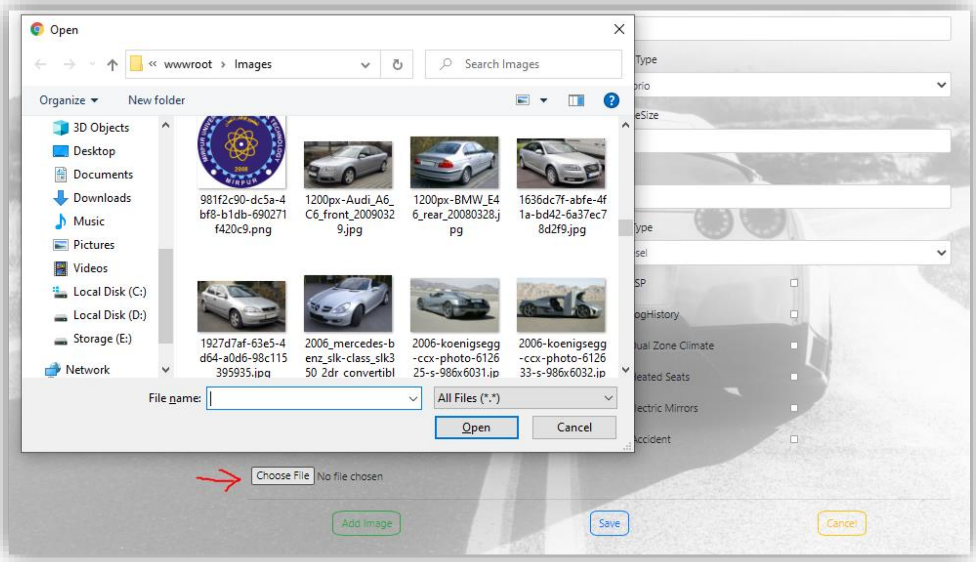


Figure 32. Add image explorer

Below is a section of the front-end code from *AddUpdate.cshtml* view, and it can be observed the way a dropdown is filled and how the data is selected:

```
<div class="row">
    <div class="col-md-6">
        <div class="form-group row">
            <label class="col-md-12">Price (EUR)</label>
            <div class="col-md-12">
                <input type="number" class="form-control" value="@Model.Price" id="Price" name="Price" />
            </div>
        </div>
    </div>

    <div class="col-md-6">
        <div class="form-group row">
            <label class="col-md-12">Date</label>
            <div class="col-md-12">
                <input type="date" class="form-control" value="@Model.Date" id="Date" name="Date" />
            </div>
        </div>
    </div>
</div>

<div class="row">
    <div class="col-md-6">
        <div class="form-group row">
            <label class="col-md-12">Location</label>
            <div class="col-md-12">
                <input type="text" id="Location" name="Location" placeholder="ex: Craiova, Dolj" value="@Model.Location" class="form-control" />
            </div>
        </div>
    </div>
    <div class="col-md-6">
        <div class="form-group row">
            @if (Model.Specification.IsNegotiable)
            {
                <label class="col-md-12" style="color: forestgreen"><strong>Is Negotiable</strong></label>
            }
            else
            {
                <label class="col-md-12" style="color: red"><strong>Is Negotiable</strong></label>
            }
            <div class="col-md-12">
                <select class="form-control" id="IsNegotiable" name="Specification.IsNegotiable">
                    @if (Model.Specification.IsNegotiable == true)
                    {
                        <option value="true" selected>Yes</option>
                        <option value="false">No</option>
                    }
                    else
                    {
                        <option value="true">Yes</option>
                        <option value="false" selected>No</option>
                    }
                </select>
```

Figure 33. AddUpdate.cshtml

The checks on js are looking like this, and the user notifications are created with *toastr.error* function. If the seller does not complete the required boxes and options, then he can't save the announcement.

```
var handleSaveAnnouncement = function () {

    var obj = {
        AnnouncementId: $('#HiddenAnnouncementId').val(),
        Title: $('#Title').val(),
        Category: $('#Category').val(),
        Price: $('#Price').val(),
        Date: $('#Date').val(),
        Description: $('#Description').val(),
        Location: $('#Location').val(),
        PhoneNumber: $('#PhoneNumber').val()
    }
    if (obj.Title == undefined || obj.Title == "") {
        toastr.error('Title is required!');
        return false;
    }
    if (obj.Description == undefined || obj.Description == "") {
        toastr.error('Description is required!');
        return false;
    }

    if (obj.Price == undefined || obj.Price == "") {
        toastr.error('Price is required!');
        return false;
    }
    if (obj.Category == undefined || obj.Category == "" || obj.Category == "-1") {
        toastr.error('Category is required!');
        return false;
    }
    if (obj.Location == undefined || obj.Location == "") {
        toastr.error('Location is required!');
        return false;
    }
```

Figure 34. Add announcement form checks

And when the user presses add image button, it will dynamically create another „Choose file" button that on press will open the file explorer where the user upload images for the announcement, below is the javascript code of the action button „Add Image":

```
let handleMakeImageUploader = () => {
    let html = '<div class="col-md-3">';
    html += '<input type="file" class="each-image-uploaded" style="margin:5px;" />';
    html += '</div>';
    $('#ImagesContainerDiv').append(html)
}
```

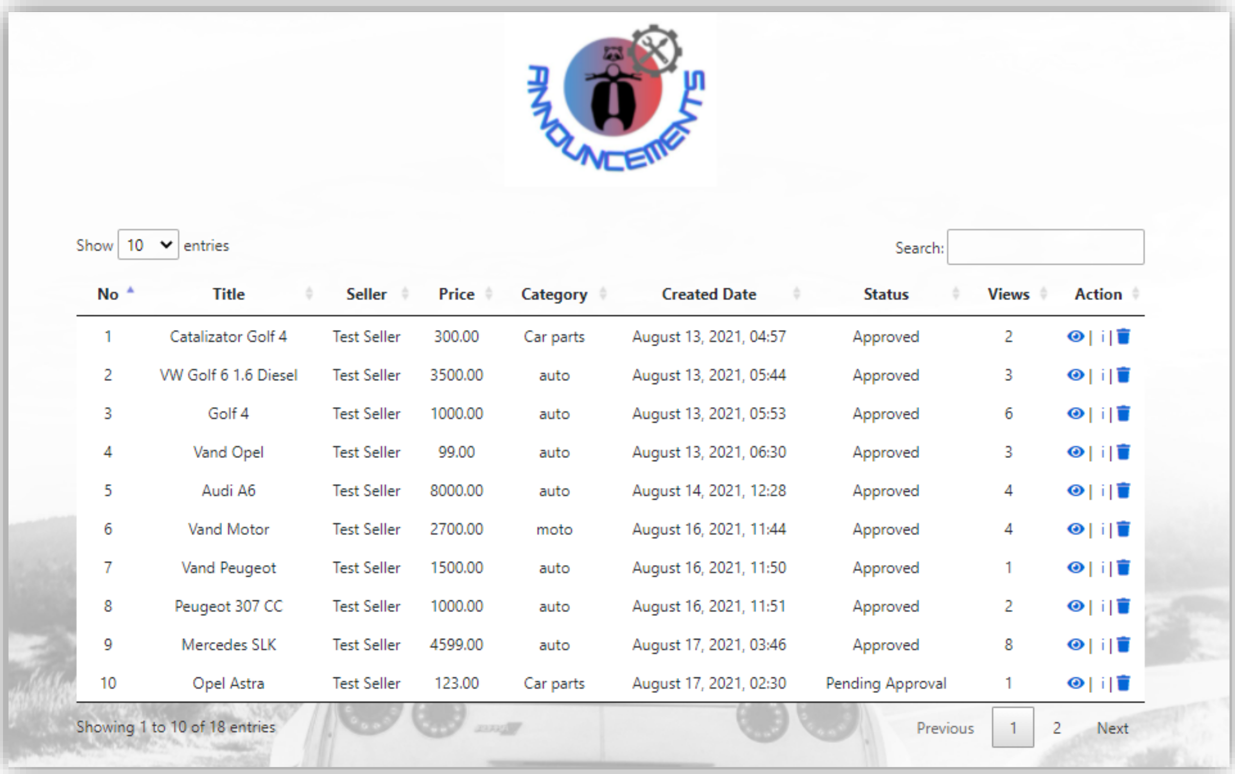
Figure 35. Upload image file explorer dynamically

- **Personal announcements page**

This is the page where the user can see, and track their posted announcement. Is the announcement panel with some actions like preview, update and delete a particular announcement.

Here are present usefull information like the announcement title, price, category, created date, status that can be „Approved” or „Pending Approval” and the number of views.

The diplaying is made in a table, also with pagination, search box and the number of entries the user wants to display in the table at once:



No	Title	Seller	Price	Category	Created Date	Status	Views	Action
1	Catalizator Golf 4	Test Seller	300.00	Car parts	August 13, 2021, 04:57	Approved	2	| i |
2	VW Golf 6 1.6 Diesel	Test Seller	3500.00	auto	August 13, 2021, 05:44	Approved	3	| i |
3	Golf 4	Test Seller	1000.00	auto	August 13, 2021, 05:53	Approved	6	| i |
4	Vand Opel	Test Seller	99.00	auto	August 13, 2021, 06:30	Approved	3	| i |
5	Audi A6	Test Seller	8000.00	auto	August 14, 2021, 12:28	Approved	4	| i |
6	Vand Motor	Test Seller	2700.00	moto	August 16, 2021, 11:44	Approved	4	| i |
7	Vand Peugeot	Test Seller	1500.00	auto	August 16, 2021, 11:50	Approved	1	| i |
8	Peugeot 307 CC	Test Seller	1000.00	auto	August 16, 2021, 11:51	Approved	2	| i |
9	Mercedes SLK	Test Seller	4599.00	auto	August 17, 2021, 03:46	Approved	8	| i |
10	Opel Astra	Test Seller	123.00	Car parts	August 17, 2021, 02:30	Pending Approval	1	| i |

Figure 36. Personal announcement page

The user can sort the table entries on the columns just by clicking that column arrows and will sort the results on that specific column ascending or descending. The sort , search and pagination feature is implemented using a library: DataTables CDN[34] (Content Delivery Network) that is a javascript library helping with improving the tables visualization and features.

I needed to make a reference in project to this library, between script tags and also to add some lines of js in order to apply it on my table.

Below is a part of front-end code of the Personal announcement page:

```
@model IEnumerable<Racooter.DataTransferObjects.Announcement.AnnouncementDto>
@{
    int count = 1;
}
<div class="col-md-12 text-center">
    <img src="~/pictures/announceGear.jpg" style="height: 180px; display: block; margin-left: auto;margin-right: auto;" /><br />

    <div class="row" style="margin-top:20px;">
        <table class="table table-borderless" id="AnnouncementsGrid">
            <thead>
                <tr>
                    <th>No</th>
                    <th>Title</th>
                    <th>Seller</th>
                    <th>Price</th>
                    <th>Category</th>
                    <th>Created Date</th>
                    <th>Status</th>
                    <th>Views</th>
                    <th>Action</th>
                </tr>
            </thead>
            <tbody>
                @foreach (var item in Model)
                {
                    <tr>
                        <td>@count</td>
                        <td>@item.Title</td>
                        <td>@item.UserName</td>
                        <td>@item.Price</td>
                        <td>@item.CategoryString</td>
                        <td>@item.CreatedDate.ToString("MMMM dd, yyyy, hh:mm")</td>
                        <td>@(item.IsApprovedByAdmin ? "Approved" : "Pending Approval")</td>
                        <td>@item.Views</td>
                        <td>
                            <a href="/Announcements/Announcement?Id=@item.AnnouncementId">
                                <i class="fa fa-eye"></i>
                            </a>
                            |
                            <a href="/Announcements/AddUpdate?Id=@item.AnnouncementId">
                                <i class="fa fa-pencil-square-o"></i>
                                i
                            </a>
                                <span>
                                    |
                                    <a href="/Announcements/Delete?id=@item.AnnouncementId">
                                        <i class="fa fa-trash"></i>
                                    </a>
                                </span>

                        </td>
                    </tr>
                    count++;
                }
            </tbody>
        </table>
    </div>
```

Figure 37. Index.cshtml

The library is called in this way, where *AnnouncementsGrid* is the div id where the table is present:

```
<script src="~/lib/jquery/dist/jquery.min.js"></script>
<script>
    $(document).ready(function () {
        $('#AnnouncementsGrid').DataTable();
    })
</script>
```

Figure 38. DataTable library integration

- **Inbox page**

This is the page where the users can access the inbox and chat with eachother inside the application using the online chat system. The chat is updating frequently in order to receive the new messages without refreshing the whole page everytime.

The front-end page should have the inbox window divided in three sides, one is represented by all the conversation list where the user will choose a user to chat with, the second section will be in the right and is representing the actual messages between users that are containing a message text and also a timestamp when every message was sent. The 3rd section is represented by the chat box where the user can write messages and send them to a recipient.

This is the visualization and layout[35] of the Inbox page:

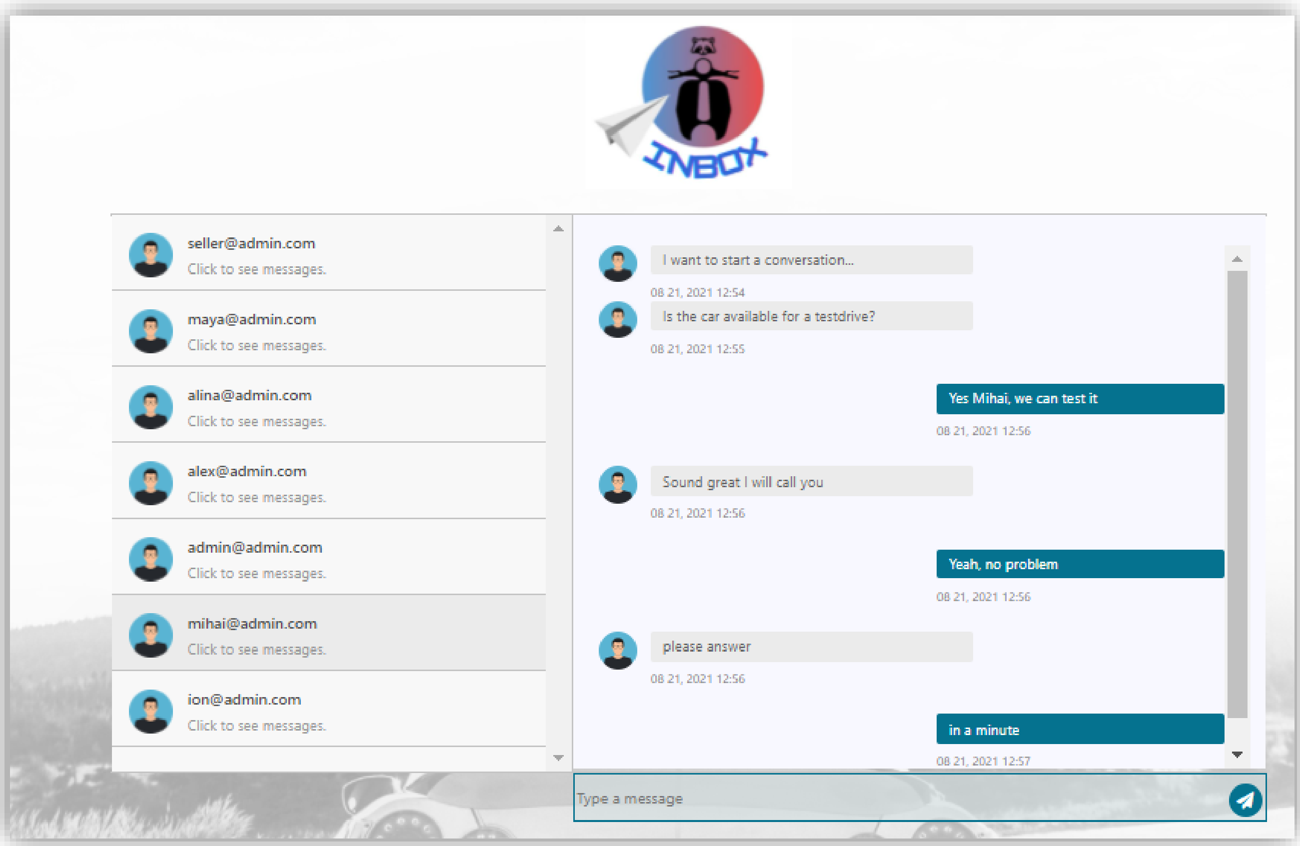


Figure 39. Inbox page

This page consists in two views:

Messages.cshtml – where is present the visualization of the users in the left section that are representing the conversation list, and also it contains the chat box and the send button

UserMessages.cshtml – where is representing a template for each message that should contain the MessageText, CreatedDate for the incoming and outgoing messages.

The razor page view for *Message.cshtml* is showned below:

```
<input type="hidden" value="@ViewBag.ReceipentId" id="hiddenReceipentId" />
<input type="hidden" value="" id="selectedUserId"/>
<img src="~/pictures/inbox.jpg" style="height: 170px; display: block; margin-left: auto;margin-right: auto;" /><br />
<div class="row">
    <div class="col-12">
        <div class="messaging" style="height:350px !important;">
            <div class="inbox_msg">
                <div class="inbox_people">
                    <div class="inbox_chat">
                        @if (ViewBag.MessageUsers != null)
                        {
                            foreach (var item in ViewBag.MessageUsers as List<KeyValuePair<string, string>>)
                            {
                                <div class="chat_list">
                                    <div class="chat_people" data-id="@item.Key" onclick="App.LoadUserMessages('@item.Key')">
                                        <div class="chat_img"> <img src="https://ptetutorials.com/images/user-profile.png" alt="sunil"> </div>
                                        <div class="chat_ib">
                                            <h5>@item.Value <span class="chat_date"></span></h5>
                                            <p>
                                                Click to see messages.
                                            </p>
                                        </div>
                                    </div>
                                </div>}
                        }
                    </div>
                </div>
                <div class="mesgs" id="UserMessagesContainer" style="background: #F8F8FF; border-color: black">

                </div>
            </div>

        </div>
    </div>
</div>
<div class="row">
    <div class="col-12">
        <div class="type_msg">
            <div class="input_msg_write">
                <input type="text" id="txtBoxMessage" class="write_msg" placeholder="Type a message" />
                <button class="msg_send_btn" id="btnSaveMessage" onclick="App.SaveMessage()" type="button"><i class="fa fa-paper-plane" aria-hidden="true"></i></button>
            </div>
        </div>
    </div>
</div>
<script src="~/lib/jquery/dist/jquery.min.js"></script>
<script>
    $(document).ready(function () {
        if ($('.chat_list').length > 0) {
            $('.chat_list:first').find('.chat_people').trigger('click');
            setInterval(reloadMessages, 2000);

            $("#txtBoxMessage").keyup(function (event) {
                if (event.keyCode === 13) {
                    $("#btnSaveMessage").click();
                }
            });
        }
    });
</script>
```

Figure 40. Message.cshtml

As we can observe, the list with conversations are placed inside the *chat_list* div where are all the users that the current logged in user has started a conversation with them. If one of the conversation is pressed then the method *LoadUserMessages()* will be triggered from *site.js* and will load the actual messages sent in that conversation. Below is attached a screenshot with the method on js:

```
let handleLoadUserMessages = (userId) => {

    $('.chat_list').removeClass('active_chat');
    $('div[data-id="' + userId + '"]').parents('.chat_list').addClass('active_chat');
    $('#selectedUserId').val(userId);
    $.ajax({
        type: "GET",
        url: "/Announcements/UserMessages",
        data: {
            userId: userId
        },
        success: function (response) {
            $('#UserMessagesContainer').html(response);
        }
    })
}
```

Figure 41. LoadUserMessages method

The messages will appear dynamically after the request to the controller is made, and the response will be displayed in the div where the id is *UserMessagesContainer*. The conversation that is clicked will have the class name *active_chat*, to know which is the current one.

The *UserMessages.cshtml* represents the messages between a sender and receiver, the front-end code is attached below:

```
@model IEnumerable<Racooter.DataTransferObjects.Announcement.MessageDto>

<div class="msg_history">
    @foreach (var item in Model)
    {
        if (item.CreatedBy == ViewBag.CurrentUserId)
        {
<div class="outgoing_msg">
    <div class="sent_msg">
        <p>
            @item.MessageText
        </p>
        <span class="time_date"> @item.CreatedDate.ToString("MM dd, yyyy hh:mm")</span>
    </div>
</div> }
else
{
<div class="incoming_msg">
    <div class="incoming_msg_img"> <img src="https://ptetutorials.com/images/user-profile.png" alt="sunil"> </div>
    <div class="received_msg">
        <div class="received_withd_msg">
            <p>
                @item.MessageText
            </p>
            <span class="time_date"> @item.CreatedDate.ToString("MM dd, yyyy hh:mm")</span>
        </div>
    </div>
</div>}
}

</div>
```

Figure 42. UserMessage.cshtml

We can observe the layout and the order in which are displayed, also are marked with *outgoing_msg* the messages that are sent by the current user and with *incoming_msg* are marking the messages of the recipient, in order to apply different styling to be displayed as a chat visualization.

The chat is refreshing by automatically calling the *LoadUserMessages(userId)* in a document ready function at an interval of 2000 ms. The userId is taken from the ViewBag that is filled from the backend with the user id of the logged user.

Below is the code for *reloadMessages()* and its call:

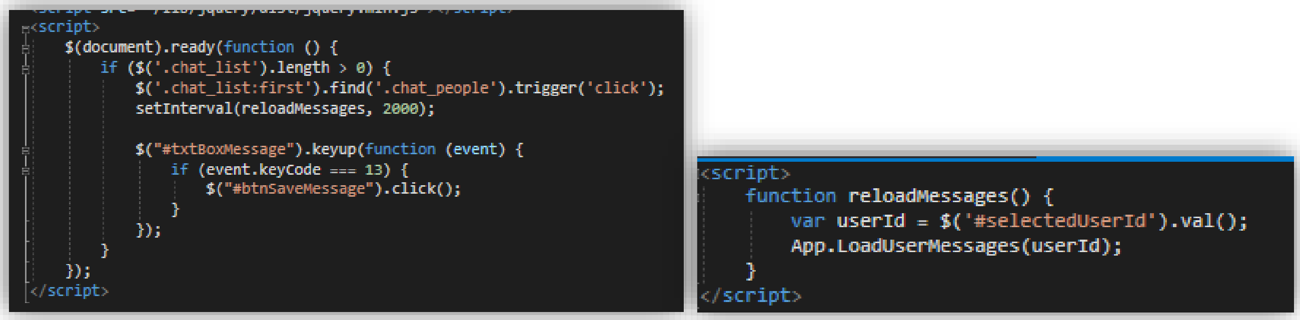


Figure 43. ReloadMessages method and call

- **Category page**

This is a page that the admin can access where he can add a new category in the application. It consists in two views:

Categories.cshtml – where is placed a textbox where the admin can save a new category name, the method called on javascript is *SaveCategory()*, and another method is *LoadCategory()* that will make a call on the controller and display the response in a div dyncamically, similar to displaying the messages in a conversation.

_Categories.cshtml – where the table with the visualization of all categories saved on the db and also an action to delete a particular category.

Below will be attached the user interface on the categories, where the both views are used:

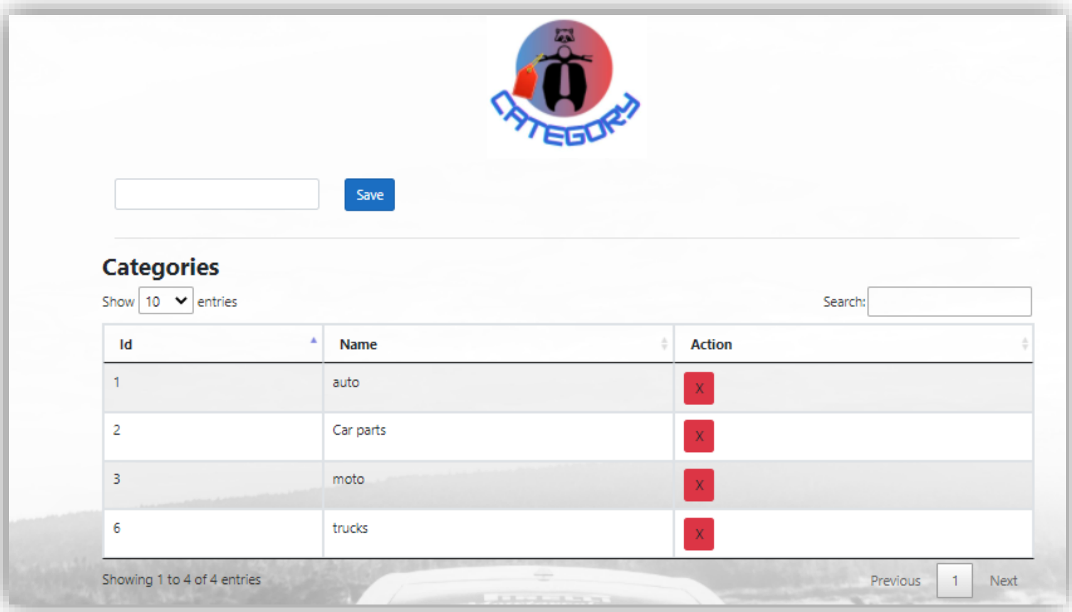


Figure 44. Categories views

The admin can filter the results and delete a category by pressing the red button.

The code for *Categories.cshtml* is attached below:

```
<div class="col-md-12">
    <img src="~/pictures/cat.jpg" style="height: 170px; display: block; margin-left: auto;margin-right: auto;" /><br />
    <div class="row" style="margin: 0 auto">
        <div class="form-group row">
            <div class="col-md-10">
                <input type="text" data-id="0" id="txtCategoryName" class="form-control" />
            </div>
            <div class="col-md-2">
                <a class="btn btn-primary" style="color:white" onclick="App.SaveCategory(this)">Save</a>
            </div>
        </div>
    </div>

    <hr />


    <div class="row" id="CategoriesContainer">

    </div>
</div>
<script src="~/lib/jquery/dist/jquery.min.js"></script>
<script>
    $(document).ready(function () {
        App.LoadCategories();
    });
</script>
```

Figure 45. Categories.cshtml

And the method *LoadCategories()* is present on site.js attached below:

```
let handleLoadCategories = () => {
    $.ajax({
        type: "GET",
        url: "/Announcements/_Categories",
        success: function (response) {
            $('#CategoriesContainer').html(response);
        }
    })
}
```

Figure 46. LoadCategories method js

And the *_Categories.cshtml* with the table and visualization of the search box, column sorting and pagination by calling the *DataTable()* library:

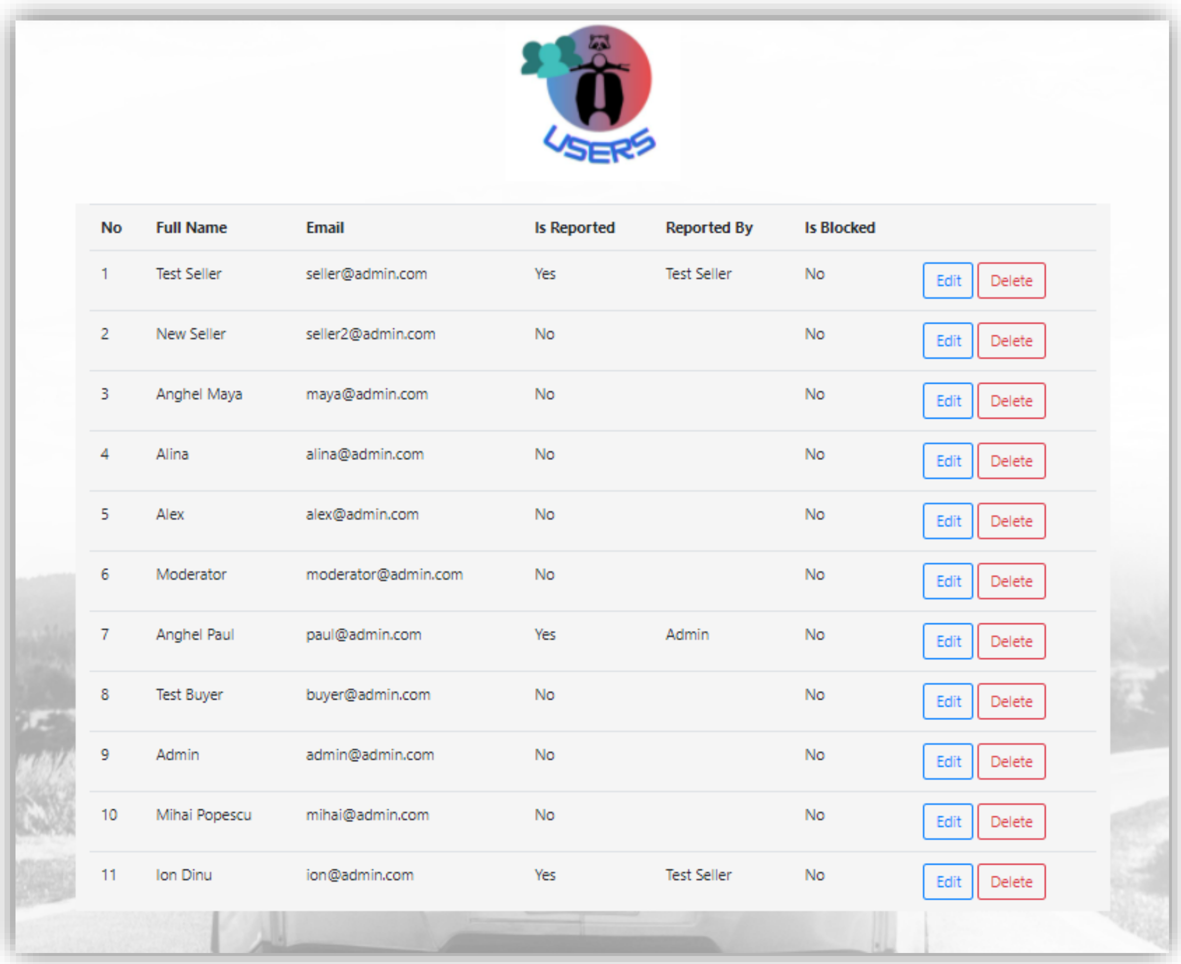
```
@model IEnumerable<Racooter.DataTransferObjects.Announcement.CategoryDto>

<div class="col-md-12">
    <div class="row">
        <h3>
            <strong>Categories</strong>
        </h3>
    </div>
    <div class="row">
        <table class="table table-bordered table-striped" id="CategoriesGrid">
            <thead>
                <tr>
                    <th>Id</th>
                    <th>Name</th>
                    <th>Action</th>
                </tr>
            </thead>
            <tbody>
                @foreach (var item in Model)
                {
                    <tr>
                        <td>@item.Id</td>
                        <td>@item.Name</td>
                        <td>
                            <a class="btn btn-danger" onclick="App.DeleteCategory(@item.Id)">X</a>
                        </td>
                    </tr>
                }
            </tbody>
        </table>
    </div>
</div>

<script>
    $(document).ready(function () {
        $('#CategoriesGrid').DataTable();
    })
</script>
```

Figure 47. _Categories.cshtml

- **User panel UI**

This is the page where the admin can see all the user accounts with information like Full name, email, if is reported, who reported him, if is blocked from posting announcements and other two actions delete and edit.

Here is the visualization of the users panel, accessed by the admin:



No	Full Name	Email	Is Reported	Reported By	Is Blocked	
1	Test Seller	seller@admin.com	Yes	Test Seller	No	Edit Delete
2	New Seller	seller2@admin.com	No		No	Edit Delete
3	Anghel Maya	maya@admin.com	No		No	Edit Delete
4	Alina	alina@admin.com	No		No	Edit Delete
5	Alex	alex@admin.com	No		No	Edit Delete
6	Moderator	moderator@admin.com	No		No	Edit Delete
7	Anghel Paul	paul@admin.com	Yes	Admin	No	Edit Delete
8	Test Buyer	buyer@admin.com	No		No	Edit Delete
9	Admin	admin@admin.com	No		No	Edit Delete
10	Mihai Popescu	mihai@admin.com	No		No	Edit Delete
11	Ion Dinu	ion@admin.com	Yes	Test Seller	No	Edit Delete

Figure 48. Users panel view

The view is implemented using a different controller, and is called *UsersController*, so this view will be present in the folder with the same name as the controller with the name of *Index.cshtml*

The view code will be attached below, is basically a table where the data are loaded from UserDto object:

```
@model IEnumerable<Racooter.DataTransferObjects.UserDto>
@{
    int count = 1;
}
<img src="~/pictures/users.jpg" style="height: 170px; display: block; margin-left: auto;margin-right: auto;" />
<br />
<div class="col-md-12" style="background-color: #F5F5F5">
    <table class="table" id="UsersGrid">
        <thead>
            <tr>
                <th>No</th>
                <th>Full Name</th>
                <th>Email</th>
                <th>Is Reported</th>
                <th>Reported By</th>
                <th>Is Blocked</th>
                <th></th>
            </tr>
        </thead>
        <tbody>
            @foreach (var item in Model)
            {
                <tr>
                    <td>@count</td>
                    <td>@item.FullName</td>
                    <td>@item.Email</td>
                    <td>@(item.IsReportedForBlock==true?"Yes":"No")</td>
                    <td>@(item.ReportedBy)</td>
                    <td>@(item.IsBlockFromPost==true?"Yes":"No")</td>
                    <td>
                        <a href="/Users/UpdateUser?id=@item.id" class="btn btn-outline-primary">Edit</a>

                        <a href="/Users/DeleteUser?id=@item.id" class="btn btn-outline-danger">Delete</a>
                    </td>
                </tr>
                count++;
            }
        </tbody>
    </table>
</div>
```

Figure 49. Users Index.cshtml

If the users will press the button „Edit" then a new page will be opened where the information about that user will be displayed, and also the status if is blocked, or is reported.

The update user page is represented by the view *UpdateUser.cshtml*

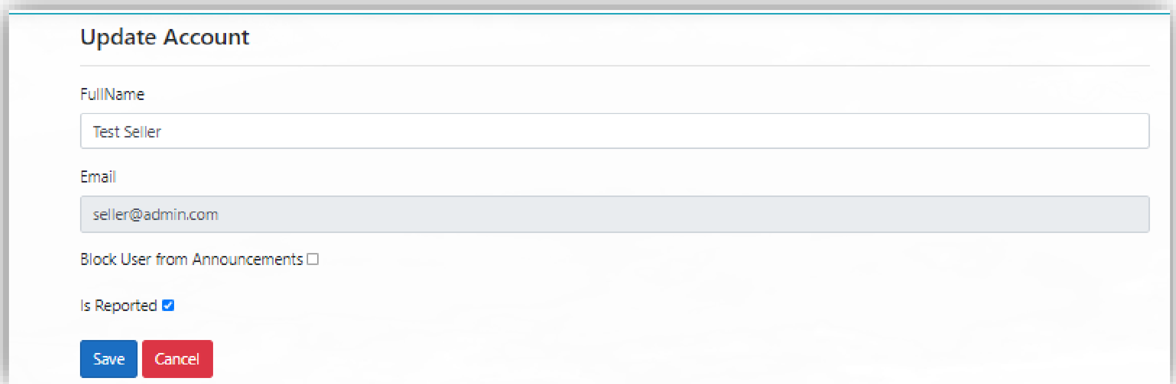


Figure 50. Update Account view

The email field is disabled from html because is better to not modify the email, being an essential credential in the login process.

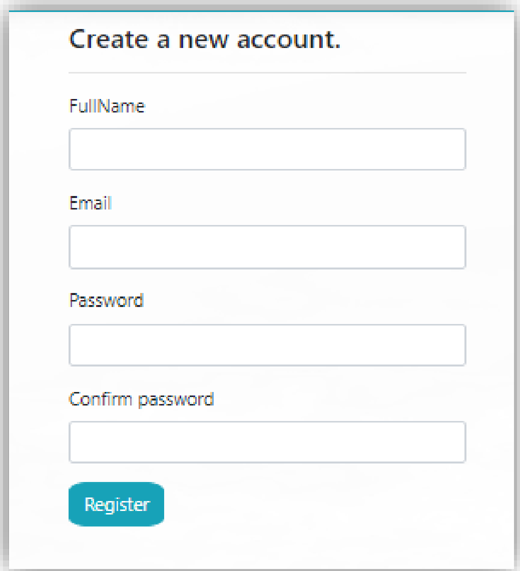
Below is the code to *UpdateUser.cshtml*:

```
@model Racooter.DataTransferObjects.UserDto

<div class="row">
    <div class="form-horizontol col-md-12">
        <h4>Update Account</h4>
        <hr />

        <div class="form-group">
            <label>FullName</label>
            <input value="@Model.FullName" id="txtFullName" class="form-control" />
            <input type="hidden" id="hiddenUserID" value="@Model.id" />
            <span class="text-danger"></span>
        </div>

        <div class="form-group">
            <label>Email</label>
            <input disabled value="@Model.Email" id="txtEmail" class="form-control" />
            <span class="text-danger"></span>
        </div>
        <div class="form-group">
            <label>Block User from Announcements</label>
            <input type="checkbox" value="@Model.IsBlockFromPost" @(Model.IsBlockFromPost == true ? "checked" : "") id="txtIsBlockFromPost" />
            <span class="text-danger"></span>
        </div>

        <div class="form-group">
            <label>Is Reported</label>
            <input type="checkbox" value="@Model.IsReportedForBlock" @(Model.IsReportedForBlock == true ? "checked" : "") id="txtIsReportedForBlock" />
            <span class="text-danger"></span>
        </div>

        <button type="button" class="btn btn-primary" onclick="App.UpdateUser()">Save</button>
        <a class="btn btn-danger" href="/Users/Index">Cancel</a>
    </div>
</div>
```

Figure 51. UpdateUser.cshtml

- ## Login/Register pages

These pages are the ASP.Net Identity forms and are looking like this:



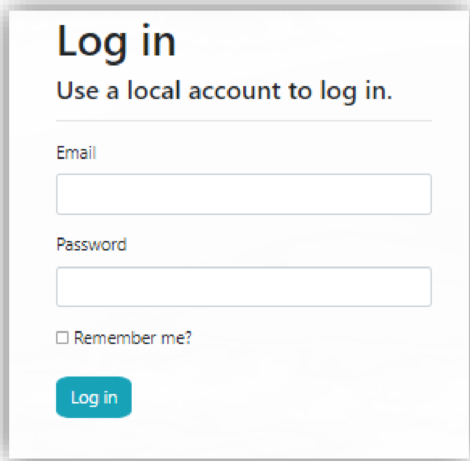


Figure 52. Register/Login view

- **Toolbar visualization**

The toolbar should contain the app logo, and if is pressed it should redirect the user to the first page with the newsletter, a menu that will be a dropdown list with all actions that the current logged user can perform depending on its access role. On the toolbar is present also the home button that will redirect the user to the announcements homepage and the buttons for login/register/logout.

There will be shortcut buttons on the toolbar, displayed in red, that will represent the current role main features, demonstrated below in some screenshots:

Figure 53. Buyer/Seller toolbar

Figure 54. Moderator toolbar

Figure 55. Admin toolbar

4.3 Functionalities implementation

In this section will be described all the features of the application based on the functional requirements, design patterns and project architecture.

It will be presented with code snippets and explainations, also at the end of each feature implementation it will be a demonstration of a full working example.

4.3.1 Announcements features

In this area will be grouped and described all the features that are related with the announcements, and all the flow that was required at implementing the features.

4.3.1.1 Create & Update

This is an important feature that represents the adding of an announcement, by completing all the required fields in the *AddUpdate.cshtml* presented in the previous chapter. Also for the updating of an announcement is used the same method but is adapted in order to have this dual functionality, so will be presented these two features in parallel.

An Ajax call will be made and will provide as parameters an object filled with all user inputs that will be passed on to the controller *Announcements* on the GET method *AddUpdate*:

```
$.ajax({
    type: "POST",
    url: "/Announcements/AddUpdate",
    data: {
        data: obj
    },
    success: function (response) {

        App.SaveImages(response);
        toastr.success('Data Saved Successfully!');
        location.href = "/Announcements/Index";
    }
})
```

Figure 56. SaveAnnouncement Ajax

There should be two methods on the controller with this name, one is of type **HttpGet**, that will be responsable with returning the view, in our case is the adding or updating announcements form, and the other method is of type **HttpPost** that will be called when an announcement should be saved on the database.

In the following code snippet I will present the HttpGet method of AddUpdate present on the Announcement controller:

```
public async Task<IActionResult> AddUpdate(Guid? Id)
{
    var currentUserId = await GetCurrentUserId();
    var currentUserEmail = User.Identity.Name;
    var isAllowed = await _announcementService.IsAllowForAnnouncementCreation(currentUserEmail);
    if (User.IsInRole("Admin") || User.IsInRole("Moderator"))
    {
        isAllowed = true;
    }
    ViewBag.IsAllowed = isAllowed;
    var model = await _announcementService.GetAnnouncement(Id, currentUserId);
    ViewBag.CurrentUserId = currentUserId;
    return View(model);
}
```

Figure 57. AddUpdate HttpGet

So, this method is called also for the **update**, thats why we need to assign by parameter an id, because when we want to update an announcement the app must know which announcement the user want to update, this is solved by providing the id of that particular announcement. But the id can be null in the case of a fresh **add**, because that announcement id is not generated yet, this case is handled as well by adding the question mark representing the nullable status.

In case of an update we should know the current logged user in order to autofill the model with his announcements. The method GetCurrentUserId() is implemented below

```
public async Task<string> GetCurrentUserId()
{
    var name = User.Identity.Name;
    return await _announcementService.GetUserIdByUserName(name);
}
```

Figure 58. GetCurrentUserId controller

We can find out the logged user by asking the ASP Identity with the presented syntax and get the name of the current user. But we need the id of this user, so we need an additional method: *GetUserIdByUserName()* that is implemented in the *AnnouncementService.cs* that will call again another method on the repository following the flow presented in the Application architecture chapter.

```
public async Task<string> GetUserIdByUserName(string userName)
{
    return await _announcementRepository.GetUserIdByUserName(userName);
}
```

Figure 59. GetUserIdByUserName service

As we can see the next method called is *GetUserIdByUserName*() on the *AnnouncementRepository.cs*

```
public async Task<string> GetUserIdByUserName(string userName)
{
    return await _context.Users.Where(x => x.UserName == userName).Select(r => r.Id).FirstOrDefaultAsync();
}
```

Figure 60. GetUserIdByUserName repository

Here I used LINQ to extract from Users table the id of that specific user that name is the one received as parameter, and with *FirstOrDefaultAsync*() we are retrieving the first object asynchronously that fulfills the condition in *Where*.

Now we can go back to **Figure 56** to continue with the explaination. Next we need to retrieve the existing model in case of an **update** by providing as parameter the announcement Id that we need and also the current user Id that we just got it. The method is called *GetAnnouncement*():

```
public async Task<AnnouncementDto> GetAnnouncement(Guid? Id, string CurrentUserId)
{
    var announcement = await _announcementRepository.GetAnnouncement(Id);
    if (announcement.CreatedBy != CurrentUserId)
    {
        await _announcementRepository.AddAnnouncementView(Id);
        _unitOfWork.Commit();
    }
    return announcement;
}
```

Figure 61. GetAnnouncement service

Here we are again the *AnnouncementService.cs* and is calling the method GetAnnouncment() form the repository, that should retrieve the specified announcement with all its properties.

We can obsersve in the next image that is checked if the Id has a value in order to start retrieving the model using GetAll() method present in the Base repository that will retrieve all the entities of that type, but in our case we need the particular one with that Id so we place a Where condition to filter the results. A new instance of **AnnouncementDto** is created and loaded up with model information, and we need also to load **SpecificationDto**, **AnnouncementImageDto** and **CategoryDto** in the same manner, that are representing all the elements forming the announcement according to the diagrams presented in the previous chapters.

```
#region Announcements Management
public async Task<AnnouncementDto> GetAnnouncement(Guid? Id)
{
    var model = new AnnouncementDto();

    if (Id.HasValue)
    {
        model = await GetAll().Where(x => x.AnnouncementId == Id).Select(x => new AnnouncementDto
        {
            AnnouncementId = x.AnnouncementId,
            Category = x.Category,
            CreatedBy = x.SellerInfo.Id,
            CreatedDate = x.CreatedDate,
            Date = x.Date,
            Description = x.Description,
            IsApprovedByAdmin = x.IsApprovedByAdmin,
            Price = x.Price,
            Title = x.Title,
            Views = x.Views,
            UserName = x.SellerInfo.FullName,
            UserEmail = x.SellerInfo.Email,
            Location = x.Location,
            PhoneNumber = x.PhoneNumber

        }).FirstOrDefaultAsync();

        model.CategoryString = await _context.Categories.Where(x => x.Id == model.Category).Select(r => r.Name).FirstOrDefaultAsync();

        model.Specification = await _context.Specifications.Where(x => x.AnnouncementId == Id.Value).Select(data => new SpecificationDto
        {
            BodyType = data.BodyType,
            BodyTypeSelected = data.BodyTypeSelected,
            Color = data.Color,
            Emissions = data.Emissions,
            EngineSize = data.EngineSize,
            GearBox = data.GearBox,
            GetFuelType = data.GetFuelType,
            FuelTypeSelected = data.FuelTypeSelected,
            HadAccident = data.HadAccident,
            HasABS = data.HasABS,
            HasCruiseControl = data.HasCruiseControl,
            HasDualZoneClimate = data.HasDualZoneClimate,
            HasElectricMirrors = data.HasElectricMirrors,
            HasESP = data.HasESP,
            HasFullElectricWin = data.HasFullElectricWin,
            HasHeatedSeats = data.HasHeatedSeats,
            HasHeatedStWheel = data.HasHeatedStWheel,
            HasLogHistory = data.HasLogHistory,
            HasVentedSeats = data.HasVentedSeats,
            HasWarranty = data.HasWarranty,
```

Figure 62. GetAnnouncement repository

After all the announcement info are loaded in the model, is created the AddAnnouncementView that is basically returning the retrieved data in the user interface. If the announcement Id is Not provided then it means is an Add, then the model will be empty and the app will wait for the user to populate the fields.

The save announcement method is located on the HttpPost function AddUpdate in the controller:

```
[HttpPost]
public async Task<IActionResult> AddUpdate(AnnouncementDto data, string CurrentUserId)
{
    if (string.IsNullOrEmpty(CurrentUserId))
        CurrentUserId = await GetCurrentUserId();
    var announcementId = await _announcementService.SaveAnnouncement(data, CurrentUserId);
    return Ok(announcementId);
}
```

Figure 63. AddUpdate HttpPost

The main method that will be called here is *SaveAnnouncement*() with the parameters an AnnouncementDto and the current user id to know who is the seller of this announcement.

I will represent the method directly on the repository because there is the level where the main logic is present, the service is used to call the methods from the repository, and it can have other methods.

```
public async Task<Guid> SaveAnnouncement(AnnouncementDto data, string CurrentUserId)
{
    if (data.AnnouncementId != null && data.AnnouncementId != Guid.Empty)
    {
        var announcement = _context.Announcements.Where(x => x.AnnouncementId == data.AnnouncementId).FirstOrDefault();
        if (announcement != null)
        {
            announcement.Category = data.Category;
            announcement.Date = data.Date;
            announcement.Description = data.Description;
            announcement.Title = data.Title;
            announcement.Price = data.Price;
            announcement.PhoneNumber = GetUserPhoneNumberById(CurrentUserId);
            announcement.Location = data.Location;
        }
        if (data.Specification != null)
        {
            var specification = _context.Specifications.Where(x => x.AnnouncementId == data.AnnouncementId).FirstOrDefault();
            if (specification != null)
            {
                specification.BodyType = data.Specification.BodyType;
                specification.BodyTypeSelected = GetBodyType(specification.BodyType);
                specification.Color = data.Specification.Color;
                specification.Emissions = data.Specification.Emissions;
                specification.EngineSize = data.Specification.EngineSize;
                specification.GearBox = data.Specification.GearBox;
                specification.GetFuelType = data.Specification.GetFuelType;
                specification.FuelTypeSelected = GetFuelType(specification.GetFuelType);
                specification.HadAccident = data.Specification.HadAccident;
                specification.HasABS = data.Specification.HasABS;
                specification.HasCruiseControl = data.Specification.HasCruiseControl;
                specification.HasDualZoneClimate = data.Specification.HasDualZoneClimate;
                specification.HasElectricMirrors = data.Specification.HasElectricMirrors;
                specification.HasESP = data.Specification.HasESP;
                specification.HasFullElectricWin = data.Specification.HasFullElectricWin;
                specification.HasHeatedSeats = data.Specification.HasHeatedSeats;
                specification.HasHeatedStWheel = data.Specification.HasHeatedStWheel;
                specification.HasLogHistory = data.Specification.HasLogHistory;
                specification.HasVentedSeats = data.Specification.HasVentedSeats;
                specification.HasWarranty = data.Specification.HasWarranty;
                specification.IsFullOptions = data.Specification.IsFullOptions;
                specification.IsNegotiable = data.Specification.IsNegotiable;
                specification.Make = data.Specification.Make;
                specification.Mileage = data.Specification.Mileage;
                specification.Model = data.Specification.Model;
                specification.NrOfDoors = data.Specification.NrOfDoors;
                specification.Power = data.Specification.Power;
                specification.Year = data.Specification.Year;
                await _context.SaveChangesAsync();
            }
        }
        return announcement.AnnouncementId;

    }
    else
```

Figure 64. SaveAnnouncement update

This method should make the **Update** and the **Create** so in the picture attached above is the section that will make the update and below will place the other section that will be the code for adding an announcement, sepparated by the *if else* structure that is checking if the *AnnouncementDto* model has the *AnnouncementId* filled with a valid value, that is meaning we got an existing announcement that we want to update it. The update feature will need to retrieve first the old announcement and update its fields according to the new values from *AnnouncementDto*.

Here is presented the second section that will perform the **Create** of a new announcement and is filled with the data provided by user inputs and then taken further by the AnnouncementDto. Also a new Specification object is made.

```
else
{
    var ann = new Announcement();

    ann.Category = data.Category;
    ann.Date = data.Date;
    ann.Description = data.Description;
    ann.Title = data.Title;
    ann.Price = data.Price;
    ann.CreatedDate = DateTime.Now;
    ann.Views = 0;
    ann.IsApprovedByAdmin = false;
    ann.SellerInfo = GetSeller(CurrentUserId);
    ann.Location = data.Location;
    ann.PhoneNumber = GetUserPhoneNumberById(CurrentUserId);

    Add(ann);
    await _context.SaveChangesAsync();

    if (data.Specification != null)
    {
        var specification = new Specification();
        specification.AnnouncementId = ann.AnnouncementId;
        specification.BodyType = data.Specification.BodyType;
        specification.BodyTypeSelected = GetBodyType(specification.BodyType);
        specification.Color = data.Specification.Color;
        specification.Emissions = data.Specification.Emissions;
        specification.EngineSize = data.Specification.EngineSize;
        specification.GearBox = data.Specification.GearBox;
        specification.GetFuelType = data.Specification.GetFuelType;
        specification.FuelTypeSelected = GetFuelType(specification.GetFuelType);
        specification.HadAccident = data.Specification.HadAccident;
        specification.HasABS = data.Specification.HasABS;
        specification.HasCruiseControl = data.Specification.HasCruiseControl;
        specification.HasDualZoneClimate = data.Specification.HasDualZoneClimate;
        specification.HasElectricMirrors = data.Specification.HasElectricMirrors;
        specification.HasESP = data.Specification.HasESP;
        specification.HasFullElectricWin = data.Specification.HasFullElectricWin;
        specification.HasHeatedSeats = data.Specification.HasHeatedSeats;
        specification.HasHeatedStWheel = data.Specification.HasHeatedStWheel;
        specification.HasLogHistory = data.Specification.HasLogHistory;
        specification.HasVentedSeats = data.Specification.HasVentedSeats;
        specification.HasWarranty = data.Specification.HasWarranty;
        specification.IsFullOptions = data.Specification.IsFullOptions;
        specification.IsNegotiable = data.Specification.IsNegotiable;
        specification.Make = data.Specification.Make;
        specification.Mileage = data.Specification.Mileage;
        specification.Model = data.Specification.Model;
        specification.NrOfDoors = data.Specification.NrOfDoors;
        specification.Power = data.Specification.Power;
        specification.Year = data.Specification.Year;
        _context.Specifications.Add(specification);
        await _context.SaveChangesAsync();
    }
```

Figure 65. SaveAnnouncement create

The saving in the database table is done using the context, represented by the *RacooterDbContext.cs* file and calling *SaveChangesAsync*() that will update the database with the new entry.

The Add() method represents the add from the base repository that performs an add of the specified entity type.

4.3.1.1.1 Demo Create

Let's say we completed the AddUpdate form with the following values:

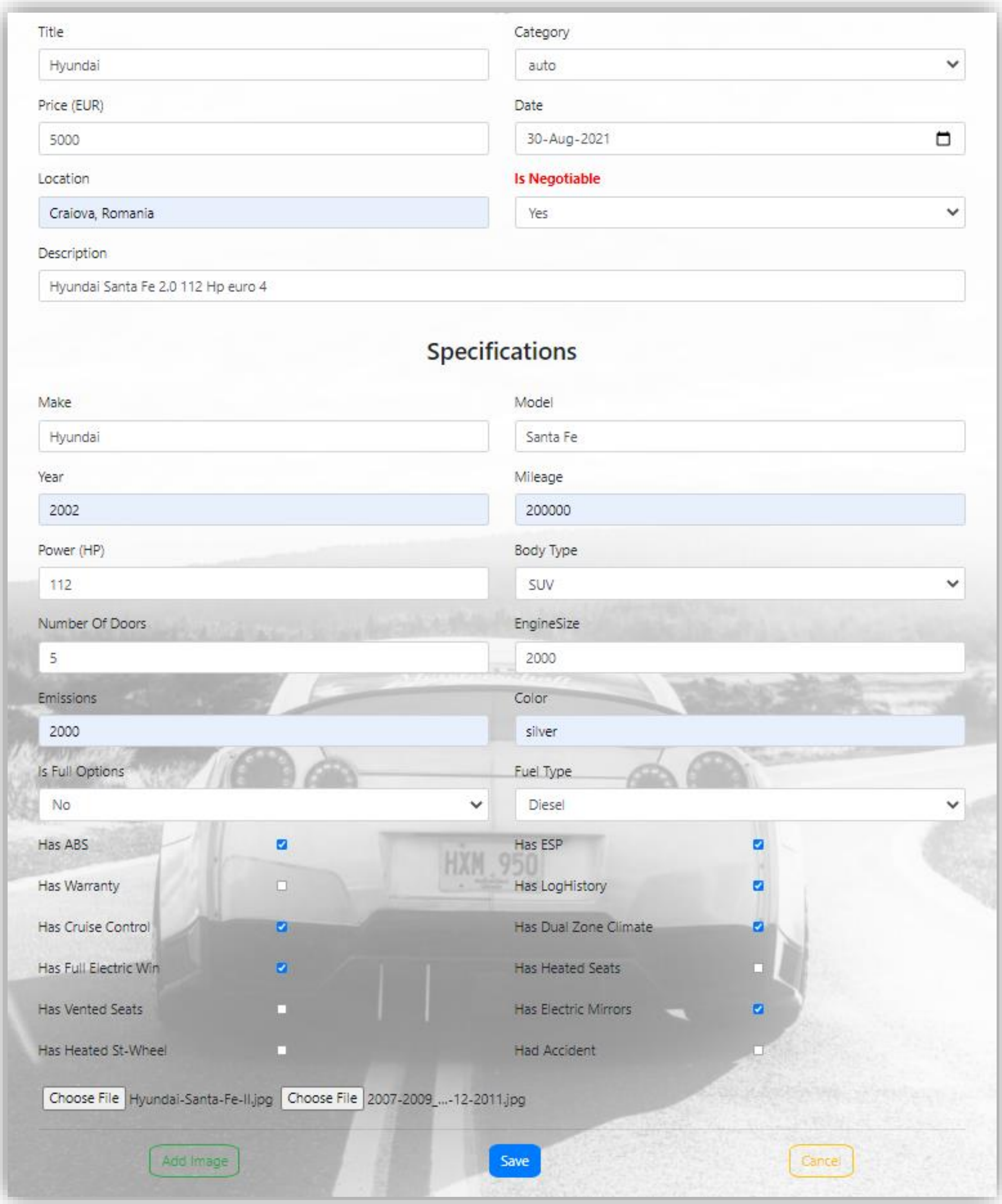


Figure 66. Create demo A

Then we can check the announcement panel to see the added entry with Pending approval status.

In the below picture we can notice that the announcement is created for the current logged user and assigned as the seller.

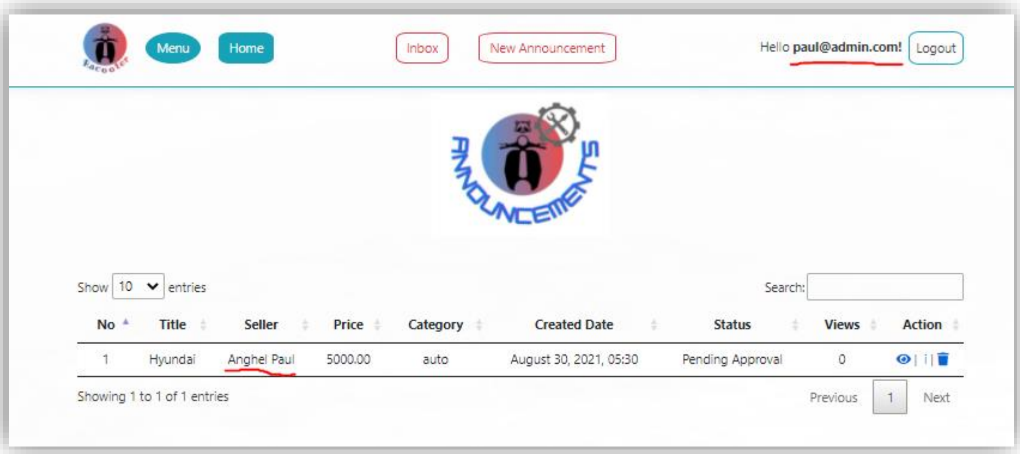


Figure 67. Create demo B

And on the preview button we can see all the announcement information are saved successfully:

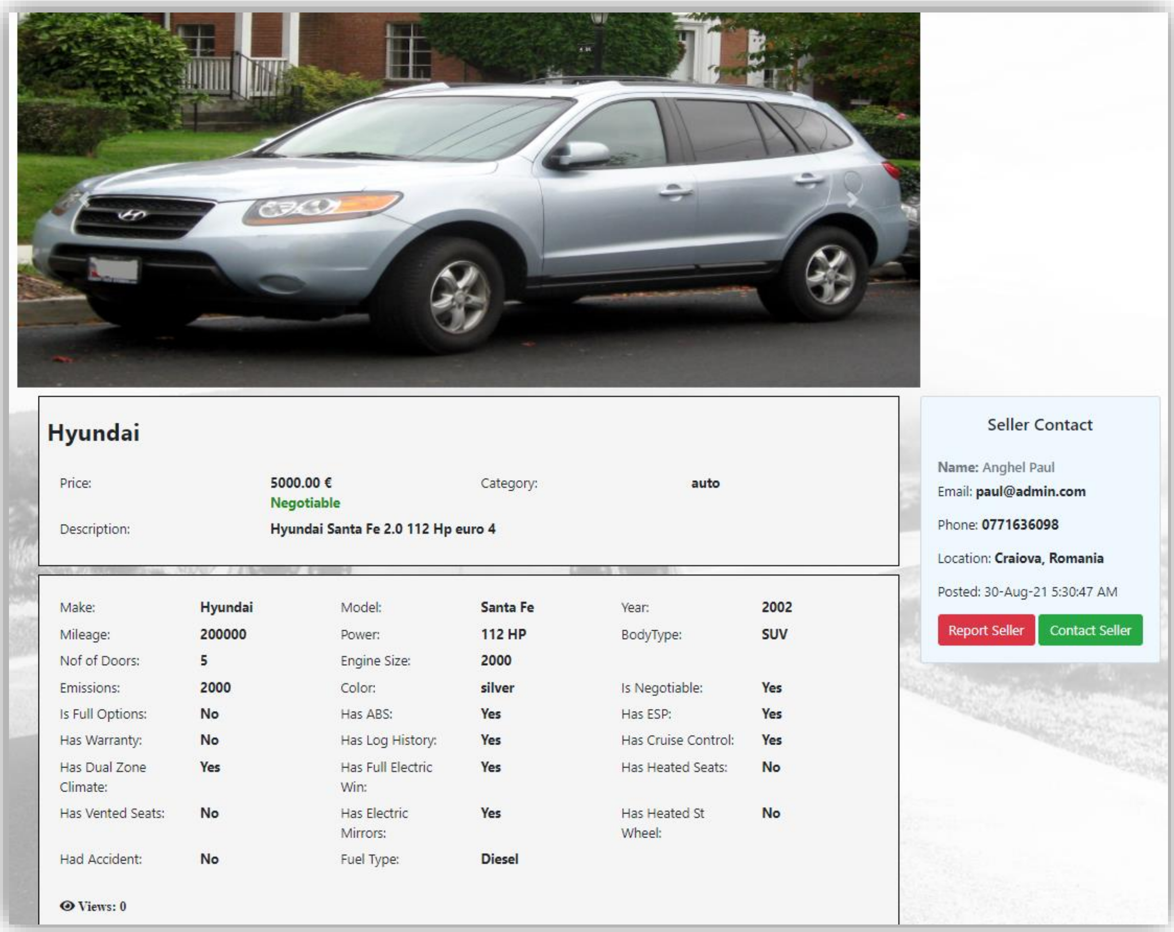


Figure 68. Create demo C

4.3.1.1.2 Demo Update

Let's try to update the announcement that we just added and modify the **Price** to **3500** EUR, the negotiable status will be set to No so on the interface will appear as **„Fixed"**, the Power will be updated to **200 HP**, the checkbox **HasWarranty** should be checked in order to be displayed as **yes**.

The updated announcement will be updated as expected:

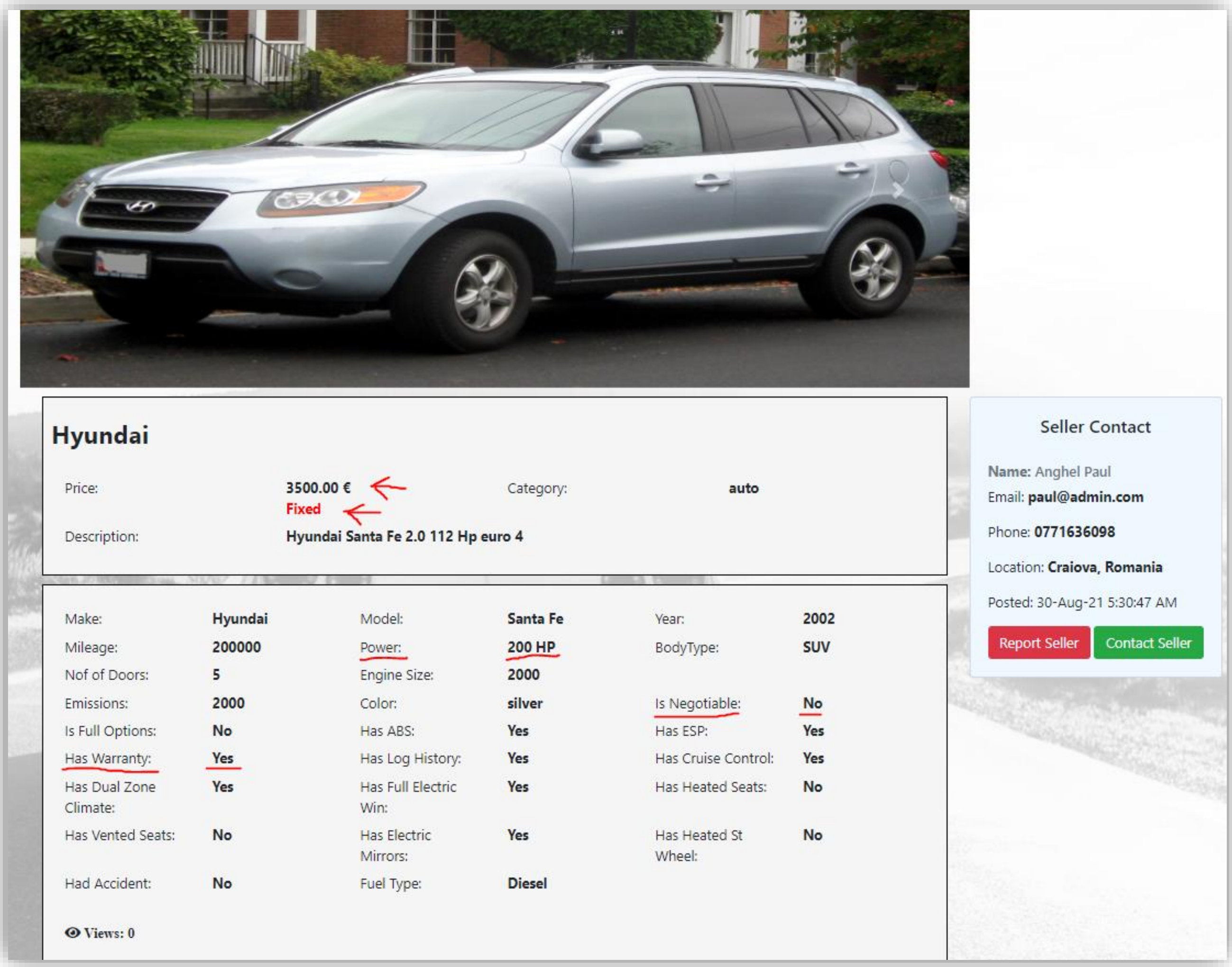


Figure 69. Update demo

4.3.1.2 Delete

The delete announcement method is triggered by the trashcan icon from the front-end of announcements panel. In order to delete an entry we need the id of it in order to search it and destroy it by removing from database using the repository.

When the button is clicked will call directly the method *Delete*() from the controller:

```
public async Task<IActionResult> Delete(Guid id)
{
    await _announcementService.DeleteAnnouncement(id);
    return RedirectToAction("Index");
}
```

Figure 70. Delete method

Then on the AnnouncementService.cs is present the method *DeleteAnnouncement*() that will call the repository that will deal with the delete:

```
public async Task DeleteAnnouncement(Guid id)
{
    var announcement = await _context.Announcements.FindAsync(id);
    if (announcement != null)
    {
        var specifications = _context.Specifications.Where(x => x.AnnouncementId == announcement.AnnouncementId).AsEnumerable();
        var announcementImages = _context.AnnouncementImages.Where(x => x.AnnouncementId == announcement.AnnouncementId).AsEnumerable();
        _context.Specifications.RemoveRange(specifications);
        _context.AnnouncementImages.RemoveRange(announcementImages);
        _context.Announcements.Remove(announcement);
        await _context.SaveChangesAsync();
    }
}
```

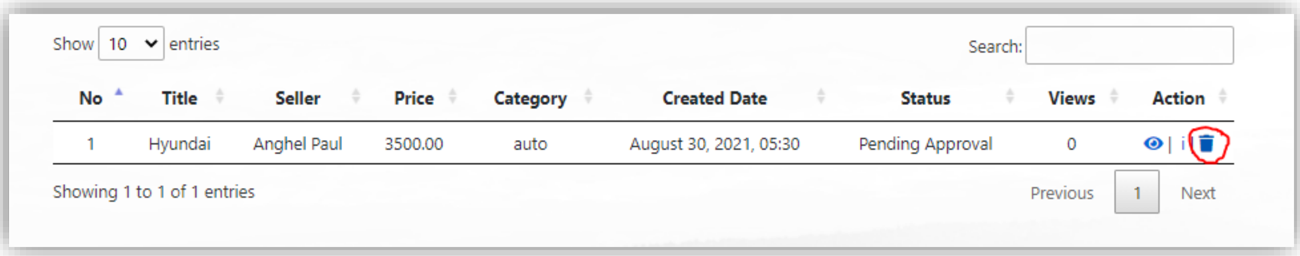
Figure 71. DeleteAnnouncement repository

We need to extract the needed announcement first using the id, similar to the update method. And we need also to remove the other objects that are linked with an announcement, like specification and announcement images.

Using *RemoveRange()* method to track down the entity and delete the references., and then saving the changes on the database.

4.3.1.2.1 Demo

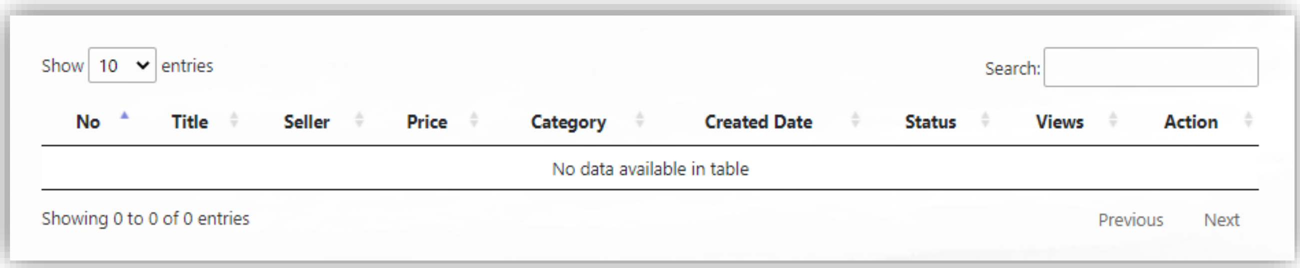
So, we got our entry that we just added:



No	Title	Seller	Price	Category	Created Date	Status	Views	Action
1	Hyundai	Anghel Paul	3500.00	auto	August 30, 2021, 05:30	Pending Approval	0	

Figure 72. Delete demo A

And we press the trashcan icon in order to delete the entry, and the entry will be gone as expected, there wont be other entries in the table:



No	Title	Seller	Price	Category	Created Date	Status	Views	Action
No data available in table								

Figure 73. Delete demo B

4.3.1.3 Filters

In this section will be implemented and demonstrated the filtering announcement feature. In the front-end implemenation part we could observe how the filters are displayed, being part of the partial view *Home.cshtml* and all the announcement cards are displayed in the *_Home.cshtml* along with the appended filter boxes in the top of the page.

When the user provides input for the filter, an pressing the apply button, it will call a method on javascript named *LoadHomePageAnnouncement*() that will have an handler method that will call the controller method _Home:

```
let handleLoadHomePageAnnouncements = ($that) => {

    let loaderHtml = '<div class="col-md-12"><h3 class="text-center alert alert-warning"> Please wait while loading data!!!</h3></div>';
    $('#AnnouncementsContainer').html(loaderHtml);
    var $parentDiv = $($that).parents('#FiltersContainerCollapse');

    var filterObj = {
        Title: $('#ftrTitle').val(),
        Category: $('#ftrCategory').val(),
        Price: $('#ftrPrice').val(),
        Model: $('#ftrModel').val(),
        Make: $('#ftrMake').val(),
        Year: $('#ftrYear').val(),
        Mileage: $('#ftrMileage').val(),
        Power: $('#ftrPower').val(),
        PageNumber: 1
    };

    $.ajax({
        type: "Get",
        url: "/Announcements/_Home",
        data: filterObj,
        success: function (response) {
            $('#AnnouncementsContainer').html(response);
        }
    })
}
```

Figure 74. LoadHomePageAnnouncements js

All the user input values are loaded in filterObj from the UI elements and are sent to the controller, and after it receive a successfull response, it will display the filtered announcments in the *AnnouncementsContainer* div id. Below is the method *_Home* on the controller:

```
[AllowAnonymous]
public async Task<IActionResult> _Home(AnnouncementFilter filter)
{
    var currentUserId = string.Empty;
    if (User.Identity.IsAuthenticated)
    {
        currentUserId = await GetCurrentUserId();
    }
    ViewBag.CurrentUserId = currentUserId;
    var approvedAnnouncements = await _announcementService.GetFiltererdAnnouncements(filter, currentUserId);
    return PartialView(approvedAnnouncements);
}
```

Figure 75. _Home method controller

Basically is calling further the function on the service GetFilteredAnnouncements() that will deal also with recommender system presented in the further chapters, thats why the function is taking also the current user Id, but we will describe that in detail at recommender system chapter.

On the repository there is the method called by the service:

```
public async Task<List<AnnouncementDto>> GetFiltererdAnnouncements(AnnouncementFilter filter, string CurrentUserId)
{
    bool isFilterApplied = false;
    if (!string.IsNullOrEmpty(CurrentUserId))
    {
        isFilterApplied = await SaveFilterByUser(filter, CurrentUserId);
    }

    var query = (from x in _context.Announcements
                 join y in _context.Specifications on x.AnnouncementId equals y.AnnouncementId
                 where x.IsApprovedByAdmin == true
                 select new
                 {
                     Announcement = x,
                     Specification = y
                 });


    if (!string.IsNullOrEmpty(filter.Title))
    {
        query = query.Where(x => x.Announcement.Title.ToLower().Contains(filter.Title.ToLower()));
    }
    if (filter.Category.HasValue)
    {
        query = query.Where(x => x.Announcement.Category == filter.Category.Value);
    }
    if (!string.IsNullOrEmpty(filter.Make))
    {
        query = query.Where(x => !string.IsNullOrEmpty(x.Specification.Make) && x.Specification.Make.ToLower().Contains(filter.Make.ToLower()));
    }
    if (filter.Mileage.HasValue)
    {
        query = query.Where(x => x.Specification.Mileage.HasValue && x.Specification.Mileage.Value == filter.Mileage.Value);
    }
    if (!string.IsNullOrEmpty(filter.Model))
    {
        query = query.Where(x => !string.IsNullOrEmpty(x.Specification.Model) && x.Specification.Model.ToLower().Contains(filter.Model.ToLower()));
    }
    if (filter.Power.HasValue)
    {
        query = query.Where(x => x.Specification.Power.HasValue && x.Specification.Power.Value == filter.Power.Value);
    }
    if (filter.Price.HasValue)
    {
        query = query.Where(x => x.Announcement.Price <= filter.Price.Value);
    }
    if (filter.Year.HasValue)
    {
        query = query.Where(x => x.Specification.Year.HasValue && x.Specification.Year.Value == filter.Year.Value);
    }

    var data = await query.Select(x => new AnnouncementDto
    {
        AnnouncementId = x.Announcement.AnnouncementId,
        Category = x.Announcement.Category,
        CreatedDate = x.Announcement.CreatedDate,
        Date = x.Announcement.Date,
        Price = x.Announcement.Price,
        Title = x.Announcement.Title,
        Views = x.Announcement.Views,
```

Figure 76. GetFilteredAnnouncements repository A

We are interested in the filter feature for now, so basically I am extracting in a query only the approved announcements, that means are available for display, and also the Specification object related to each announcement using the AnnouncementId and a join on Specifications table.

Then there are multiple if statements that are checking to see if the user provided information in the filter box and about which fields, in order to filter more the extracted query at the previous step with the provided values by the user input. This is checked using LINQ syntax with *Where* to specify the condition and *Contains* that will do some checks, for example the title from any extracted announcements are checked if are containing the keyword searched by the user for the title field.

After the query is containing only the announcements that satisfy all the filtered conditions, it will create a new AnnouncementDto object that will be filled with the information from the filtered query.

Then we need also to extract the announcement images and category using the *AnnouncementId*, and return the data that will be displayed in the frontend as filtered announcements:

```
var announcementIds = data.Select(x => x.AnnouncementId).ToList();

var allImages = await _context.AnnouncementImages.Where(x => announcementIds.Contains(x.AnnouncementId.Value)).Select(x => new
{
    ImagePath = x.ImagePath,
    AnnouncementId = x.AnnouncementId
}).ToListAsync();

var categories = await _context.Categories.ToListAsync();

foreach (var item in data)
{
    item.ImagesPath = allImages.Where(x => x.AnnouncementId == item.AnnouncementId).Select(x => x.ImagePath).ToList();
    item.CategoryString = categories.Where(x => x.Id == item.Category).Select(r => r.Name).FirstOrDefault();
    item.FilterColumn = item.FilterColumn == '\0' ? Char.MinValue : item.FilterColumn;
    bool isOk = item.FilterColumn == '\0';
}

return data;
```

Figure 77. GetFilteredAnnouncements repository B

4.3.1.3.1 Demo

Here will be a quick demo on the filters option. Let's provide some filter information and see if we get announcement that are matching all the filters not only one at once, we need all the constraints to work in order to be correct.

Let's assume we search in the title the keyword „golf", then we got all the announcements that contains this keyword in the title:

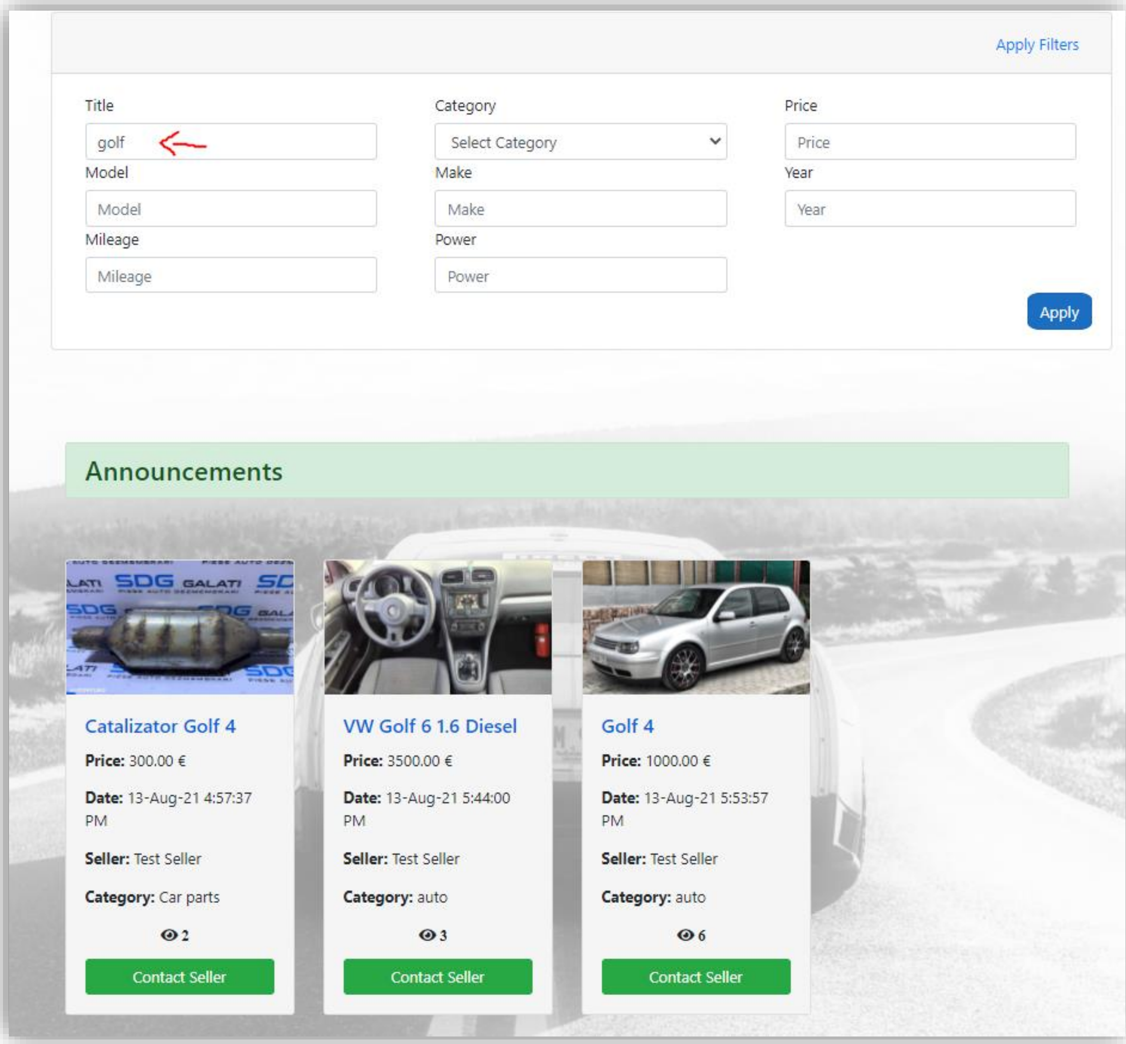


Figure 78. Filters demo A

Now let’s apply a second filter in order to narrow down the filtering, we select also the category to be „auto”, so the result should be the one with „Catalizator Golf 4” should be gone because is part of a different category, and we can add another filter by price to be „1200” meaning that only the announcement „Golf 4” should remain because is meeting all the requirements:

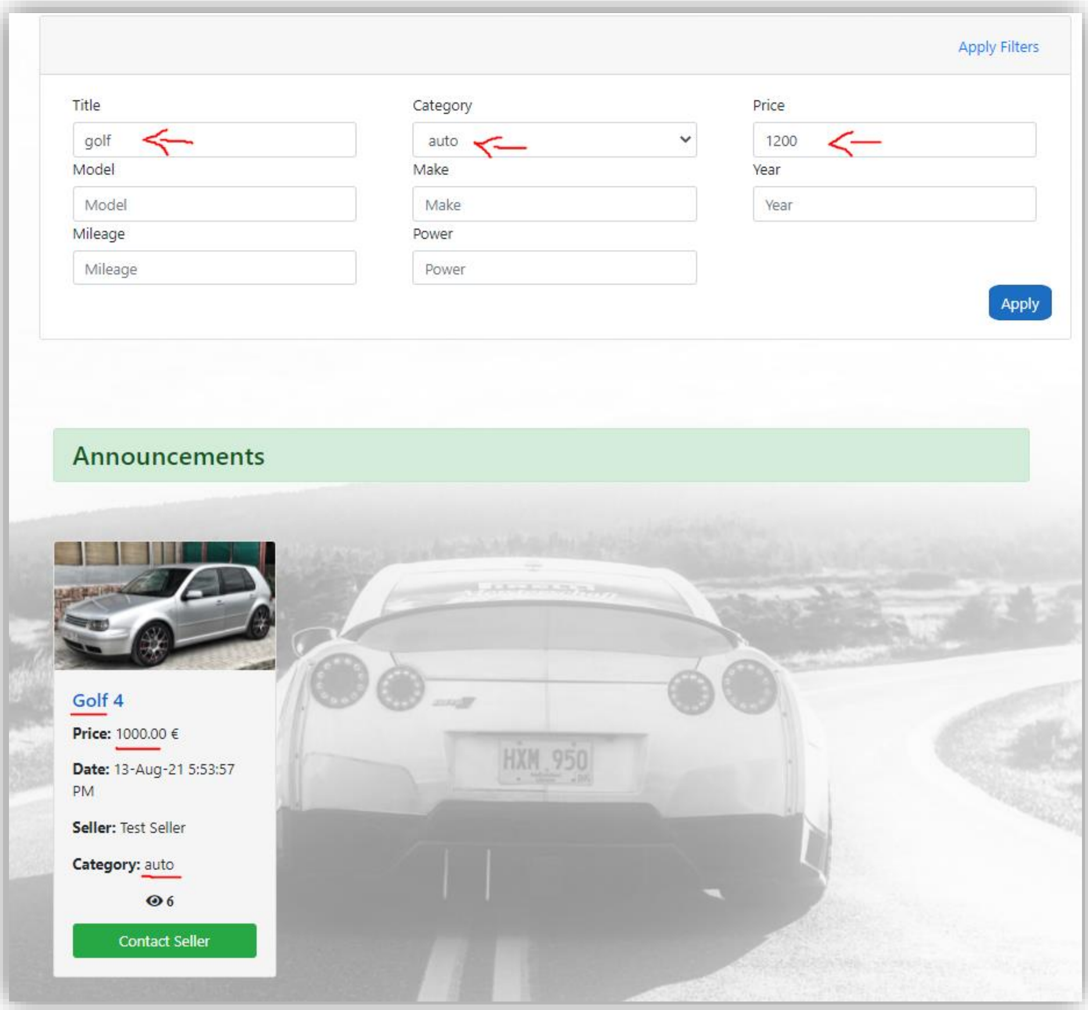


Figure 79. Filters demo B

As we can observe the filtering feature is working properly and we got the expected result.

4.3.1.4 Review

One of the application requirement is to not display the announcement immediately after was created, it should wait for the admin or moderator approval in order to review that announcement. So, the announcement is not visible yet on the home page only if was approved, by changing the status in „Approved”. When a new announcement is created, on the status properity will receive „Pending Approval” so the system will know not to display this kind of announcements.

The reviewing announcements feature is accessed by the moderator and admin, so when they open the announcement panel will notice an „Approve” button on the update announcement.

When the button is pressed in order to approve an announcement it will call a javascript method named *handleApproveAnnouncement*(), that will make a GET call on the *Announcements* controller calling the *Approve* method:

```
let handleApproveAnnouncement = (AnnouncementId) => {
    $.ajax({
        type: "GET",
        url: "/Announcements/Approve",
        data: {
            Guid: AnnouncementId
        },
        success: function (response) {
            toastr.success("Announcement is approved!")
            location.href = '/Announcements/Index';
        }
    })
}
```

Figure 80. ApproveAnnouncement js

In order to know which announcement to be approved, we should send by parameter the AnnouncementId. Below is the method on the repository that is handling the approval:

```
public async Task Approve(Guid guid)
{
    var announcement = await _context.Announcements.FindAsync(guid);
    announcement.IsApprovedByAdmin = true;
    _context.Entry(announcement).State = EntityState.Modified;
}
```

Figure 81. Approve repository

We use the is to extract that particular announcement and update the field IsApprovedByAdmin to true, and then save the changes back. Basically is like a short update that is performed on the announcement but only on that particular field.

And in the frontend we are allowing only announcements that has the property IsApprovedByAdmin set to true, so now this announcement should appear in the home page.

4.3.1.4.1 Demo

Below is an example of approving an announcement and see how the status is changing in „Approved" and also it should appear on the announcements page.

Let’s consider this „Passat CC” announcement that is in the pending list of the admin:

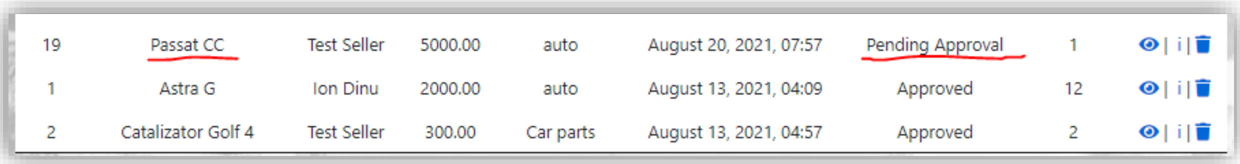


Figure 82. Approve demo A

Then we approve it by pressing the „i” button and instead of save and update we have the approve and cancel button:

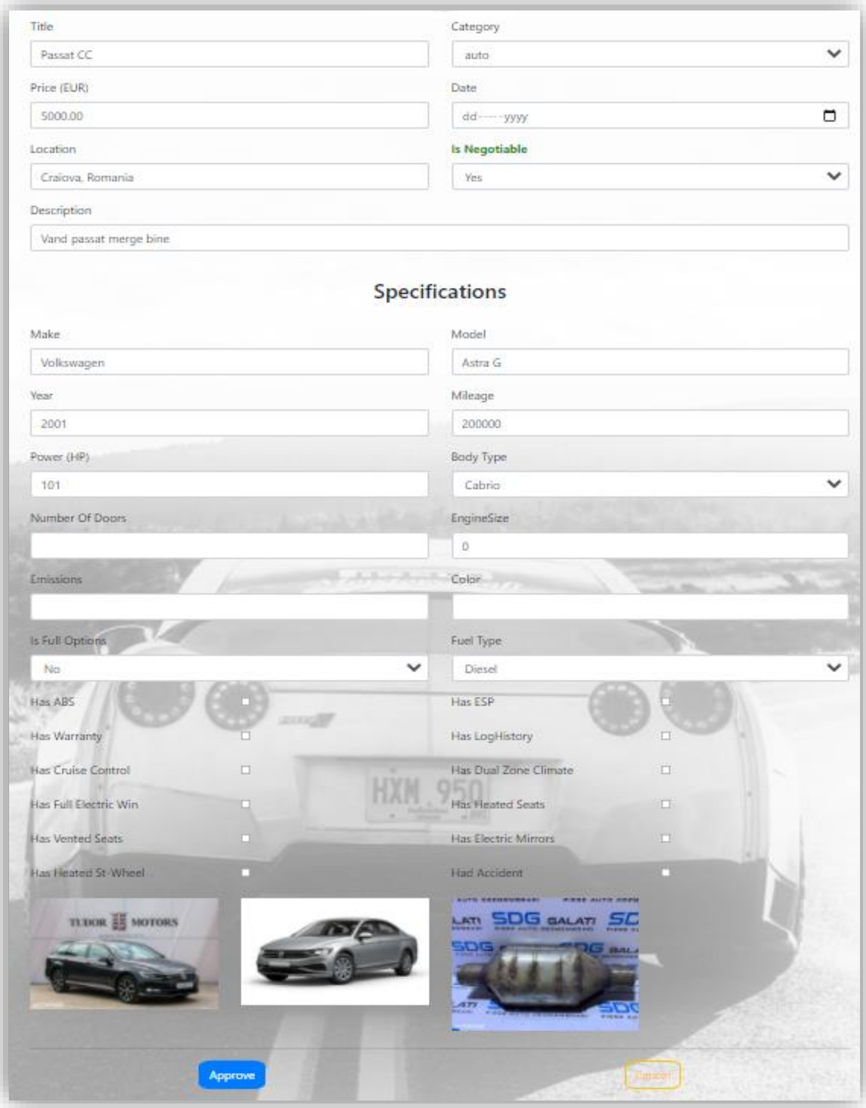


Figure 83. Approve demo B

And we can observe that the announcement status is changed to „Approved” and is displayed on the announcements list, we can search it by title as its showned below:

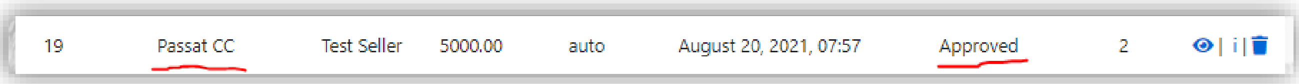


Figure 84. Approve demo C

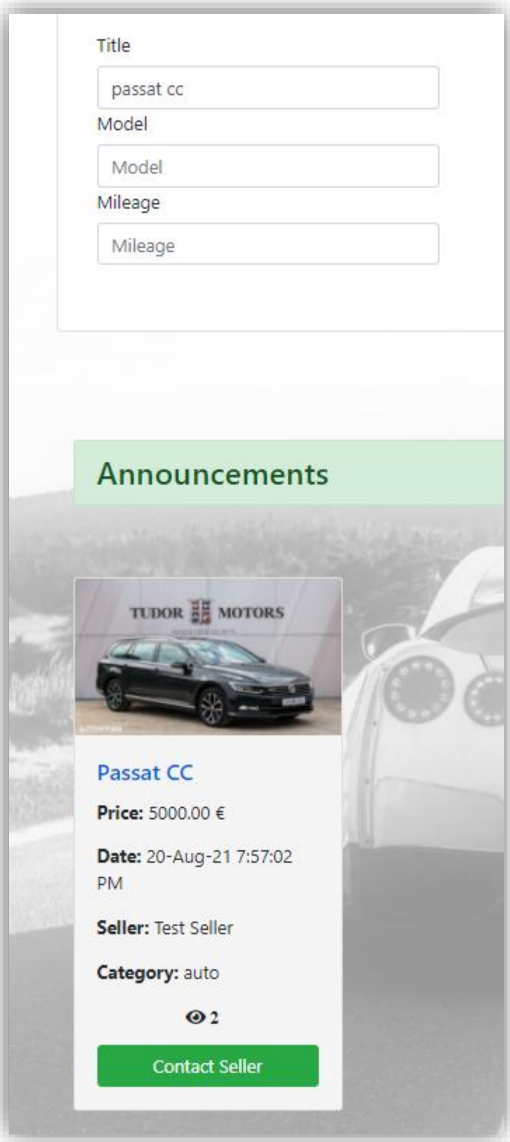


Figure 85. Approve demo D

4.3.1.5 Number of Views

This feature is allowing the seller of the announcement to track the visibility and exposure of his announcements, but also the other users can see the number of views in order to have an ideea of how much popularity that buyer has or how much in demand is the respective car.

Basically when a new announcement is created, the views counter is set to 0 because no one has watched it yet. The thing is, the views counter should not increment when the author of it is checking or visualize it, because can manipulate his own views being false.

The views counter will increase when other users than the seller, are clicking on that announcement that will be redirected to announcement details page and the counter will increase by one unit.

The method responsable when an announcement is clicked, its called GetAnnouncement() that was presented in the other feature description, but now we are interested in the AddAnnouncementView() method. We can observe below that a check is made before calling the method, to see if is accessed not by the actual seller that created it, in order to not generate views for himself:

```
public async Task<AnnouncementDto> GetAnnouncement(Guid? Id, string CurrentUserId)
{
    var announcement = await _announcementRepository.GetAnnouncement(Id);
    if (announcement.CreatedBy != CurrentUserId)
    {
        await _announcementRepository.AddAnnouncementView(Id);
        _unitOfWork.Commit();
    }
    return announcement;
}
```

Figure 86. AddAnnouncementView service

Then the method is implemented on the repository:

```
public async Task AddAnnouncementView(Guid? id)
{
    if (id != null && id != Guid.Empty)
    {
        var announcement = await _context.Announcements.FindAsync(id);
        announcement.Views += 1;
    }
}
```

Figure 87. AddAnnouncementView repository

First is checking the id to make sure is not null, in order to retrieve that particular announcement by id using the built-in method *FindAsync*() that will return an entity given its primary key value, in our case will return an *Announcement* type entry that will be increased the field *Views* by one unit.

4.3.1.5.1 Demo

Let’s take an announcement and see how the number of views is increasing:

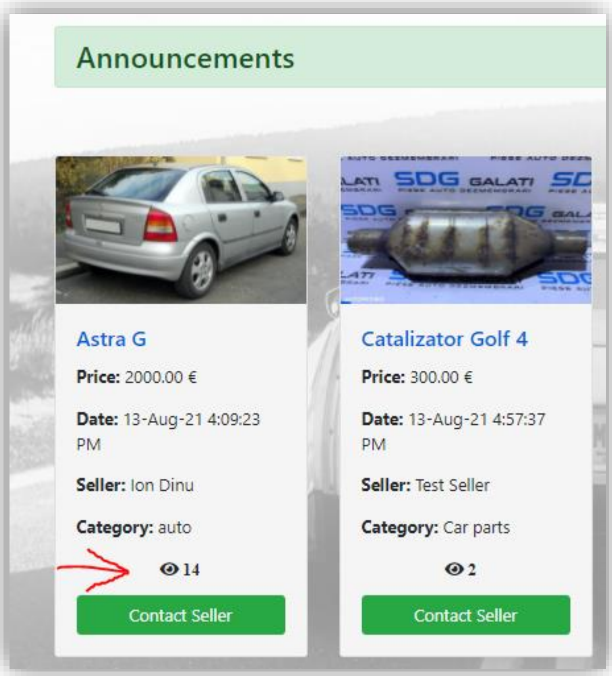


Figure 88. Views demo A

Then we click on it and are present in the announcement details page, then when we check again the announcements, the view is increased by one as expected, and we are logged as the admin, not the seller:

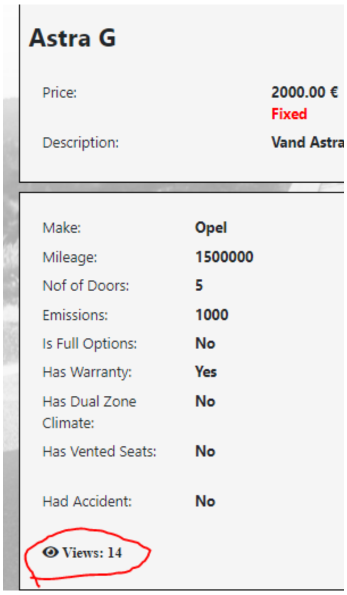


Figure 89. Views demo B

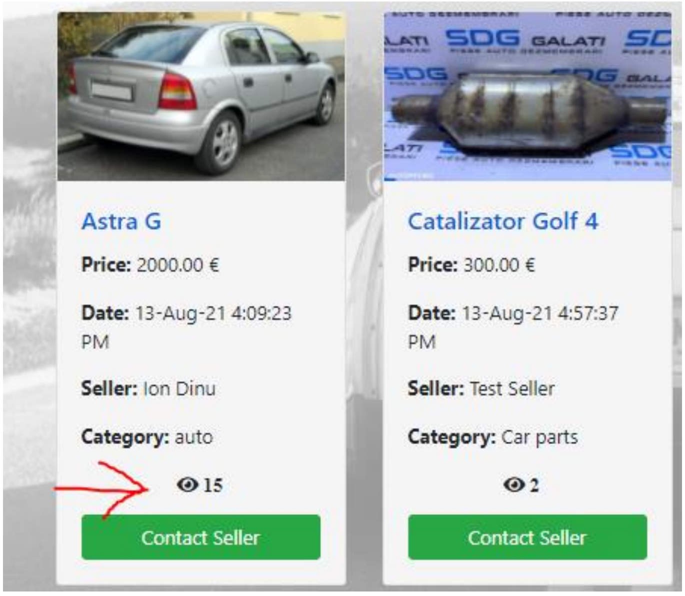


Figure 90. Views demo C

4.3.2 Newsletter feature

In this section will be presented the newsletter that is present on the first page of the application. Only the admin and moderator role can add or delete a news.

After completing the fields and the button Save is pressed, then it will be called a method on javascript that will make an Ajax request to the HttpPost method *SaveNewsPost* from the *Announcements* controller, providing as parameter an object filled with all the extracted values from the user input for title, sub-title and content:

```
let handleSaveNewsPost = () => {
    let txtPostTitle = $('#txtPostTitle').val();
    let txtPostSubTitle = $('#txtPostSubTitle').val();
    let txtPostContent = $('#txtPostContent').val();

    if (txtPostTitle == "" || txtPostSubTitle == "" || txtPostContent == "") {
        toastr.error("Fields are required");
        return false;
    }

    let _hhtml = '<div class="col-md-12 text-center"><h4 class="alert alert-success"> Please wait while saving data</h4></div>';
    $('#NewsPostsFormContainer').append(_hhtml);

    $.ajax({
        type: "POST",
        url: "/Announcements/SaveNewsPost",
        data: {
            Title: txtPostTitle,
            SubTitle: txtPostSubTitle,
            Content: txtPostContent
        },
        success: function (response) {
            location.reload();
        }
    })
}
```

Figure 91. SaveNewsPost js

The method on the controller will call the service and then the repository where the saving will be made. Basically we need to create a new object of the model *NewsPost* filling it with the parameters provided by the admin or moderator, then an Add is made with the new created entity and saved in the NewsPosts table using the context. The changes are saved by calling the method *SaveChangesAsync*()

```
public async Task SaveNewsPost(string Title, string SubTitle, string Content)
{
    var newsPost = new NewsPost()
    {
        Content = Content,
        CreatedDate = DateTime.Now,
        SubTitle = SubTitle,
        Title = Title
    };
    _context.NewsPosts.Add(newsPost);
    await _context.SaveChangesAsync();
}
```

Figure 92. SaveNewsPost repository

And the delete feature on the news post is implemented using this method on the repository:

```
public async Task DeletePost(int Id)
{
    var post = await _context.NewsPosts.FindAsync(Id);
    if (post != null)
    {
        _context.NewsPosts.Remove(post);
        await _context.SaveChangesAsync();
    }
}
```

Figure 93. DeletePost repository

Basically when the admin/moderator is pressing the trashcan icon on a news post, it will trigger on the controller the method *DeleteNewsPost* that will call the service and then the method presented above on the repository. This function will take as parameter the id of the news post that was clicked and retrieved the correspondent news object, then it will be deleted by using the *Remove* method from the base repository, and the changes are saved on the database.

4.3.2.1 Demo

Let’s try to add a new post and then delete it. We complete the following values on the interface:

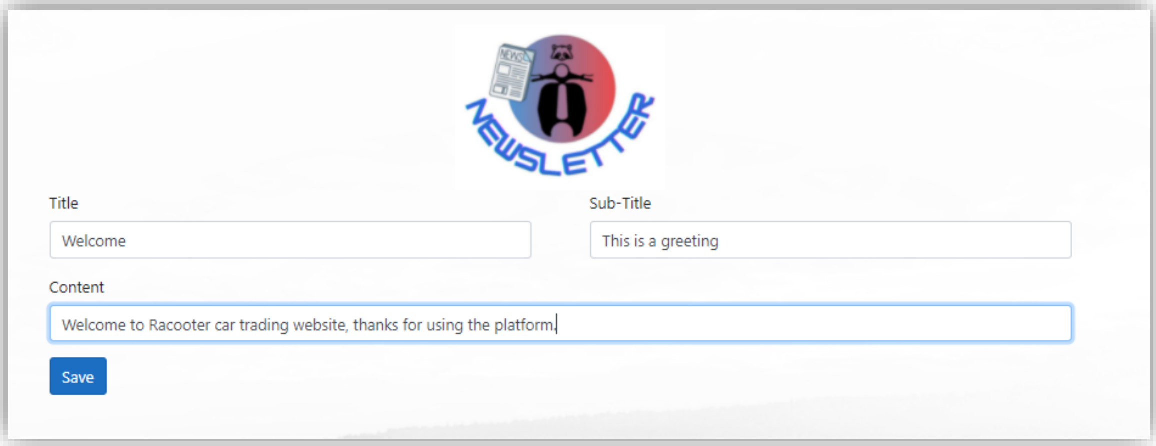


Figure 94. Newsletter demo A

Then in the interface will appear our fresh entry, because are displayed from the NewsPosts model:

Figure 95. Newsletter demo B

As we can see all the information are added, including the timestamp displayed in the right-down corner of the news post. All the posts are ordered desceneding by the property CreatedDate, so the latest will be always on top.

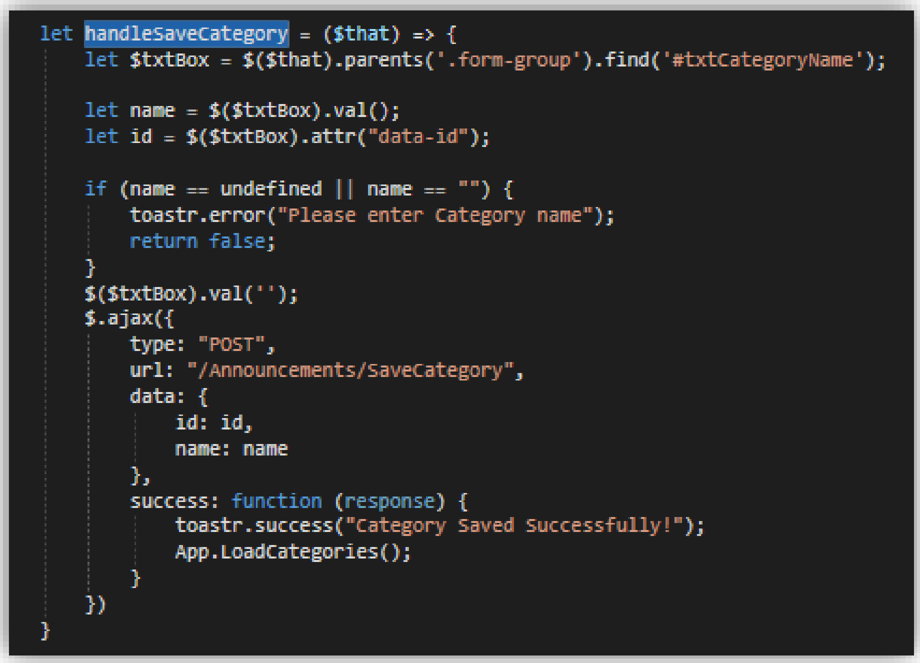
Now let’s try to hit the trashcan icon in order to delete the entry and we can notice that the entry is deleted and the next one will take its place to be the first:

Figure 96. Newsletter demo C

4.3.3 Category feature

In this section will be presented the feature that allows the admin to add a new category in order that the users can use the new category on a fresh added announcement, also there is a possibility to delete an existing category using the red button from the *Categories* view.

When the admin is pressing the „Save" button on the interface after providing a category name, it will call a method on javascript:



```
let handleSaveCategory = ($that) => {
    let $txtBox = $($that).parents('.form-group').find('#txtCategoryName');

    let name = $($txtBox).val();
    let id = $($txtBox).attr("data-id");

    if (name == undefined || name == "") {
        toastr.error("Please enter Category name");
        return false;
    }
    $($txtBox).val('');
    $.ajax({
        type: "POST",
        url: "/Announcements/SaveCategory",
        data: {
            id: id,
            name: name
        },
        success: function (response) {
            toastr.success("Category Saved Successfully!");
            App.LoadCategories();
        }
    })
}
```

Figure 97. SaveCategory js

It will retrieve from the input boxes the name, and will be assigned an id that is incresead with every add, like a counter. We can observe that are also some checks to see if the admin has really typed a category name before hitting the save button.

Then it will call the method *SaveCategory* on the *Announcements* controller that will deal with the saving in a method on the repository.

As parameter will take the category name and the id, because can be helpful to know the current category id in order to delete or update it. A new *Category* object is created and in the field *Name* is assigned the value defined by the admin, then the changes are saved on the *Categories* entity.

```
public async Task SaveCategoryAsync(int id, string name)
{
    var category = new Category();
    if (id > 0)
    {
        category = await _context.Categories.FindAsync(id);
        category.Name = name;
        _context.Entry(category).State = EntityState.Modified;
    }
    else
    {
        category.Name = name;
        _context.Categories.Add(category);
    }
}
```

Figure 98. SaveCategoryAsync repository

And the delete method on the repository is the following:

```
public async Task DeleteCategory(int id)
{
    var category = await _context.Categories.FindAsync(id);
    _context.Categories.Remove(category);
    //await _context.SaveChangesAsync();
}
```

Figure 99. DeleteCategory repository

Basically, is using the provided id of the current category entry and extract the model object from the Categories table and then it will remove the whole entry by using the Remove method on the base repository.

4.3.3.1 Demo

In this section will be demonstrated the adding, removal and displaying of the categories. Let's assume we want to add the following category in the app:

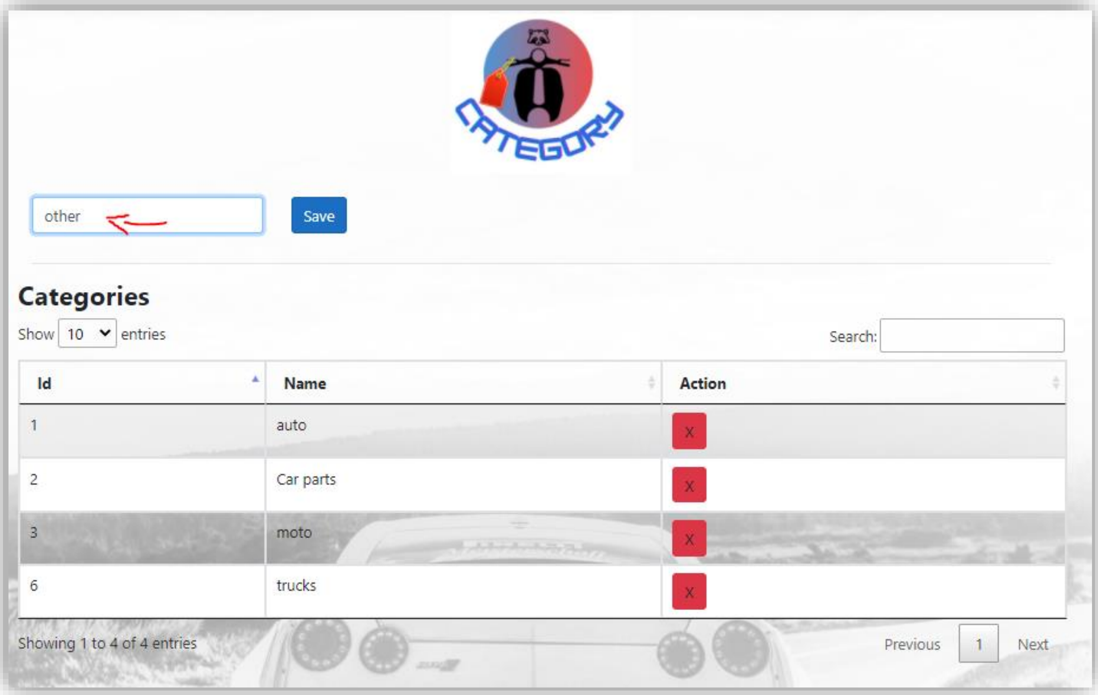


Figure 100. Category demo A

Then we save it and it should be displayed in the table:

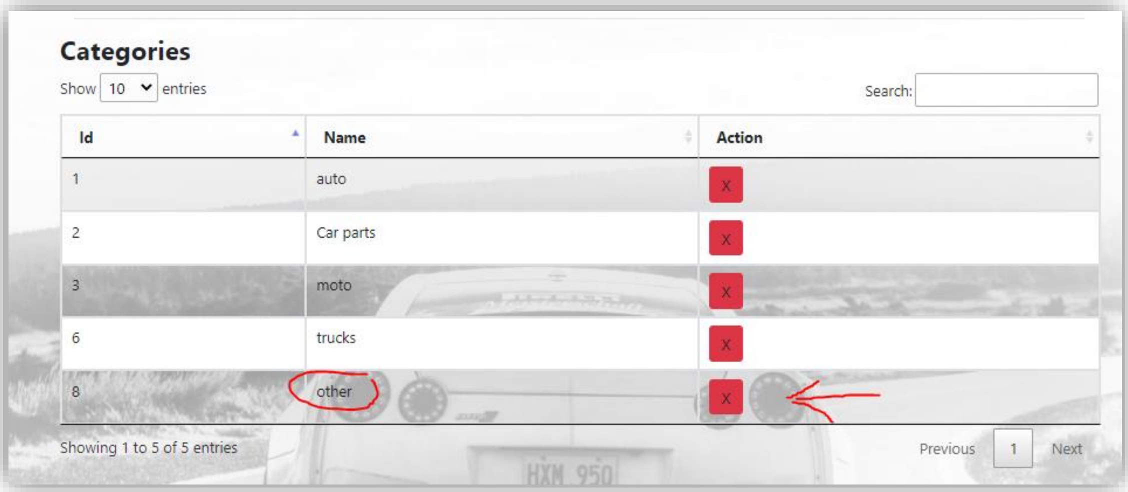


Figure 101. Category demo B

The id that it received is 8 because when delete actions are performed the id counter will go only in increasing order so the next number in order is 8 in our case, because the missing numbers werer deleted. In order to perform a delete the red button is pressed and the entry should be deleted:

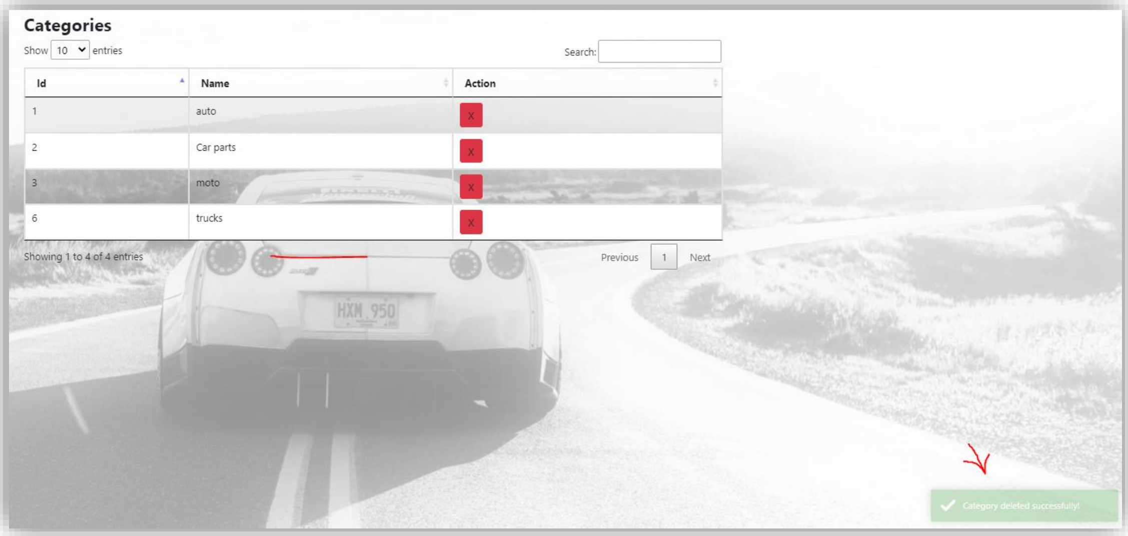


Figure 102. Category demo C

As we can notice the entry is deleted from the table and also the user is notified using a green bootstrap alert in the bottom-right corner of the screen.

4.3.4 Users panel

In this category will be presented the user panel features that consists in visualization, updating the users information and deleting a user account, all these actions are performed only by the admin in users panel.

4.3.4.1 Display

The users are displayed using the following method on the UsersController:

```
public async Task<IActionResult> Index()
{
    var data = await _announcementService.GetAllUsers();
    return View(data);
}
```

Figure 103. GetAllUsers controller

Then, using the service will call the method GetAllUsers(), and then on the repository will be implemented the function that will extract all the users information form the database, with all their properties. In the table will be displayed the name, email, if is reported, which user reported him and if is blocked from posting an announcement.

```
public async Task<List<UserDto>> GetAllUsers()
{
    var data = await _context.Users.Select(x => new UserDto
    {
        Email = x.Email,
        FullName = x.FullName,
        id = x.Id,
        IsBlockFromPost = x.IsBlockFromPost ?? false,
        IsReportedForBlock = x.IsReportedForBlock ?? false,
        ReportedBy = x.ReportedBy,
        Role = ""
    }).ToListAsync();

    var reportedBy_IDs = await _context.Users.Where(x => !string.IsNullOrEmpty(x.ReportedBy)).Select(r => r.ReportedBy).ToListAsync();

    var reportedByUsers = await _context.Users.Where(x => reportedBy_IDs.Contains(x.Id)).Select(x => new
    {
        UserId = x.Id,
        Name=x.FullName
    }).ToListAsync();
    foreach (var item in data)
    {
        if (!string.IsNullOrEmpty(item.ReportedBy))
        {
            item.ReportedBy = reportedByUsers.Where(x => x.UserId == item.ReportedBy).Select(r => r.Name).FirstOrDefault();
        }
    }
    return data;
}
```

Figure 104. GetAllUsers repository

As we can observe we are creating a new *UserDto* based on all the users from the *Users* table, then is selecting the id of the users that reported a user, then will be created an anonymous type object where will be grouped the user id and his name. Then we can iterate through the data containing all the users and we assign in the field *ReportedBy* the correspondent user based on the id, and we return a list of UserDto object.

The demonstration regarding the visualization of all users will be presented in the next section along with the update feature.

4.3.4.2 Update

This feature will give the possibility to update information about the user, like the display name of that user on every announcement, the status on if is reported, and the ban status, that will interdict to post new announcements.

When the edit button is pressed then the method *UpdateUser* on *UsersController* will be triggered. The id of the speciffic user is provided in order to select the correct one when updating its account and fill automatically the update fields with the existing values, so we need to get that specific user.

The method on repository that is getting the user by is called *GetUser* and looks like this:

```
public async Task<UserDto> GetUser(string Id)
{
    return await _context.Users.Where(x => x.Id == Id).Select(x => new UserDto
    {
        Email = x.Email,
        FullName = x.FullName,
        id = x.Id,
        IsBlockFromPost = x.IsBlockFromPost ?? false,
        IsReportedForBlock = x.IsReportedForBlock ?? false,
        ReportedBy = x.ReportedBy,
        Role = ""
    }).FirstOrDefaultAsync();
}
```

Figure 105. GetUser repository

We simply make a query on the *Users* table and extract in a new *UserDto* object the information about this particular user, based on his id.

And when the admin is clicking the save button from the update user form will be called an *HttpPost* method that will update the user information.

```
public async Task SaveUser(UserDto model)
{
    var user = await _context.Users.FindAsync(model.id);
    if (user != null)
    {
        user.FullName = model.FullName;
        user.IsBlockFromPost = model.IsBlockFromPost;
        user.IsReportedForBlock = model.IsReportedForBlock;
    }
}
```

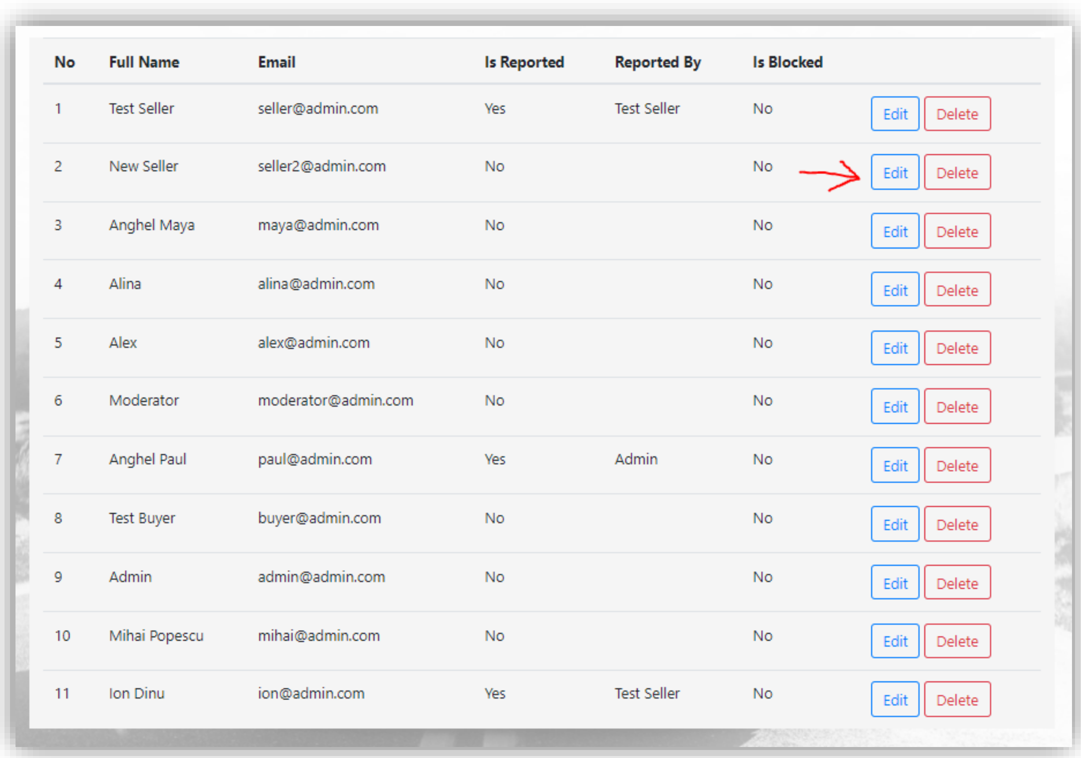
Figure 106. SaveUser repository

Basically we are retrieving from the database an *ApplicationUser* object and we do a check if the extraction is ok and the object is not null, then we just update the fields from the provided model, filled with admin inputs.

4.3.4.2.1 Demo

In this section will be demonstrated the display of all application users, including the admin and moderator, also the update name of the user. We have the possibility to update also the email but is better to let the email alone because is used for credentials and authentification.

Let’s check the admin panel and update the selected user name:



No	Full Name	Email	Is Reported	Reported By	Is Blocked	
1	Test Seller	seller@admin.com	Yes	Test Seller	No	Edit Delete
2	New Seller	seller2@admin.com	No		No	Edit Delete
3	Anghel Maya	maya@admin.com	No		No	Edit Delete
4	Alina	alina@admin.com	No		No	Edit Delete
5	Alex	alex@admin.com	No		No	Edit Delete
6	Moderator	moderator@admin.com	No		No	Edit Delete
7	Anghel Paul	paul@admin.com	Yes	Admin	No	Edit Delete
8	Test Buyer	buyer@admin.com	No		No	Edit Delete
9	Admin	admin@admin.com	No		No	Edit Delete
10	Mihai Popescu	mihai@admin.com	No		No	Edit Delete
11	Ion Dinu	ion@admin.com	Yes	Test Seller	No	Edit Delete

Figure 107. User panel demo A

Let’s modify the Full Name attribute, and block him for posting announcements:

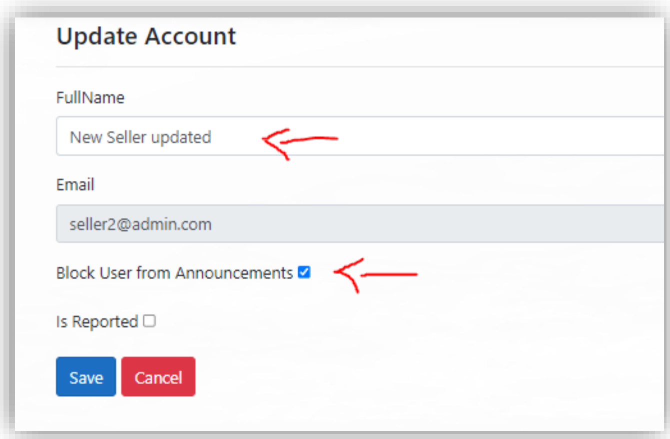


Figure 108. User panel demo B

The updated user will look like this in the users panel:

No	Full Name	Email	Is Reported	Reported By	Is Blocked	
1	Test Seller	seller@admin.com	Yes	Test Seller	No	Edit Delete
2	New Seller updated	seller2@admin.com	No		Yes	Edit Delete
3	Anghel Maya	maya@admin.com	No		No	Edit Delete

Figure 109. User panel demo C

Let's try to publish an announcement using that user account and see if we can post something anymore. When we try to access the „New Announcement" feature the following message is displayed:

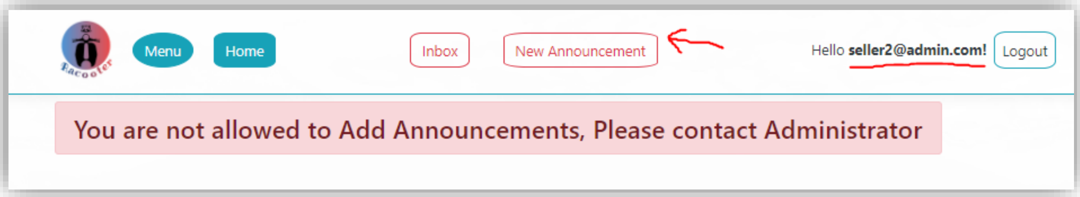


Figure 110. User panel demo D

4.3.4.3 Delete

In this section will be presented the feature of deleting users account by the admin using the *DeleteUser* method from the *UsersController*. The main method that is removing the users account from the database is present on the repository and looks like this:

```
public async Task DeleteUser(string id)
{
    var user = await _context.Users.FindAsync(id);

    var userAnnouncementsId = _context.Announcements.Where(a => a.SellerInfo.Id == user.Id).Select(x => x.AnnouncementId).ToList();
    if (userAnnouncementsId != null)
    {
        foreach(var item in userAnnouncementsId)
        {
            await DeleteAnnouncement(item);
        }
    }
    _context.Users.Remove(user);

}
```

Figure 111. DeleteUser repository

We need of course the id of the selected user that is passed by parameter when clicking the delete button on the interface. Then we should extract all the announcements id of this user, and will check if the user has some posted announcements extracted, in order to delete them first. Otherwise, won't work to delete the user because his id will be on the posted announcements in the field *SellerInfo* and an exception will be thrown.

Then we iterate through the announcementsId list and apply the *DeleteAnnouncement*() method on each of items, that will permanently remove that particular announcement at one iteration, along with other objects related with the announcement, like specification and images.

After all its announcements are deleted, then the user account can be safely deleted as well, using the context and *Remove()* method.

4.3.4.3.1 Demo

Let's try to delete this user:

No	Full Name	Email	Is Reported	Reported By	Is Blocked	
1	Test Seller	seller@admin.com	Yes	Test Seller	No	Edit Delete
2	New Seller updated	seller2@admin.com	No		Yes	Edit Delete
3	Anghel Maya	maya@admin.com	No		No	Edit Delete

Figure 112. DeleteUser demo A

Then the Users panel will be updated:

No	Full Name	Email	Is Reported	Reported By	Is Blocked	
1	Test Seller	seller@admin.com	Yes	Test Seller	No	Edit Delete
2	Anghel Maya	maya@admin.com	No		No	Edit Delete
3	Alina	alina@admin.com	No		No	Edit Delete

Figure 113. DeleteUser demo B

We clearly can see that this user is gone, now let's try to login using the deleted user credentials in order to see if the user still exists:

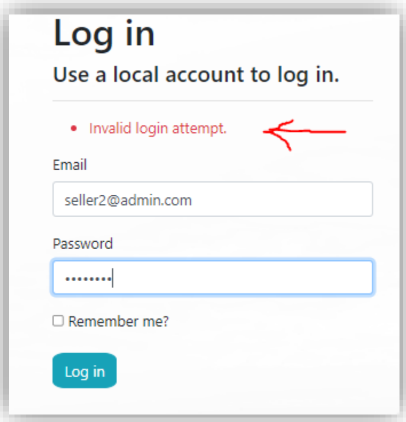


Figure 114. DeleteUser demo C

We are giving the correct credentials but the application is not recognizing this user anymore, so it was permanently deleted from the database, is working as expected. Now all the announcements posted by this user are deleted as well.

4.3.5 Chat system

In this section will be presented in detail the chat between users features. The implementation is based on the functional requirements, application architecture and user interface implementation presented in the previous chapters. In the User Interface implementation chapter, we noticed how the messages and conversations should be displayed, which views are responsible for this, and some front-end js code responsible with reloading the message presented in the figure: **Figure 43. ReloadMessages method and call.**

Further I will present the back-end part of the chat system, with code snippets and explanation, along with a working example.

So, the chat system is working based on creating and managing entries in the Messages table, that contains a recipientId, represented by the receiver of the conversation and the CreatedBy property is represented by the current user id. There is also the text of the messages and a timestamp when the messages were created.

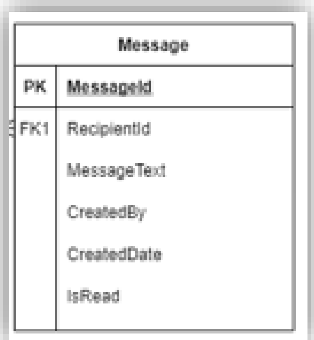


Figure 115. Messages table

When the user is pressing in the interface the Inbox button, a method Messages will be triggered:

```
<a class="dropdown-item" href="/Announcements/Messages">Inbox</a>
```

Figure 116. Messages trigger no id

This is an HttpGet method that will receive as parameter an id, because when we contact the seller, then the id to the recipient is provided, so the current user will be redirected to that particular conversation and a fisrt message will be sent automatically in order to notify the recipient that this user wants to start a conversation.

This method will also load all the conversations list, found on the left of the chat visualization, basically these are the peoples whom the current user opened a conversation with them.

Below is the screenshot with *Messages* method on the controller:

```
public async Task<IActionResult> Messages(string id)
{
    var currentUserId = await GetCurrentUserId();
    await _messageService.AddFirstMessage(id, currentUserId);
    ViewBag.MessageUsers = await _messageService.GetAllUsersForMessages(currentUserId);
    ViewBag.ReceipentId = id;
    return View();
}
```

Figure 117. Messages controller

We will take a look now to the method *AddFirstMessage*(), below is the method implementation on the repository:

```
public async Task AddFirstMessage(string id, string CurrentUserId)
{
    if (!string.IsNullOrEmpty(id))
    {
        var isAlready = await _context.Messages.Where(x => x.Recipient == id && x.CreatedBy==CurrentUserId).AnyAsync();
        if (!isAlready)
        {
            //first message
            var msg = new Message();
            msg.CreatedBy = CurrentUserId;
            msg.CreatedDate = DateTime.Now;
            msg.IsRead = true;
            msg.MessageText = "I want to start a conversation...";
            msg.Recipient = id;
            _context.Messages.Add(msg);
            await _context.SaveChangesAsync();
        }
    }
}
```

Figure 118. AddFirstMessage repository

First of all there is a check made on the id, because it can be null when the user is accessing only the Inbox button, it will be loaded with a value only when the user clicks on the “Contact Seller” button because it will be redirected to that specific user using its id, as we can see on the front-end code:

```
<a class="btn btn-success" href="/Announcements/Messages?id=@Model.CreatedBy">Contact Seller</a>
```

Figure 119. Messages trigger with id

In this case the *AddFirstMessages* method will pass the check regarding if the id is not null, and will make sure to send the first message only once, so that’s why I used a variable *isAlready*.

Then a new instance of the Message model will be created providing all the information needed: the CreatedBy is actually the current user because he/she started the conversation, the CreatedDate is the time recorded at that moment, the recipient is represented by the parameter id and the MessageText will be a default “I want to start a conversation…”, that will notify the recipient that this user wants to start chatting.

The other important method called on the *Messages* method is *GetAllUsersForMessages*() that will load the conversations user lists and group in pairs of two strings the user id and its name.

Below is the function implementation on the *MessageRepository*:

```
public async Task<List<KeyValuePair<string, string>>> GetAllUsersForMessages(string userId)
{
    List<KeyValuePair<string, string>> list = new List<KeyValuePair<string, string>>();

    var userIds = await _context.Messages.Where(x => x.CreatedBy == userId).Select(x => x.Recipient).Distinct().ToListAsync();
    var senderUserIds = await _context.Messages.Where(x => x.Recipient == userId).Select(x => x.CreatedBy).Distinct().ToListAsync();

    userIds.AddRange(senderUserIds);
    userIds = userIds.Where(x => x != userId).Distinct().ToList();

    var users = await _context.Users.Where(x => userIds.Contains(x.Id)).Select(x => new
    {
        UserId = x.Id,
        Name = x.UserName
    }).ToListAsync();

    foreach (var item in users)
    {
        list.Add(new KeyValuePair<string, string>(item.UserId, item.Name));
    }
    return list;
}
```

Figure 120. GetAllUsersForMessages repository

As we can notice, the method will extract a list of userId representing all the users which had a conversation with the current logged user. As I mentioned, we need to upload also the conversations that were not initiated by the current user, so in this case the current user was the recipient and the initiator of the conversation is the other user.

Then the two extracted lists are merged together, using the *AddRange* method that will add the elements of the second list to the end of the first one.

After we are making sure the current user will not appear in the conversation list because you can’t chat with yourself. Then will be created objects extracted from the Users table that will contain those ids from the userIds list, it will be created KeyValuePair using the UserId and its Name that will be displayed in the front-end.

The Messages.cshtml view has a section marked with the id *#UserMessagesContainer* that will be the div where all the messages between two users will be displayed. Below is the js method that will fill that div dynamically after a call on the controller will be made using the method *UserMessages*:

```
let handleLoadUserMessages = (userId) => {

    $('.chat_list').removeClass('active_chat');
    $('div[data-id="' + userId + '"]').parents('.chat_list').addClass('active_chat');
    $('#selectedUserId').val(userId);
    $.ajax({
        type: "GET",
        url: "/Announcements/UserMessages",
        data: {
            userId: userId
        },
        success: function (response) {
            $('#UserMessagesContainer').html(response);
        }
    })
}
```

Figure 121. LoadUserMessages js

The method called on the controller will be detailed below:

```
public async Task<IActionResult> UserMessages(string userId)
{
    var currentUserId = await GetCurrentUserId();
    var list = await _messageService.GetUserMessages(userId, currentUserId);
    ViewBag.CurrentUserId = currentUserId;
    return PartialView(list);
}
```

Figure 122. UserMessages controller

It will get the current user if using the ASP Identity system and then will retrieve all the messages between two users, using their unique id. The implementation of the *GetUserMessages* on the repository will be displayed below:

```
public async Task<List<MessageDto>> GetUserMessages(string senderId, string receiverId)
{
    List<MessageDto> list = new List<MessageDto>();
    list = await _context.Messages.
        Where(x => (x.CreatedBy == senderId && x.Recipient == receiverId) || (x.Recipient == senderId && x.CreatedBy == receiverId) && !string.IsNullOrEmpty(x.MessageText))
        .Select(x => new MessageDto
        {
            CreatedBy = x.CreatedBy,
            Recipient = x.Recipient,
            CreatedDate = x.CreatedDate,
            MessageText = x.MessageText
        }).ToListAsync();
    return list;
}
```

Figure 123. GetUserMessages repository

The method will take as parameter the sender Id and receiver Id. Using the *Messages* table, we extract all the entries, respecting the condition that the *senderId* is equal with *CreatedBy* field and the *Recipient* property is equal with *receiverId*, but in practice both of the users can be sender and receivers, and when we load the messages between two users, we should display also the conversations that not the current logged user has started.

So, we need to add a second check to verify if the *Recipient* field is equal with the *senderId* this time, and the *CreatedBy* property is equal with the *receiverId*. The current user is represented by the variable *senderId*. In this case we are covered in both ways, so it doesn't matter who started the conversation between the two of them, we need to display that is there.

Then a new MessageDto object is created and filled with correspondent values, in order to display it on the front-end in the right side of the chat visualization, where the user messages will be present.

In the view, all the user's conversations are displayed using the *ViewBag* filled with data in **Figure 117** as we can observe on the View code:

```
@if (ViewBag.MessageUsers != null)
{
    foreach (var item in ViewBag.MessageUsers as List<KeyValuePair<string, string>>)
    {
        <div class="chat_list">
            <div class="chat_people" data-id="@item.Key" onclick="App.LoadUserMessages('@item.Key')">
                <div class="chat_img"> <img src="https://ptetutorials.com/images/user-profile.png" alt
                <div class="chat_ib">
                    <h5>@item.Value <span class="chat_date"></span></h5>
                    <p>
                        Click to see messages.
                    </p>
                </div>
            </div>
        </div>}
}
```

Figure 124. Users Conversations display

On the chat system we have other main feature that is represented by the possibility to actually send a message and save it in the database. Each time the send button is clicked or the enter key is pressed, the method *handleSaveMessages()* will be triggered on js:

```
let handleSaveMessage = () => {

    let msgTxt = $('#txtBoxMessage').val();
    if (msgTxt == "" || msgTxt.trim() == "" || msgTxt == undefined) {
        toastr.error("Please write a message first!");
        return false;
    }

    let receiverId = $('.active_chat').find('.chat_people').attr("data-id");

    $.ajax({
        type: "POST",
        url: "/Announcements/SaveMessage",
        data: {
            MessageText: msgTxt,
            receiverId: receiverId
        },
        success: function (response) {
            $('#txtBoxMessage').val("");
            App.LoadUserMessages(receiverId);
        }
    })
}
```

Figure 125. SaveMessages js

As we can tell a method will be called on the controller that will actually make the saving possible. The implementation of this method will be attached below and explained:

```
[HttpPost]
public async Task<IActionResult> SaveMessage(string MessageText, string receiverId)
{
    await _messageService.SaveMessage(MessageText, receiverId, await GetCurrentUserId());
    return Json(true);
}
```

Figure 126. SaveMessages controller

Its an HttpPost method and will call the service and then the repository where the handling with database occurs, below will be presented the method on the repository.

The method is taking as parameters the actual text that was provided in the chat box by the sender input and the receiver id, because we should fill the RecipientId property of a message.

```
public async Task SaveMessage(string messageText, string receiverId, string currentUserId)
{
    var msg = new Message()
    {
        CreatedBy = currentUserId,
        CreatedDate = DateTime.Now,
        IsRead = true,
        MessageText = messageText,
        Recipient = receiverId
    };
    _context.Messages.Add(msg);

}
```

Figure 127. SaveMessages repository

In this method we just create a new instance of *Message* model and fill it with all the information provided and then we add the entity as an entry in the *Messages* table on the database.

4.3.5.1 Demo

In this section it will be demonstrated the chat system feature, consisting in a live chat between two user’s accounts. We will be logged as one user and using an incognito browser page we are able to login with the second user and see how they are receiving and sending messages. Let’s contact this seller, logged as the moderator:

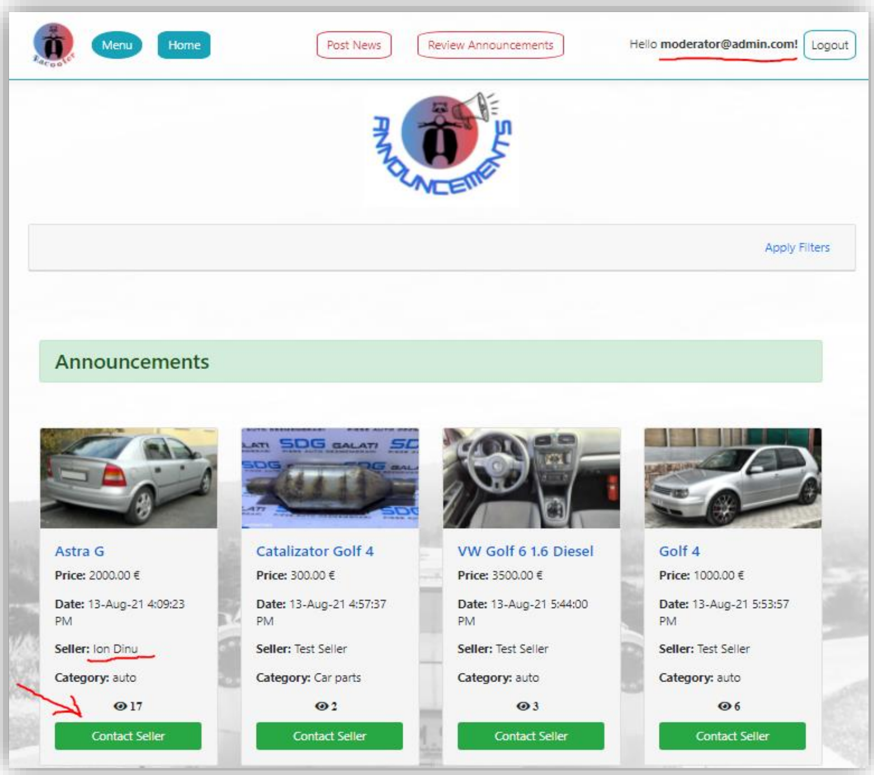


Figure 128. Chat system demo A

Now let's login with "Ion Dinu" credentials and open the inbox because it should receive the first initiation message from the moderator:

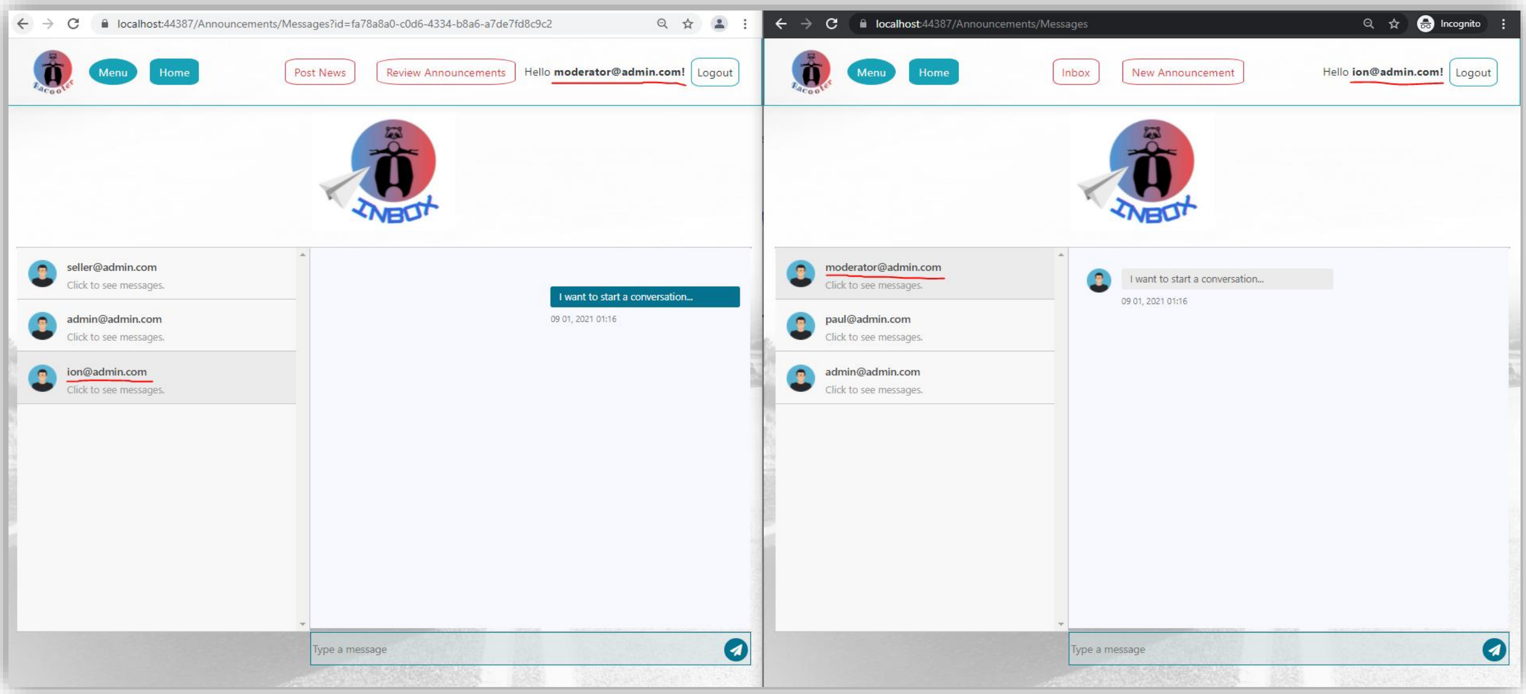


Figure 129. Chat system demo B

That's good, the reciever Ion has a new conversation and received the first auto generated message from the moderator, now let's try to send more messages between then and try to simulate a conversation:

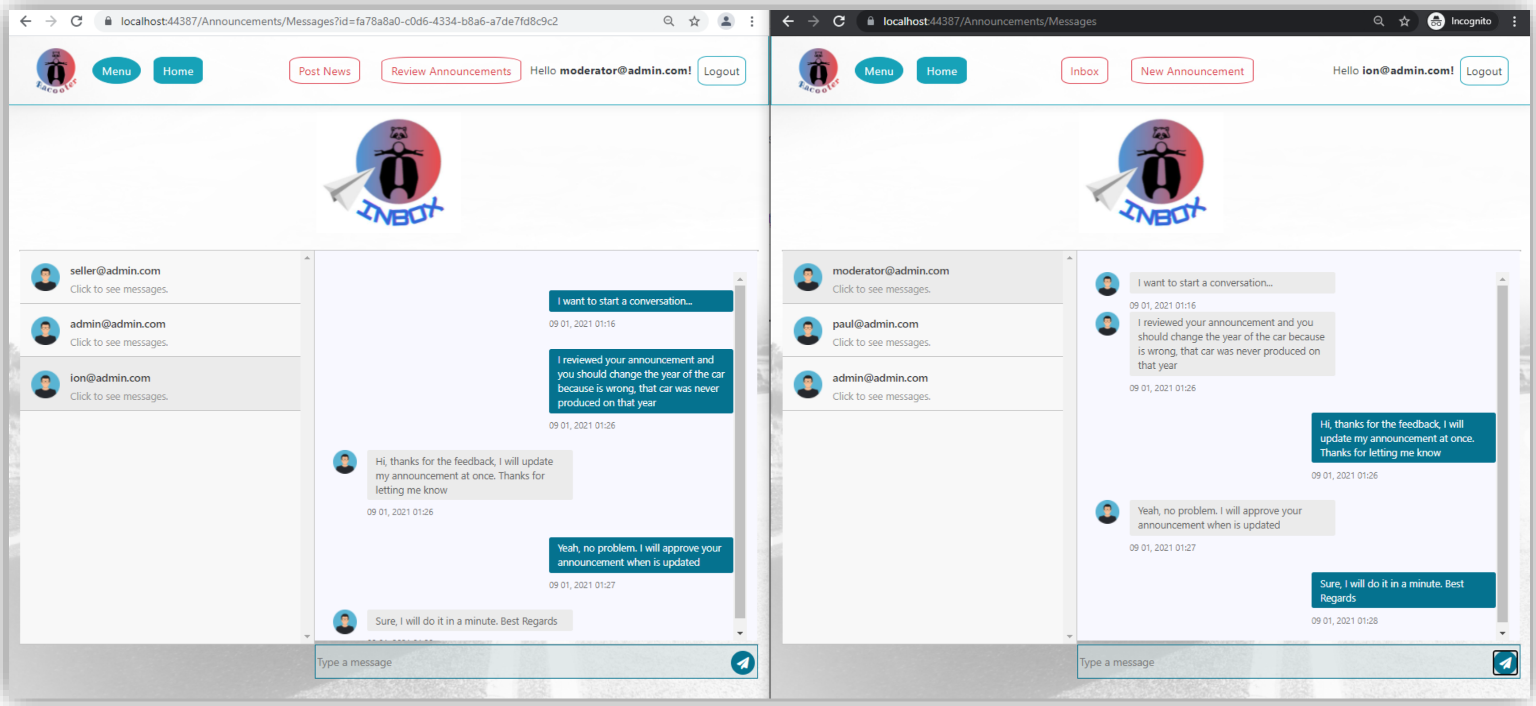


Figure 130. Chat system demo C

That is looking great, and keep it in mind that the messages appear almost instantly without refreshing the page so both users are enggaged in the online chat system.

4.3.6 Recommender system

In this section will be presented the possibility of a logged user to get recommended announcements on the homepage, based on his filtering and searching history that represents users interests.

The table where the filters are saved is called *SearchFilters* and looks like this:

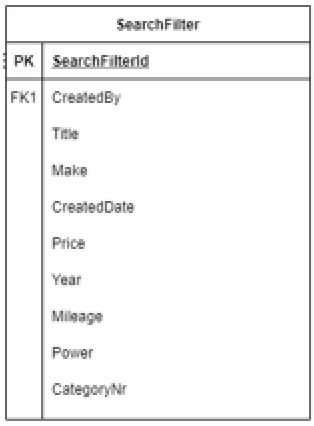


Figure 131. SearchFilter table

As we can observe we are saving the filters by current user, that will fill the *CreatedBy* property.

The recommended announcements will be displayed on the home page, more precise on the *_Home.cshtml* view in a different section called „Recommended for You”.

The main method that is in charge of displaying and filtering announcements on the home page is *GetFilteredAnnouncements* that we saw at the Filters implementation chapter, but this time I will explain other parts of this method that is containing the logic for the recommender system.

Everytime a user is filtering the announcements, parameters are saved in the database table *SearchFilter* using the method *SaveFilterByUser*. Below is a section of GetFilteredAnnouncements method on the repository that will be explained:

```
public async Task<List<AnnouncementDto>> GetFiltererdAnnouncements(AnnouncementFilter filter, string CurrentUserId)
{
    bool isFilterApplied = false;
    if (!string.IsNullOrEmpty(CurrentUserId))
    {
        isFilterApplied = await SaveFilterByUser(filter, CurrentUserId);
    }

    var query = (from x in _context.Announcements
                 join y in _context.Specifications on x.AnnouncementId equals y.AnnouncementId
                 where x.IsApprovedByAdmin == true
                 select new
                 {
                     Announcement = x,
                     Specification = y
                 });
```

Figure 132. GetFilteredAnnouncements recommender section A

As we can see, the purpose of variable *isFilterApplied* is to let the system know if we are not applying the filters, because we can't mix filtered announcements with recommended announcements. So, is better to recommend announcements when the filters are empty in order to work properly.

The method *SaveFilterByUser* should save the filter parameters like Category, Make, Model, Power, Price, Title and Year on every user search and filter. This method is implemented below:

```
public async Task<bool> SaveFilterByUser(AnnouncementFilter filter, string CurrentUserId)
{
    var fltr = new SearchFilter();
    bool AnyFieldHaveValue = false;
    if (filter.Category.HasValue)
    {
        AnyFieldHaveValue = true;
        fltr.CategoryId = filter.Category;
    }
    if (!string.IsNullOrEmpty(filter.Make))
    {
        AnyFieldHaveValue = true;
        fltr.Make = filter.Make;
    }
    if (filter.Mileage.HasValue)
    {
        AnyFieldHaveValue = true;
        fltr.Mileage = filter.Mileage;
    }
    if (!string.IsNullOrEmpty(filter.Model))
    {
        AnyFieldHaveValue = true;
        fltr.Model = filter.Model;
    }
    if (filter.Power.HasValue)
    {
        AnyFieldHaveValue = true;
        fltr.Power = filter.Power;
    }
    if (filter.Price.HasValue)
    {
        AnyFieldHaveValue = true;
        fltr.Price = filter.Price;
    }
    if (!string.IsNullOrEmpty(filter.Title))
    {
        AnyFieldHaveValue = true;
        fltr.Title = filter.Title;
    }
    if (filter.Year.HasValue)
    {
        AnyFieldHaveValue = true;
        fltr.Year = filter.Year;
    }
    if (AnyFieldHaveValue)
    {
        fltr.CreatedBy = CurrentUserId;
        fltr.CreatedDate = DateTime.Now;
        _context.SearchFilters.Add(fltr);
        await _context.SaveChangesAsync();
    }

    return AnyFieldHaveValue;
```

Figure 133. SaveFilterByUser repository

The method parameters are an *AnnouncementFilter* object that contains all the fields described above and also the current user id, in order to save the filters parameters to the logged user.

A new instance of *SearchFilter* model is crated and is filled with the correspondent values from the filter object only if has a value, because the user can complete only few filters not all of them, so we should know which of them are filled and need to be saved.

I check with a variable called *AnyFieldHaveValue* in order to know the system if we have at least one filter parameter that need to be saved, otherwise we don’t need to save anything.

The next step is to extract all the existing approved announcements that are already displayed in the home page, because the system should recommend from those announcements based on a logic presented below in this section of the *GetFilteredAnnouncement* method:

```
if (!isFilterApplied && !string.IsNullOrEmpty(CurrentUserId))
{
    //search engine logic started from here
    List<Guid> TempIDs = new List<Guid>();

    List<AnnouncementDto> newFilteredList = new List<AnnouncementDto>();
    List<AnnouncementDto> tempFilteredList = new List<AnnouncementDto>();
    char FilterColumn = 'A';
    bool isDataAdded = false;
    int tempFilterInt = 0;

    //title filter logic
    var maxOccuredTitle = GetColumnOccurences("Title", CurrentUserId).OrderByDescending(x => x.Count).Select(r => r.ColumnName).FirstOrDefault();
    if (!string.IsNullOrEmpty(maxOccuredTitle))
    {
        tempFilteredList = data.Where(x => x.Title.ToLower().Contains(maxOccuredTitle.ToLower())).ToList();
        foreach (var item in tempFilteredList)
        {
            if (!TempIDs.Contains(item.AnnouncementId))
            {
                item.FilterColumn = FilterColumn;
                isDataAdded = true;
            }
        }
        if (isDataAdded)
        {
            FilterColumn = (Char)(Convert.ToUInt16(FilterColumn) + 1);
            isDataAdded = false;
        }
        TempIDs.AddRange(tempFilteredList.Select(x => x.AnnouncementId).ToList());
        newFilteredList.AddRange(tempFilteredList);
        tempFilteredList = new List<AnnouncementDto>();
    }
    //title filter logic end
```

Figure 134. GetFilteredAnnouncements recommender section B

In this section is starting the recommender system logic. First of all we need to check the number of occurences for each filter parameter stored in the *SearchFilter* table, in order to know which are the most searched terms for Title in our example in the code above.

The method that is computing the number of occurence of an entry in a specific column inside *SearchFilter* table is called *GetColumnOccurences* presented below:

```
private List<ResponseDto> GetColumnOccurences(string ColumnName, string CurrentUserId)
{
    ColumnName = ColumnName.Trim();
    string _query = "SELECT " + ColumnName + " as ColumnName, COUNT(" + ColumnName + ") AS Count FROM SearchFilters WHERE CreatedBy='" + CurrentUserId + "' GROUP BY " + ColumnName + " ORDER BY Count DESC";
    var entities = new List<ResponseDto>();
    using (var command = _context.Database.GetDbConnection().CreateCommand())
    {
        command.CommandText = _query;
        command.CommandType = CommandType.Text;
        context.Database.OpenConnection();
        using (var result = command.ExecuteReader())
        {
            while (result.Read())
            {
                entities.Add(new ResponseDto
                {
                    ColumnName = result["ColumnName"].ToString(),
                    Count = result["Count"] != null ? Convert.ToInt32(result["Count"]) : 0
                });
            }
        }
    }
    return entities;
}
```

Figure 135. GetColumnOccurences repository

I am using an interogation to extract the column name meaning the actual values under the *Title* column in our case, and also the count for its occurence in the table. We are doing this procedure for every entry from a particular column inside the *SearchFilter* table.

So, we are creating response objects stored in *ResponseDto* that contains the *ColumnName* and *Count*, basically we are mapping every entry from the column „Title”, with the number of appearances in that table column. In this way we can see which keyword for *Title* was the most searched, meaning that the user is very interested in that keyword.

For example a person searched the *Title* „opel” 10 times and the keyword „mercedes” 5 times, then we should choose from the *Title* column the keyword „opel” because is the most searched one.

The method will return a list of *ResponseDto* objects that will be processed by ordering descending the result based on the count number, and extracting only the top result the most searched one.

In case all the items in the result were searched only once, then it will extract the first one because the order doesn’t matter, all of them has the same priority for the user.

Then we fill a temporary filtered list where we make a LINQ query on all the announcements extracted in the data variable, represented by all the announcements in the home page, and we take only those who contain the chosen maximum occured title.

In order to avoid repetition of announcements, we are storing the announcements Id in a list called *TempIDs* and checking every time if the id is not already present in the list in order to prevent adding the same announcements multiple times.

Then we will use another list called *newFilteredList* that will contain the temporary filtered list for the Title, and then we are refreshing the temporary list in order to take the values of the other columns, repeating the entire process as can be seen in the code below:

```
//mileage logic start
var maxMileageSearched = GetColumnOccurences("Mileage", CurrentUserId).OrderByDescending(x => x.Count).Select(r => r.ColumnName).FirstOrDefault();
if (!string.IsNullOrEmpty(maxMileageSearched))
{
    var isNumeric = int.TryParse(maxMileageSearched, out int n);
    if (isNumeric)
    {
        tempFilterInt = Convert.ToInt32(maxMileageSearched);
        tempFilteredList = data.Where(x => x.Specification.Mileage <= tempFilterInt).ToList();
    }
    foreach (var item in tempFilteredList)
    {
        if (!TempIDs.Contains(item.AnnouncementId))
        {
            item.FilterColumn = FilterColumn;
            isDataAdded = true;
        }
    }
    if (isDataAdded)
    {
        FilterColumn = (Char)(Convert.ToUInt16(FilterColumn) + 1);
        isDataAdded = false;
    }
    TempIDs.AddRange(tempFilteredList.Select(x => x.AnnouncementId).ToList());
    newFilteredList.AddRange(tempFilteredList);
    tempFilteredList = new List<AnnouncementDto>();
}
//mileage logic start end


//Power logic start
var maxPowerSearched = GetColumnOccurences("Power", CurrentUserId).OrderByDescending(x => x.Count).Select(r => r.ColumnName).FirstOrDefault();
if (!string.IsNullOrEmpty(maxPowerSearched))
{
    var isNumeric = int.TryParse(maxPowerSearched, out int n);
    if (isNumeric)
    {
        tempFilterInt = Convert.ToInt32(maxPowerSearched);
        tempFilteredList = data.Where(x => x.Specification.Power <= tempFilterInt).ToList();
    }
    foreach (var item in tempFilteredList)
    {
        if (!TempIDs.Contains(item.AnnouncementId))
        {
            item.FilterColumn = FilterColumn;
            isDataAdded = true;
        }
    }
    if (isDataAdded)
    {
        FilterColumn = (Char)(Convert.ToUInt16(FilterColumn) + 1);
        isDataAdded = false;
    }
    TempIDs.AddRange(tempFilteredList.Select(x => x.AnnouncementId).ToList());
    newFilteredList.AddRange(tempFilteredList);
    tempFilteredList = new List<AnnouncementDto>();
}
//Power logic start end

//category logic start
var maxCategoryIdSearched = GetColumnOccurences("CategoryId", CurrentUserId).OrderByDescending(x => x.Count).Select(r => r.ColumnName).FirstOrDefault();
if (!string.IsNullOrEmpty(maxCategoryIdSearched))
```

Figure 136. GetFilteredAnnouncements recommender section C

FilterColumn is initially ‚A’, and if data is filtered by *Title* it will remain ‚A’, but then if data is filtered by *Price* it will become ‚B’ , next if data will be filtered by *Model* it will become ‚C’ and so on. And on the front-end we are sorting records by this FilterColumn property in order to get first the records searched the most in proper priority order.

After all the SearchFilters columns are processed using this method, we will get in newFilteredList all the announcements that we should recommend to the user ordered descending by ht e FilterColumn.

We need also to extract the announcements images and categories related to these announcements based on their Id.

Then we fill the data object with all the required information and returning it, as we can observe in the code section below:

```
    //this will be the end process
    newFilteredList.AddRange(data.Where(x => !TempIDs.Contains(x.AnnouncementId)).ToList());
    //search engine logic end

    data = new List<AnnouncementDto>();
    data.AddRange(newFilteredList);

    data = data.OrderBy(x => x.FilterColumn).Distinct().ToList();
}



var announcementIds = data.Select(x => x.AnnouncementId).ToList();

var allImages = await _context.AnnouncementImages.Where(x => announcementIds.Contains(x.AnnouncementId.Value)).Select(x => new
{
    ImagePath = x.ImagePath,
    AnnouncementId = x.AnnouncementId
}).ToListAsync();

var categories = await _context.Categories.ToListAsync();

foreach (var item in data)
{
    item.ImagesPath = allImages.Where(x => x.AnnouncementId == item.AnnouncementId).Select(x => x.ImagePath).ToList();
    item.CategoryString = categories.Where(x => x.Id == item.Category).Select(r => r.Name).FirstOrDefault();
    item.FilterColumn = item.FilterColumn == '\0' ? Char.MinValue : item.FilterColumn;
    bool isOk = item.FilterColumn == '\0';
}

return data;
```

Figure 137. GetFilteredAnnouncements recommender section D

So, each time a user is logged and has some search filter history, will get recommended announcements, that will be updated every time he/she makes new searches or apply filters, keeping the priority based on *FilterColumn*.

4.3.6.1 Demo

In this section will be presented a demonstration of the recommender system in action, on a new registered user, so it doesn't have already something in recommendations.

We are creating this user John and let's search in the *Title* box the keyword „golf"

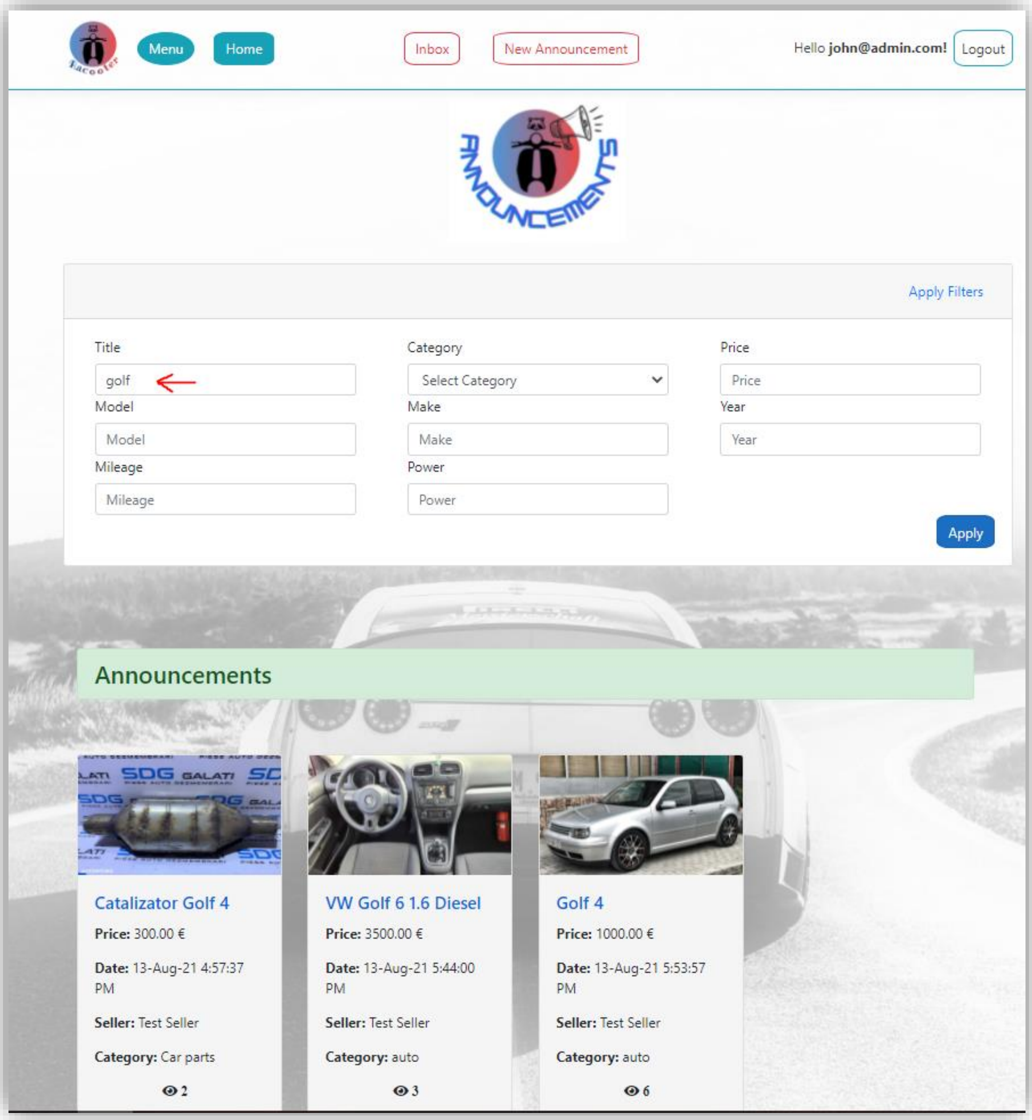


Figure 138. Recommender system Search A

Now the parameter for title „golf” is saved, next let's try to search in the title the keyword „peugeot”:

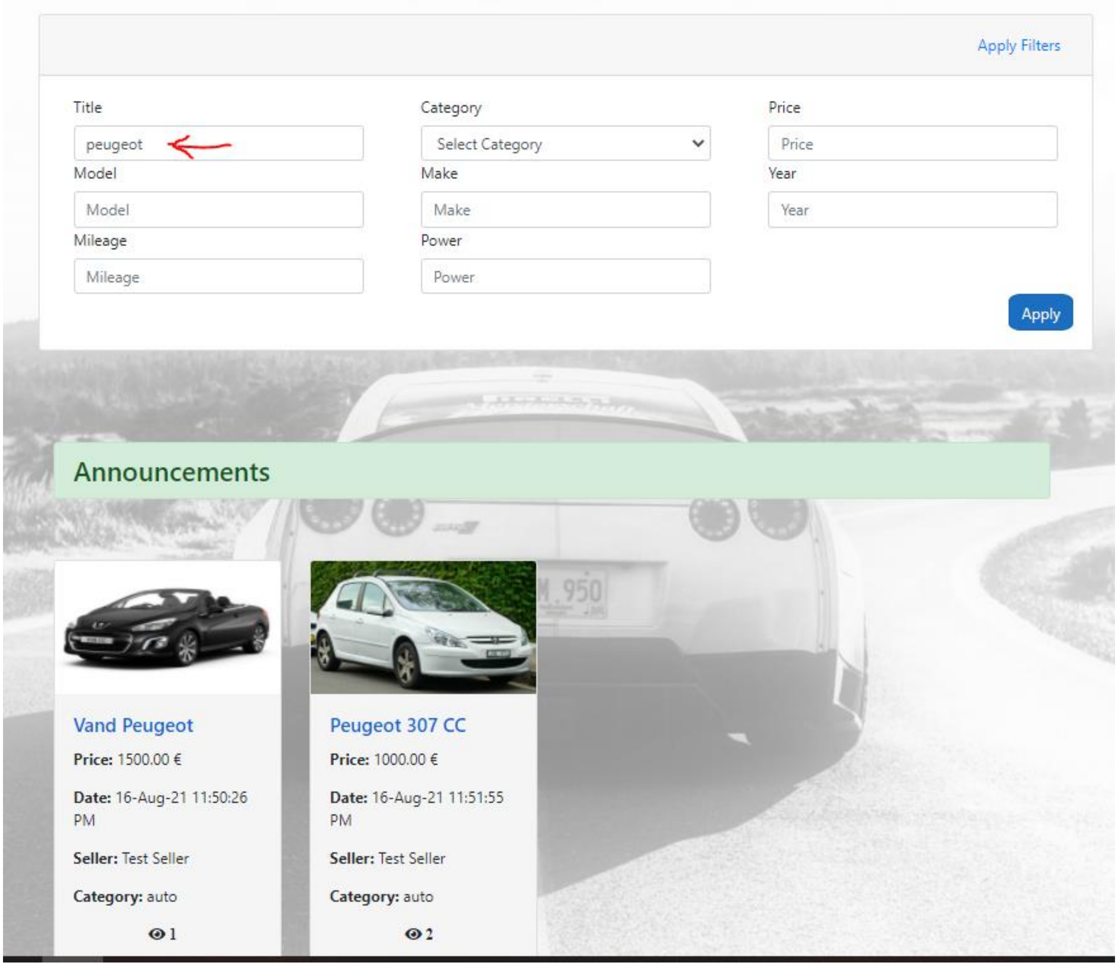


Figure 139. Recommender system Search B

Then we search again the key word „**golf**”, so the recommender system should choose **golf** over peugeot because „golf” was searched two times and „peugeot” only one time.

And we are getting recommended announcements with golf in the title as we can observe in the next screenshot:

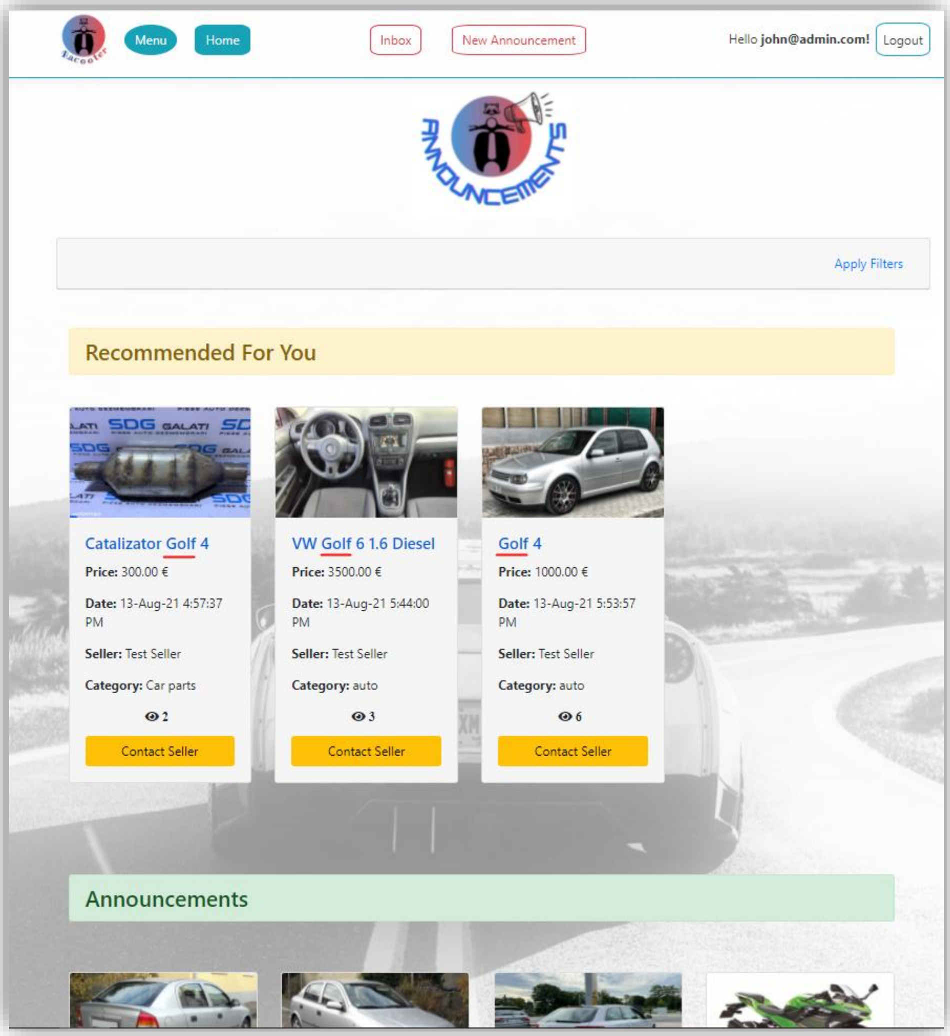


Figure 140. Recommender system results A

Then let's add more filters, let's search in the Price box the value „2000", as we can notice in the below picture:

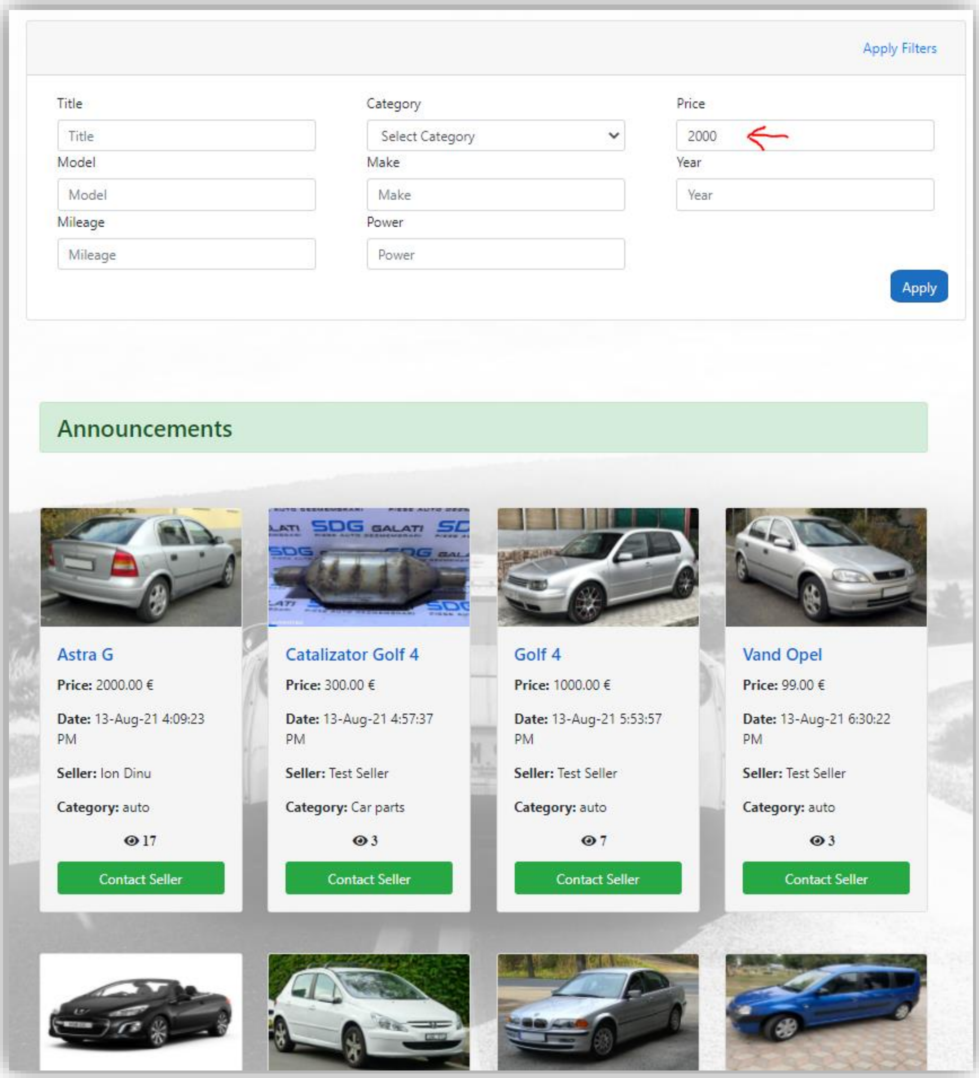


Figure 141. Recommender system Search D

In this scenario, the recommender should choose for recommending announcements with the Title „golf" as primary priority, then it should start recommending announcements with the price equals or below 2000.

We are getting the expected result as we can observe in the next screenshot:

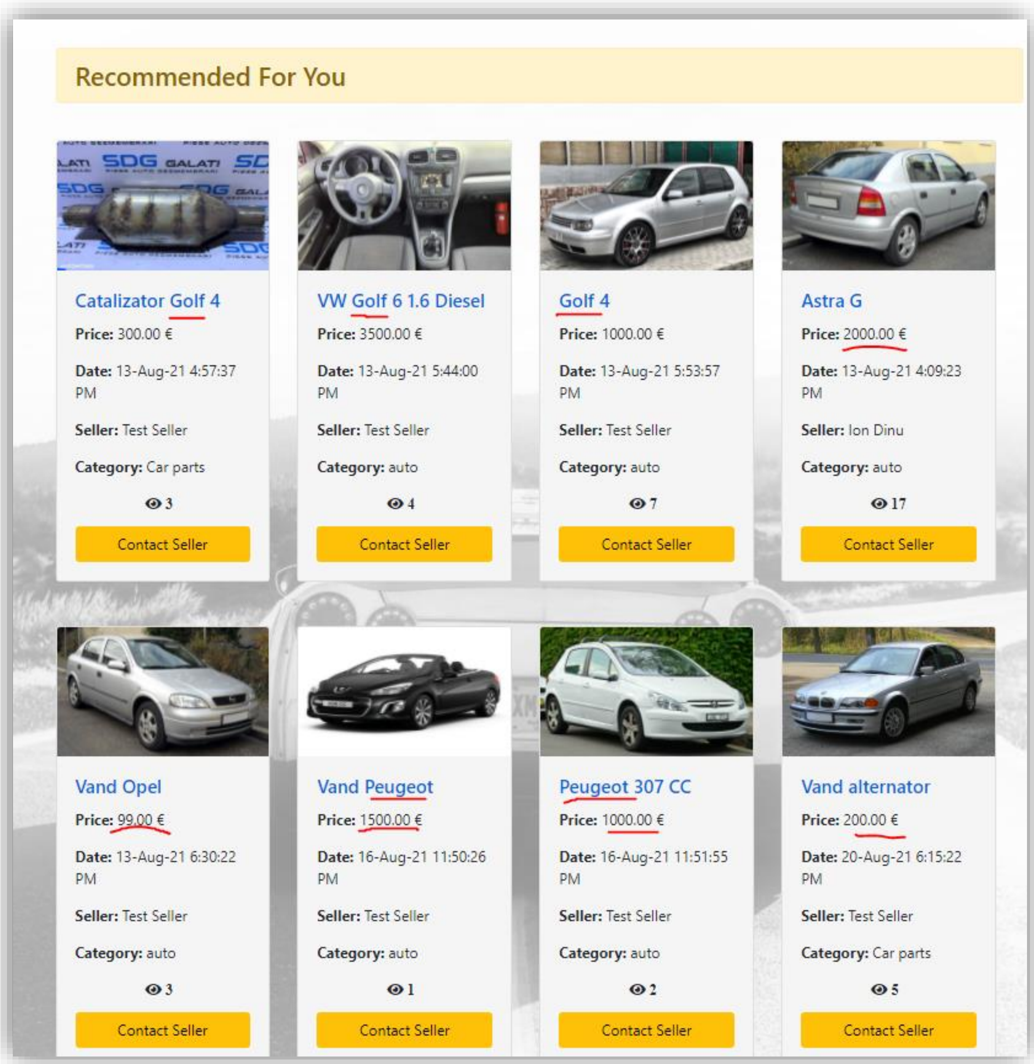


Figure 142. Recommender system results B

That is looking good, now let's try to replace the prioritization of „golf" in the Title, by searching three times the keyword „**opel**"

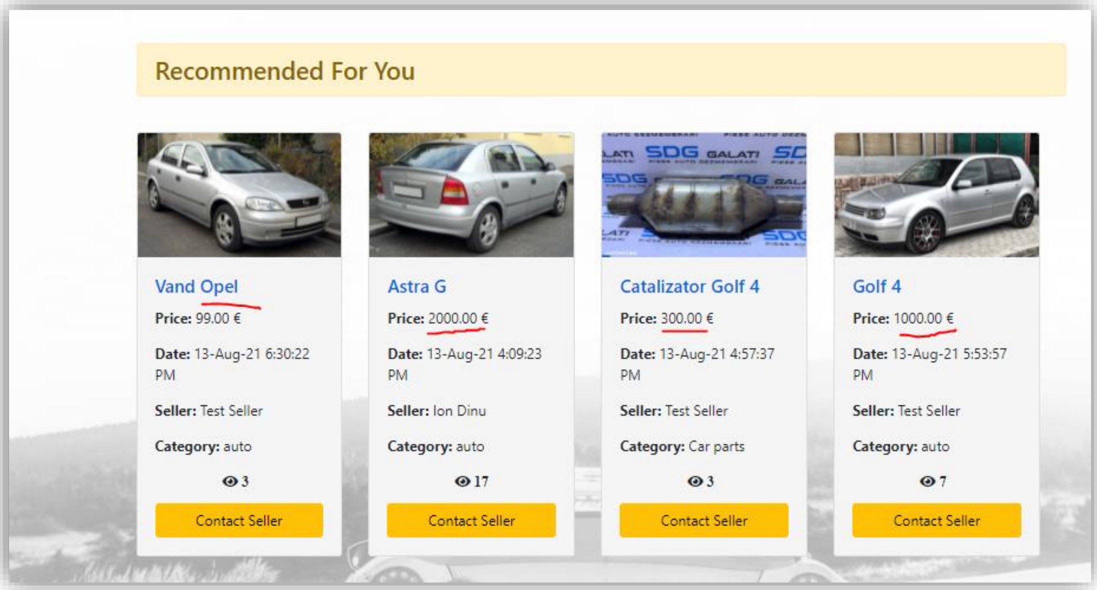


Figure 143. Recommender system results C

As we can observe now we have announcements with the Title „Opel" with priority A (first), then the second priority is the price B (second) that should be maximum 2000 and the other filters were not used yet, so are ignored.

It was selected for the title to be the first with opel because it is the most searched keyword:
opel – 3 times, golf – 2 times, peugeot – 1 time => opel is choosed

For the price is only 2000 so is taken that one, until is updated with a more recent or higher occurence price search.

5 CONCLUSIONS AND IMPROVEMENTS

In conclusion, the application Racooter – Cars trading website implements all the main functionalities presented in the functional requirements part of the project, using the principles, design patterns and application architecture presented in the correspondent chapters.

The application is representing a platform where buyers and sellers meet to satisfy their needs regarding selling and buying the perfect car for them. The website is rich in features that provides a great user experience, along side with the graphical user interface design that is intuitive, efficient and easy to use.

The application has three main access roles, represented by the buyer or seller, this is the main application user that can post announcements but also contact other sellers with the purpose of buying a car. To this type of user there are recommended announcements that will fits their interests and preferences, keeping him more enggaged and providing a great tool in matching sellers and buyers.

The supervisor of the platform is represented by the admin, that has management features in the admin panel, that will maintain a good and healthy community. Some of the main functionalities that are helping the admin reaching this goal are: reviewing announcements, because everything should be double checked before the announcements gets public. Dealing with reported users is another important feature that will keep a nice and user friendly environment where everyone can express their opinion, but also there can be consequences when the application guidelines are broken that can lead to blocking users from posting announcements or even to permanently account deletion.

Another great feature is the newsletter part where the admin or moderator can post adifferent news about the application updates, rules and guidelines, some hints how to optimize announcements in order to get the most attention and so on.

The moderator purpose is to help the admin in repetitive tasks like approving announcements and posting on the newsletter, in this way the users approval wait time is significantly lower, providing a better user experience and a plus of motivation to use the platform.

The project helped me a lot at desgning databases, applying certain design pattern, implement features that I did not created before and developing a great logical way of approaching new functionalities and how to integrate them with the rest of the application. Furthermore, developing this project also improved my programming skills using the C# programming language and my skills of using the ASP.NET framework which I do consider to be a powerful framework for implementing such a vast application, and along with repository pattern the adding more features on the way is more efficient and pleasant to do. I didn’t felt constraint in using this pattern, it offers a great flexibility and a wide range of implementation options. I used a lot LINQ and Lambda expressions to manipulate the database and entries from tables, so perfecting this skill also is really useful, especially when I needed to create more complex queries.

The research phase of the project was really important, that gave me an overview perspective of how the website should look and behave to different factors and user inputs. I studied existent online auto markets doing comparisions between them, that helped in noticing what are the most important features that every car trading webiste has, and how the application layout and interface should be designed in order to give the best user experience. Also researching the technologies that will make a great fit on my requirements was an important process, and I am really pleased with my tech stack choices.

Regarding the improvements of the application, I can refer to be able to create admin and moderator accounts directly from the interface by selecting which type of account the user wants to make. For now the admin and moderator accounts are created when the database is generated as a database seeder. Another improvement for the future is regarding the chat system visualization, it can display more information about the recipient like some profile picture that can be uploaded to become more familiar with the interlocutor and his online presence computed by the time he was last logged in the website. Also, the chat system can be upgraded to the next level consisting in using a third party to deal with sending and receiving messages, a potential solution could be using the *SignalR* library.

An improvement on the newsletter can be done, by adding the possibility that users can give likes and post comments in order to be more interactive, like a blog.

Other improvement of the app can be done in adding a favourite list section on every user account, to mark the announcements that are appealing for a particular user and also the seller should be notified if someone added his announcement as favourite to be aware that he is going in the right direction.

The last potential improvement of the platform is related to the recommender system, that has its weaknesses, a better approach is to use some machine learning algorithms that will find similarities between user searches, filtering, and favourite list and the app should start recommending this type of announcements. A library that can be used in this way on the ASP.Net environment and C# programming language is ML.NET.

It was a great experience to implement this application and document it in every detail, bringing a lot of skills improvement, the capability to write better documentation on large project, to make the proper research and to respect the functional requirements, design patterns and app architecture.

6 BIBLIOGRAPHY

[AF20] – Pro ASP.NET Core 3, Adam Freeman, Published by Apress; 8th edition, 2020

[EG94] – Design Patterns: Elements of Reusable Object-Oriented Software, Erich Gamma, Published by Addison-Wesley Professional; 1st edition, 1994

[JS19] - C# in Depth: Fourth Edition, Jon Skeet, Published by Manning Publications, 2019

[RB20] – ASP.NET Core Razor Pages: Full Stack Web Development with C#.NET, HTML, Bootstrap, CSS, JavaScript and Entity Framework Core, Robert Beasley, Published by Independently published, 2020

[TF18] – Web Development and Design Foundations with HTML5, Terry Felke-Morris, Published by Pearson, 2018

7 WEB REFERENCES

- Visual Studio IDE

[1] https://en.wikipedia.org/wiki/Microsoft_Visual_Studio

- .NET Framework

[2] https://en.wikipedia.org/wiki/.NET_Framework
[3] https://graffersid.com/advantages-and-disadvantages-of-using-net/
[4] https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-5.0

- C# programming language

[5] https://en.wikipedia.org/wiki/C_Sharp_(programming_language)
[6] https://www.c-sharpcorner.com/UploadFile/mkagrahari/introduction-to-object-oriented-programming-concepts-in-C-Sharp/
[7] https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/
[8] https://www.javatpoint.com/csharp-features

- Design patterns

[9] https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?WT.mc_id=dotnet-35129-website&view=aspnetcore-5.0
[10] https://docs.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-design

- HTML

[11] https://en.wikipedia.org/wiki/HTML
[12] https://www.w3schools.com/html/

- CSS

[13] https://www.w3schools.com/css/
[14] https://en.wikipedia.org/wiki/CSS

- Javascript

[15] https://www.w3schools.com/js/default.asp
[16] https://en.wikipedia.org/wiki/JavaScript

- Bootstrap

[17] https://www.w3schools.com/bootstrap/bootstrap_buttons.asp

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

- Microsoft SQL Server

[19] https://www.tek-tools.com/database/sql-server-best-practices-and-tools
[20] https://en.wikipedia.org/wiki/Microsoft_SQL_Server
[21] https://www.mssqltips.com/sqlservertutorial/9275/who-uses-sql-server-management-studio-and-why/

- Drawio

[22] https://www.diagrams.net/about

- GitHub

[23] https://en.wikipedia.org/wiki/GitHub
[24] https://apiumhub.com/tech-blog-barcelona/using-github/

- Auto Markets

[25] https://www.creditkarma.com/auto/i/best-site-sell-car
[26] https://en.wikipedia.org/wiki/Cars.com
[27] https://en.wikipedia.org/wiki/Auto_Trader_Group
[28] https://www.ebay.com/b/Auto-Parts-and-Vehicles/6000/bn_1865334

- Use-case diagrams

[29] https://www.ibm.com/docs/en/rational-soft-arch/9.6.1?topic=diagrams-use-case

- Class diagrams

[30] https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/

- Entity relationship diagram

[31] https://www.smartdraw.com/entity-relationship-diagram/

- UnitOfWork repository pattern

[32] https://www.c-sharpcorner.com/UploadFile/b1df45/unit-of-work-in-repository-pattern/

- Data Transfer Object (DTO)

[33] https://en.wikipedia.org/wiki/Data_transfer_object

- DataTable library

[34] https://cdn.datatables.net/

- Chat design css

[35] https://codepen.io/rKalways/pen/gNwVmE

A. THE SOURCE CODE

The source code for this project is really big, so I used GitHub for version control in order to track down the pending changes and keep always a working version on the main branch. I used two branches Develop, on which I made the changes first and add new features, and if everything is ok after testing the feature then I would make a pull request to merge it with Main branch.

Here is the link to the project repository:

https://github.com/WAEREWOLF/Bachelor-s-Thesis

C. CD / DVD

In this section it is attached the CD / DVD that contains all the source code of the project, this document and the final presentation of the thesis.

INDEX