# Дипломна работа

на тема

**Уеб приложение за покупко-продажба на автомобили**

Дипломант: **Александър Александров Свиларов**

ученик от 12А клас в

ЧПГДН "СофтУни Светлина"

Научен ръководител: **Калоян Костадинов**

Рецензент: **XXX XXX XXX**

Дата: **12 май 2023**

Сесия: май-юни 2023 г.

София, 2023 г.

# Съдържание

[Дипломна работа 1](#_Toc128415792)

[Съдържание 2](#_Toc128415793)

[Увод 3](#_Toc128415794)

[Цели на дипломния проект 4](#_Toc128415795)

[Глава 1. Проучване 5](#_Toc128415796)

[Глава 2. Използвани технологии 12](#_Toc128415797)

[2.1 Backend 12](#_Toc128415798)

[2.1.1 Какво е ORM и как функционира в Entity Framework? 13](#_Toc128415799)

[2.1.2 Защо C# е добър при разработката на уеб приложения? 16](#_Toc128415800)

[2.2 Frontend 17](#_Toc128415801)

[2.2.1 Angular и защо да използваме него? 17](#_Toc128415802)

[2.2.2 Компонентите в Angular 18](#_Toc128415803)

[2.2.3 Service-ите в Angular 19](#_Toc128415804)

[Глава 3. Проектиране и имплементация 20](#_Toc128415805)

[3.1 Архитектура на системата 20](#_Toc128415806)

[3.2 Имплементация на системата 21](#_Toc128415807)

[3.2.1 Имплементация на Backend системата 26](#_Toc128415808)

[3.2.1.1 Инициализация на базата и конфигурация 26](#_Toc128415809)

[3.2.1.2 Модели 27](#_Toc128415810)

3.2.1.2 Repository-та [28](#_Toc128415820)

[3.2.1.4 Контролери 29](#_Toc128415821)

[3.2.2 Имплементация на Frontend системата 32](#_Toc128415822)

[3.2.2.1 Компоненти 32](#_Toc128415823)

[3.2.2.2 Service layer и модели 35](#_Toc128415824)

[Глава 4. Ръководство за потребителя 30](#_Toc128415828)

[4.1 Swagger 39](#_Toc128415829)

[4.2 Frontend 40](#_Toc128415830)

[Заключение 43](#_Toc128415831)

[Информационни източници 44](#_Toc128415832)

[Рецензия на дипломен проект 45](#_Toc128415833)

# 

# Увод

DeltaCar е уеб приложение, което има за цел да улесни процеса на покупко-продажба на автомобили. То предоставя на потребителите възможност да намерят подходящия автомобил за тях, да сравняват различни марки и модели и да закупят автомобила онлайн.

Целта на проекта е да създаде едно лесно за използване приложение, което да предоставя на потребителите добра и удобна услуга. То ще бъде достъпно за всички, които желаят да купят или продадат автомобил, и ще съдържа полезна информация, която ще ги насочи към правилния избор.

Приложението ще предлага разнообразни филтри и опции за търсене, които да помогнат на потребителите да намерят автомобила, който отговаря на техните нужди и предпочитания. След като потребителите изберат автомобила, който искат да закупят, те ще могат да се свържат директно с продавача, за да се договорят за цената и да уточнят всички детайли по сделката. Например, потребителите могат да търсят автомобили по марка, модел, цена, година на производство, тип на двигателя, тип на горивната система, пробег и други параметри. Тези филтри помагат на потребителите да се фокусират върху тези автомобили, които отговарят на техните нужди. Системата за търсене на DeltaCar се отличава също и с възможността да се търсят автомобили по ключови думи, което е особено полезно за тези потребители, които търсят автомобил с даден проблем.

DeltaCar ще предлага и много други полезни функции за потребителите. Те ще могат да преглеждат описания на автомобилите, да разглеждат снимки и детайлни характеристики. Приложението ще предлага и интегрирани услуги за проверка на автомобили, които да помогнат на потребителите да се уверят, че сделката, която сключват, е сигурна и безопасна.

## Цели на дипломния проект

Целите на дипломния проект DeltaCar са да разработи уеб базирана платформа за продажба на автомобили, която да предоставя на потребителите бърз и лесен начин за търсене и закупуване на автомобили, както и за продажба на автомобили.

Специфичните цели на проекта включват следното:

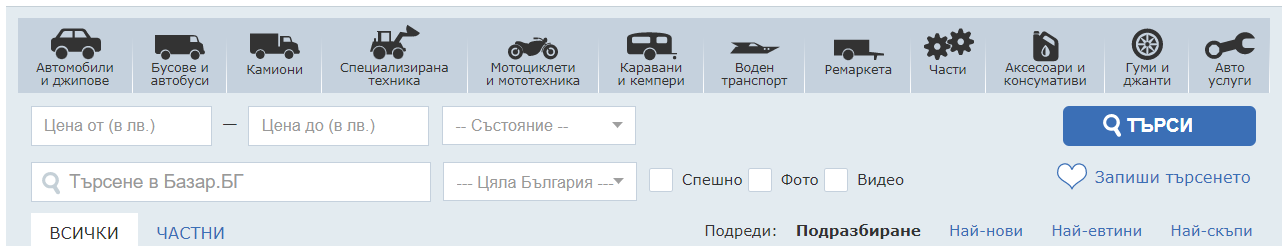
* Разработване на потребителски интерфейс, който да е удобен, лесен за използване и да предоставя ясна информация за всеки автомобил, който е наличен за продажба.
* Интегриране на търсачка за автомобили, която да позволява на потребителите да търсят автомобили по марка, модел, година на производство, цена и други характеристики.
* Интегриране на система за онлайн плащания, която да позволява на потребителите да закупуват автомобили директно през уебсайта. ***(Планирано)***
* Разработване на система за управление на продажбите, която да позволява на администраторите на сайта да управляват автомобилите, които са налични за продажба, както и да проследяват статуса на всяка продажба. ***(Планирано)***
* Интегриране на система за комуникация между потребителите и администраторите на сайта, която да позволява на потребителите да получават бърз и ефективен отговор на въпроси и да получават помощ при нужда. ***(Планирано)***

# Глава 1. Проучване

Закупуването или продажбата на автомобил може да бъде една времеемка и не винаги лесна задача. За да има успех продавачът, той трябва да притежава автомобил в добро състояние, привлекателен външен вид, добро описание, честно отношение към клиентите и тн. Преди съществуването на интернет пространството подобни сделки са се извършвали най-често в автокъщи – човек отива там, говори с човек, отговорен за продажбите, обяснява какво търси, изискванията си и предпочитанията си, след което бива насочен към няколко автомобила, с които автокъщата разполага и той избира този, който му допада най-много.

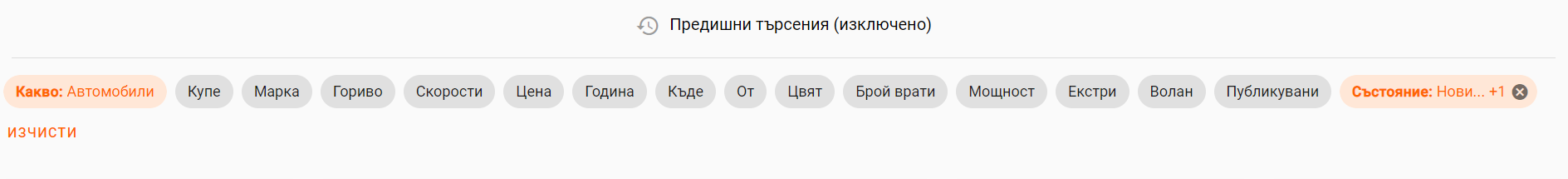
Един от основните проблеми на този начин за покупка на автомобил е времето, което отнема, без гарантиран резутат. Винаги може да се случи така, че автокъщата да не разполага с автомобил, който да покрива изискванията на даден клиент – тогава целият този процес започва отначало. След 2-3 различни срещи денят е загубен, а купувачът все още няма желаният автомобил. С развитието на интернет, подобни проблеми бързо намират своите решения – сферата на автомобилите не е изключение. В днешно време най-популярният начин за покупка на автомобил е онлайн.

В България имаме няколко основни сайта за тази цел – Mobile.bg, Cars.bg, Auto.bg, както и Bazar.bg и Olx.bg. Всеки един от тези сайтове има своите плюсове и минуси. В сайтове като Bazar.bg търсачката е силно ограничена по ключови думи, локация, състояние и цена:



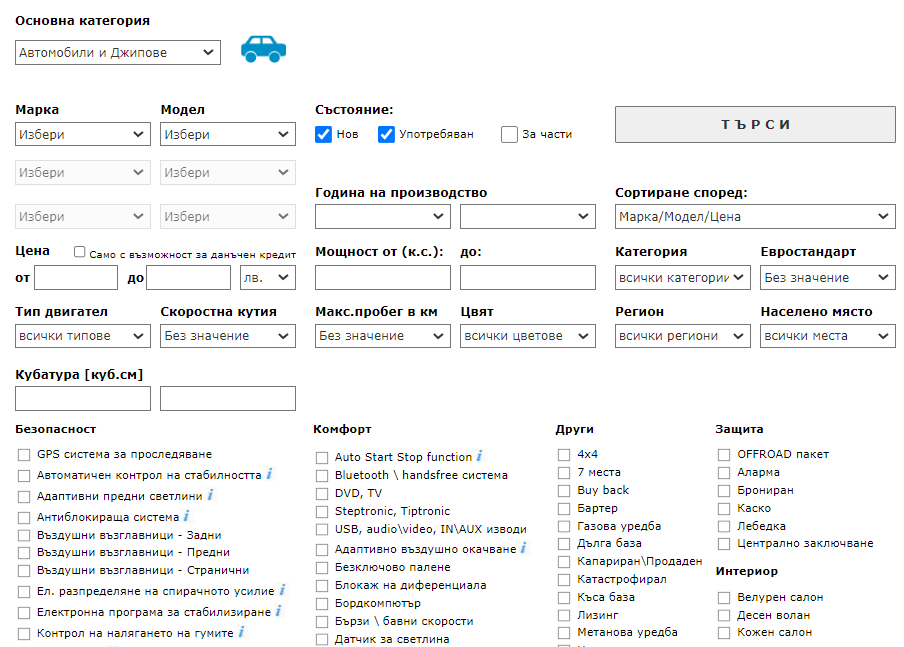
Фигура 1: Търсачката в Bazar.bg

В Auto.bg наблюдаваме подобна ситуация – ограничена търсачка, този път с малко повече опции:



Фигура 2: Търсачката в Auto.bg

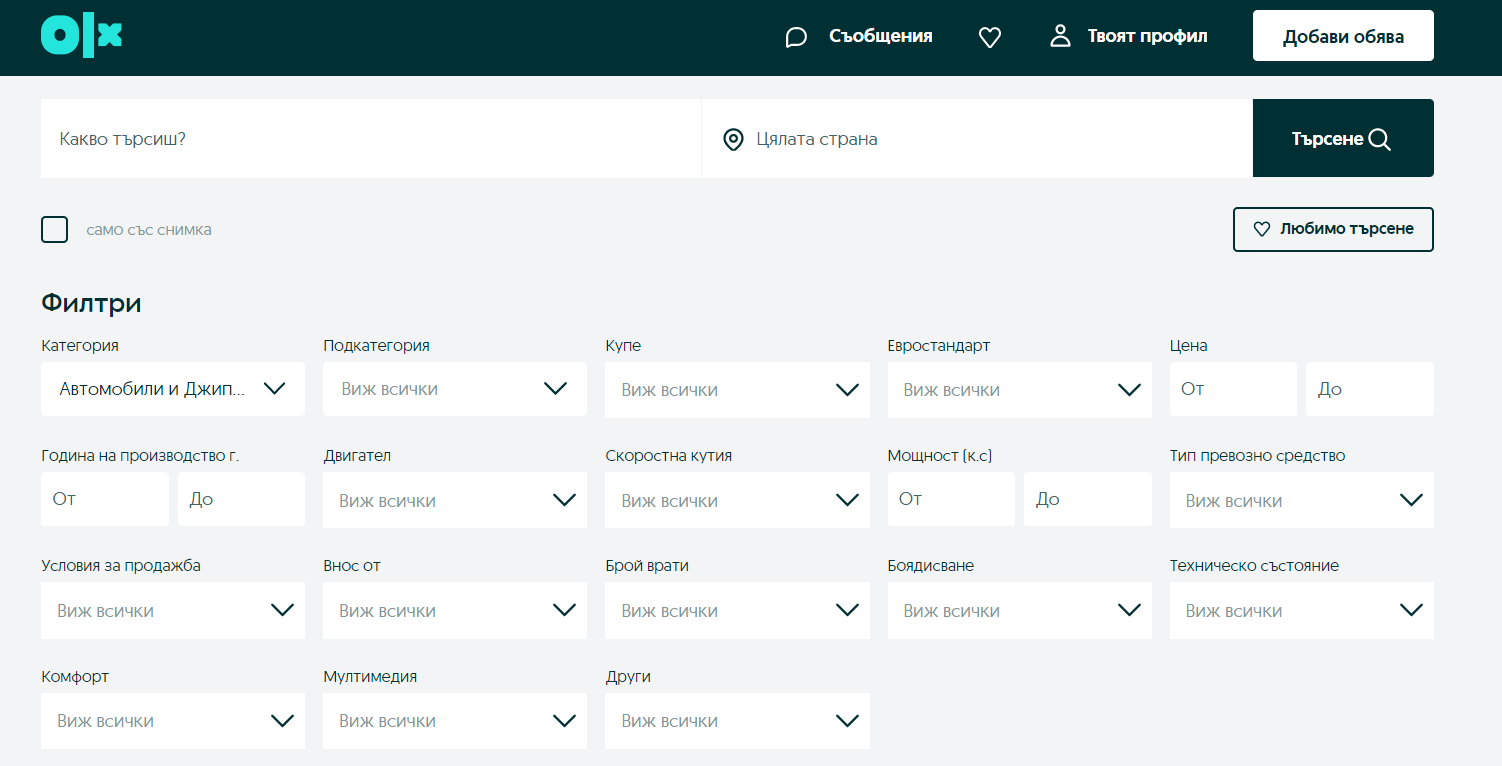
В сайтове като Mobile.bg ще забележим, че търсачката е сравнително по-обогатена, имайки редица опции, от които да изберем:



Фигура 3: Търсачката в Mobile.bg

Mobile.bg разполага с много добре разработена търсачка, която включва почти всичко, от което се нуждае един клиент. Единственото, което липсва, е търсенето по ключови думи – една важна подробност, която често бива забравена. Има хора, които търсят коли със специфичен проблем – например има механици, които търсят кола за „гарнитура на глава“, която да оправят и продадат за повече пари. DeltaCar се отличава от другите сайтове за покупко-продажба на автомобили благодарение на своята по-добре разработена система за търсене, която обединява различните видове търсачки, показани по-горе. Потребителите имат достъп до множество параметри, които могат да използват, за да намерят точно този автомобил, който отговаря на техните нужди и предпочитания. Функцията за за търсене по ключови думи е много важна част от системата за търсене, с която разполага DeltaCar.

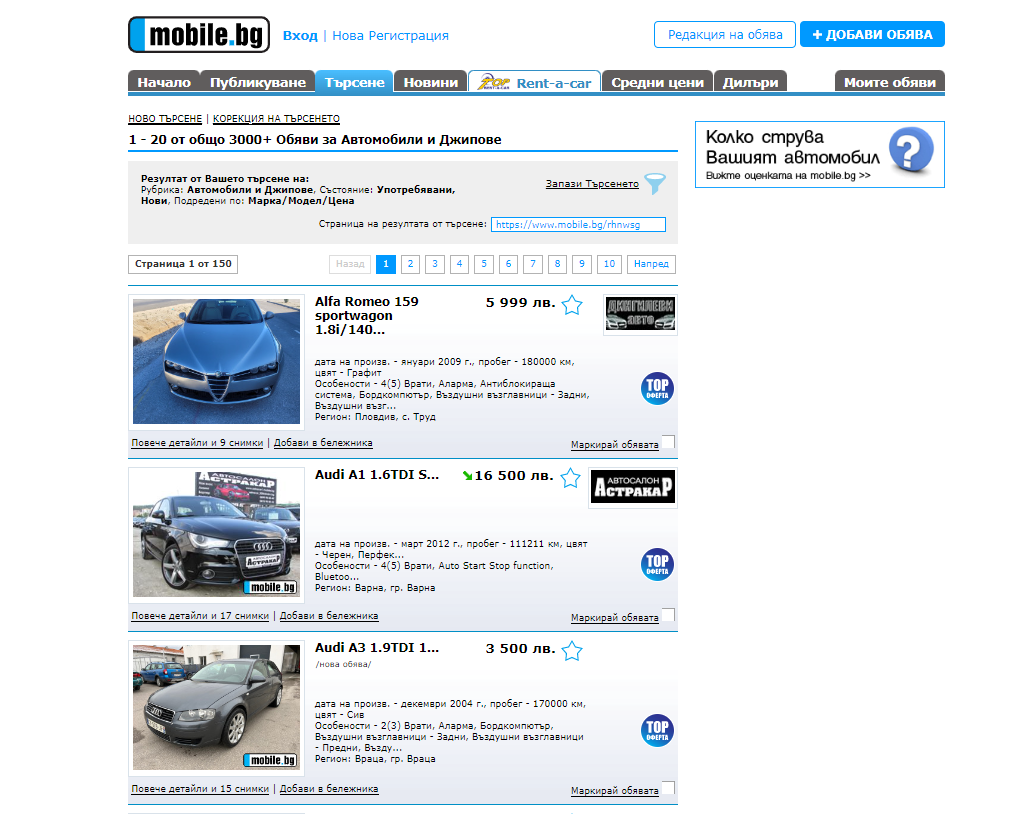
От всички подобни сайтове изненадващо най-добра търсачка има Olx.bg, като единственият проблем там, е че сайтът не е стриктно предназначен за покупко-продажба на автомобили.



Фигура 4: Tърсачката в Olx.bg

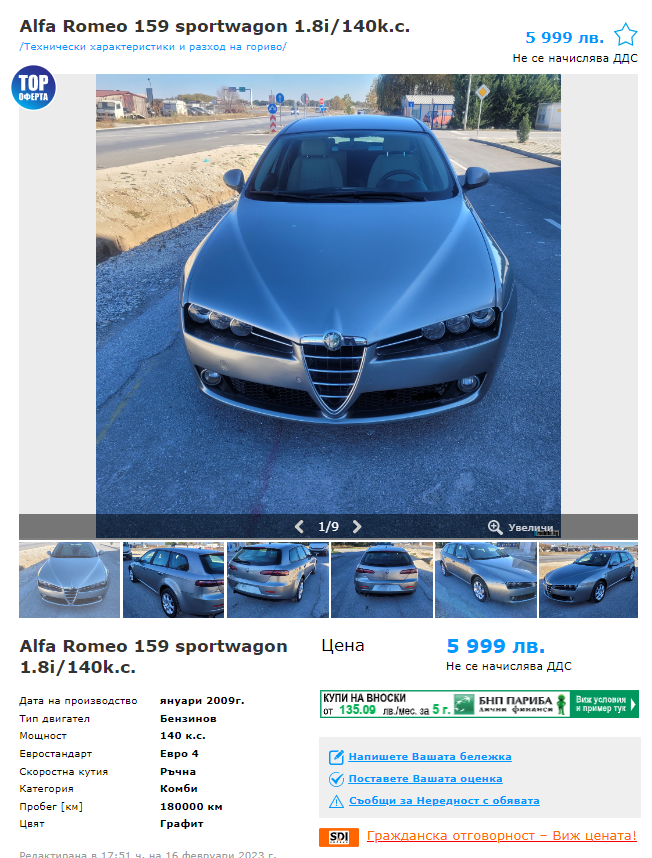
Нека разгледаме и дизайна на различните уебсайтове, които предлагат услуга за покупко-продажба на автомобили.

В Mobile.bg дизайнът е семпъл, но вече е леко остарял. Със сигурност не е в крак с тенденциите в уеб дизайна, но пък е лесен за разбиране и потребителите не се затрудняват да отгатнат как да навигират из сайта.

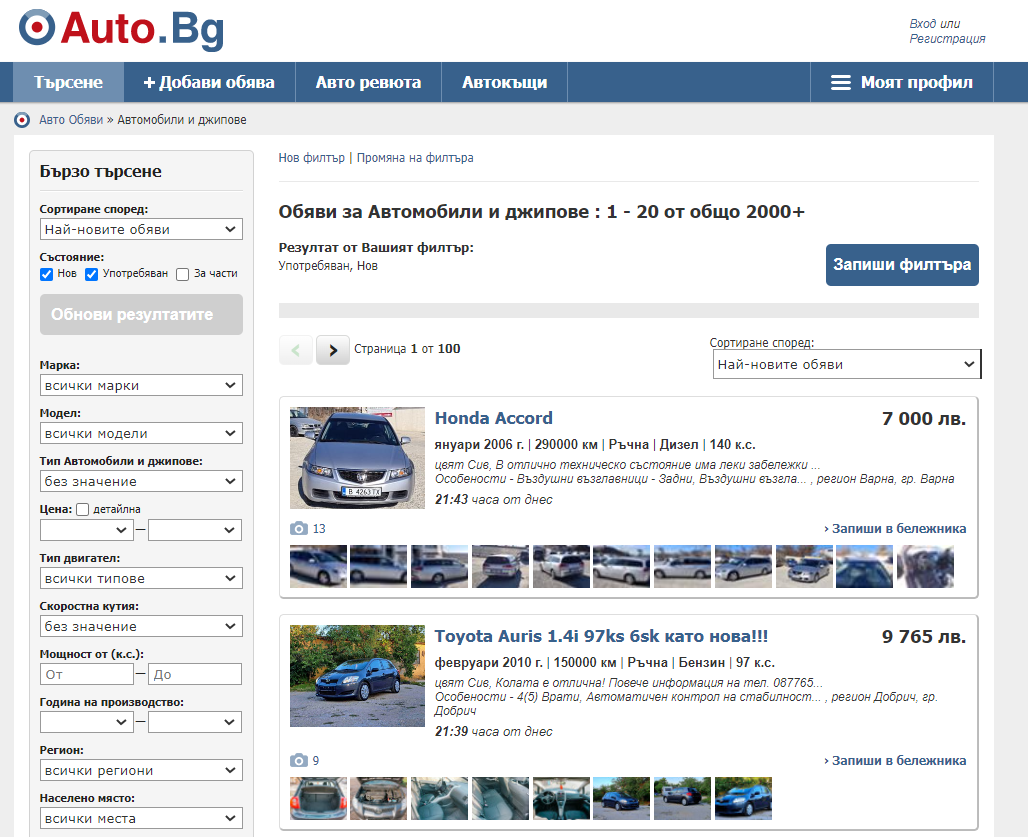


Фигура 5: Дизайн в Mobile.bg

Дизайнът в самата страница, където се намира обявата, е добре структуриран, като е предтавена най-важната за потребителя информация:

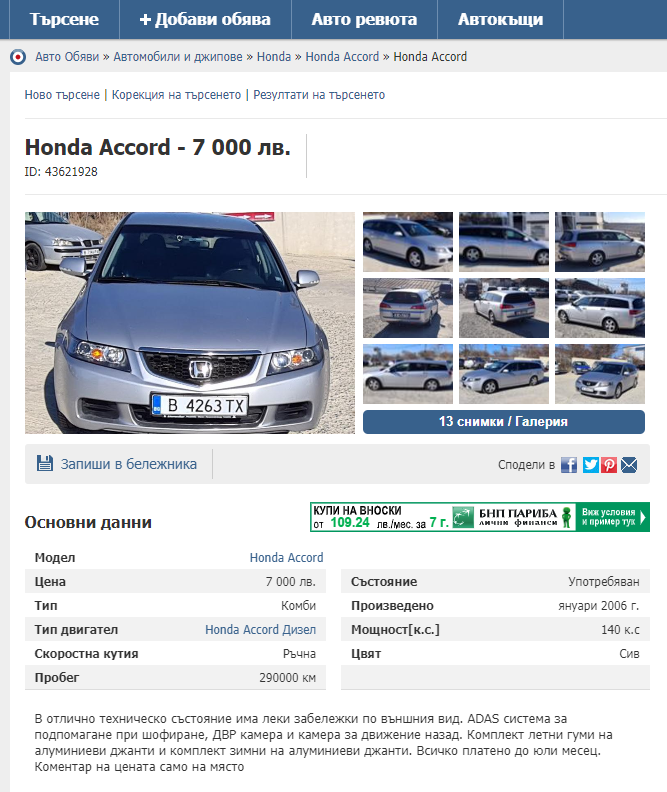


Фигура 6: Дизайн на обява в Mobile.bg

Дизайнът в Аuto.bg е подобен, но тук виждаме подобрение в Navigation Bar-a, както и опцията за Бързо Търсене, която е полезна.

Фигура 7: Дизайн в Auto.bg

Подобен дизайн наблюдаваме и в страницата на обявата:



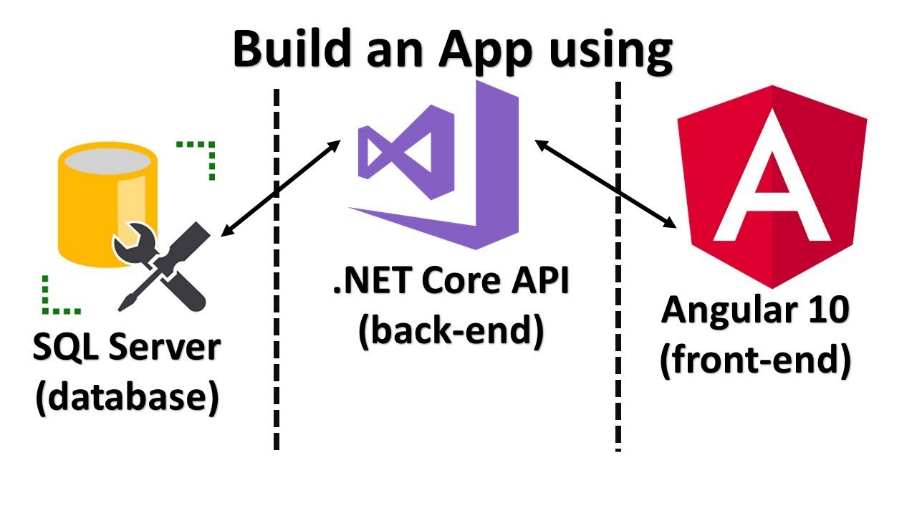
Фигура 8: Дизайн на обява в Auto.bg

# Глава 2. Използвани технологии

За да създадем уеб приложението, ще използваме различни технологии за backend-a и frontend-a. За backend-a ще използваме C# и Entity Framework, а за frontend-a – Angular (с Typescript).

# 2.1 Backend

Първата стъпка в разработката на уеб приложения е създаването на Backend-a, който случи за достъпването на данните, тяхното модифициране и т.н. В модела клиент-сървър клиентът обикновено се счита за frontend, а сървърът обикновено се счита за backend.



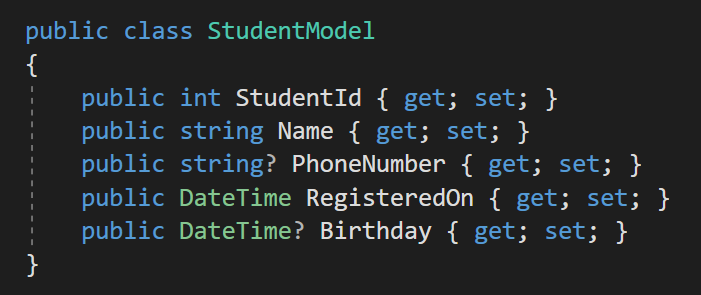
Фигура 9: Структурата на уеб приложението (технологии)

# 2.1.1 Какво е ORM и как функционира в Entity Framework?

ORM е начин за конвертиране на данни между релационна база данни и обектно-ориентиран език за програмиране, създавайки база данни с виртуални обекти, която може да се използва от обетно-ориентирания език за програмиране.

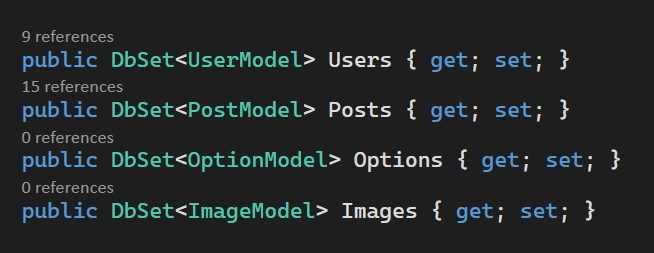
В Entity Framework това се случва чрез помощта на модели и миграции - процес на автоматично създаване и актуализиране на релационната база данни на основата на промените в моделите. Това означава, че EF Core може да проследява промените в модела на данните и да създава, актуализира или премахва таблиците и колоните в базата данни, когато е необходимо.

Един модел в Entity Framework представлява клас, който има различни property-та. Те могат да бъдат различни типове, като Entity Framework сам ще ги превърне в типове, които SQL Server да разбере – например тип ***string*** ще бъде превърнат в тип ***nvarchar***, тип ***DateTime*** ще бъде превърнат в ***datetime2*** и така нататък:



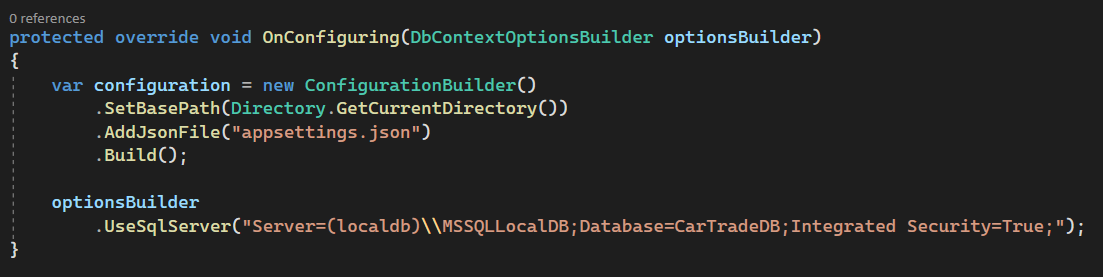
Фигура 10: Модел в Entity Framework

Тези модели можем да превърнем в таблици, използвайки т.нар ***DbSet***, където задаваме и желаното име на таблицата:



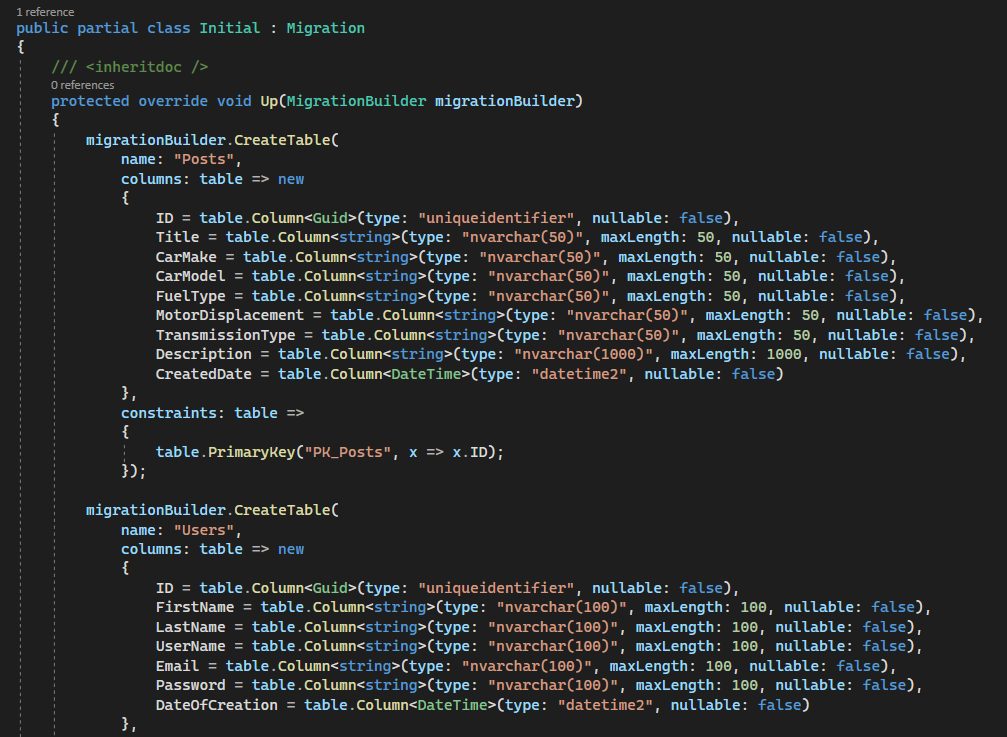
Фигура 11: Използване на DbSet в Entity Framework

Разбира се, трябва да имаме и конфигурация, коятно да посочи на Entity Framework как да намери / къде да създаде базата данни, която искаме да създаваме, използваме или модифицираме. Това можем да постигнем чрез метода **OnConfiguring**:



Фигура 12: Метод OnConfiguring в Entity Framework

След като имаме модел и сме решили, че той ще бъде таблица, задавайки го като DbSet, трябва да направим миграция. Миграцията в Entity Framework се извършва с помощта на инструмента за миграции на Entity Framework, който позволява на разработчиците да генерират т.нар миграционни скриптове и да ги приложат върху базата данни. Този процес включва изграждане на миграционна история, която записва всички промени в модела на данните, като например добавяне или премахване на таблица или колона. Когато се генерира нова миграция, инструмента за миграции автоматично генерира необходимите SQL команди за актуализиране на базата данни.

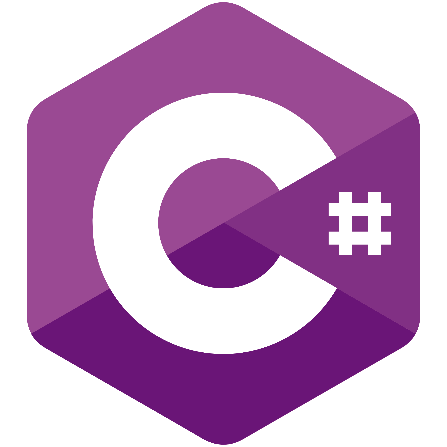


Фигура 13: Миграционен скрипт в Entity Framewok

# 2.1.2 Защо C# е добър при разработката на уеб приложения?

Ето някои от основните причини защо **C#** е сигурен избор при разработването на уеб приложения:

* Обектно-ориентиран подход: **C#** е език със строга типизация, който поддържа обектно-ориентирано програмиране. Това позволява на програмистите да създават модулни и структурирани приложения, които са лесни за поддръжка и мащабиране.
* **ASP.NET: ASP.NET** е платформа за създаване на уеб приложения, която използва **C#** като основен език за програмиране. **ASP.NET** предоставя мощни инструменти за създаване на сигурни и ефективни уеб приложения.
* Голяма общност: **C#** е един от най-популярните езици за програмиране, който има голяма и активна общност от разработчици. Това означава, че има много ресурси, като форуми, уроци и документация на разположение.
* Лесно мащабиране: **C#** езикът и **ASP.NET** платформата предлагат много възможности за мащабиране на уеб приложенията. Това означава, че програмистите могат да добавят нови функции и да променят кода на приложението, без да се налага да го изграждат отново.
* Сигурност: **C#** и **ASP.NET** са добре известни със своята сигурност. Например, **ASP.NET** предоставя вградена защита срещу най-различни видове атаки, като SQL инжекции, кръстосан скриптинг и други.



# 2.2 Frontend

След като backend-а е готов, стигаме до реализацията на приложението в уеб среда, за което ни е нужен Frontend. Frontend-a в едно уеб приложение е частта, която потребителите виждат и с която взаимодействат.

# 2.2.1 Angular и защо да използваме него?

След като backend-а е готов, стигаме до реализацията на приложението в уеб среда, за което ни е нужен Frontend. Frontend-a в едно уеб приложение е частта, която потребителите виждат и с която взаимодействат.

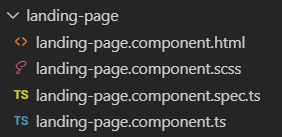
Angular е framework за разработка на едностранични приложения (SPA - Single Page Applications) с помощта на HTML, CSS и TypeScript. Angular е разработен от Google и предоставя много функционалности, които подобряват разработката на уеб приложения, като например:

* **Two-way data binding (Двупосочно свързване на данни)**: Двупосочното свързване на данни на Angular позволява на разработчиците да създадат връзка между модела и изгледа. Всички промени, направени в модела, автоматично се отразяват в изгледа и обратно.
* **Dependency injection (инжектиране на зависимости)**: Системата за инжектиране на зависимости на Angular позволява на разработчиците лесно да управляват зависимостите и да създават компоненти за многократна употреба.
* **Routing (маршрутизиране)**: Вграденият маршрутизатор на Angular позволява на разработчиците лесно да навигират между различните страници на тяхното приложение.
* **Directives (директиви)**: Директивите на Angular позволяват на разработчиците да създават персонализирани HTML тагове и атрибути, които могат да се използват за добавяне на функционалност към тяхното приложение.
* **Testing (тестване)**: Angular предоставя надеждна рамка за тестване, с която разработчиците могат да тестват своите приложения, гарантирайки, че те са надеждни и без грешки.

# 2.2.2 Компонентите в Angular

В Angular, компонентите са основните строителни блокове на приложението. Компонентите са самостоятелни единици, които обхващат HTML, CSS и JavaScript кода и позволяват на разработчиците да създават многократно използваеми UI компоненти за приложенията си.

Компонентите в Angular са създадени чрез декоратора **@Component**, който определя метаданните на компонента, като името, селектора, шаблона (template), стиловете и други. Те също така имат свойства (properties), които могат да бъдат свързани двустранно с данни в шаблона на компонента чрез двустранно свързване на данни (two-way data binding). Когато генерираме един компонент в Angular, файловете, които той геерира, са следните:



Фигура 14: Файловете, които съставят един компонент в Angular

Имаме 4 файла, като всеки един от тях има своята роля:

* Файл с разширение **.html** – в него се пише HTML кодът на компонента, неговата структура.
* Файл с разширение **.css** (**scss**) – в него се пише визуалното оформление на компонента, изгледът.
* Файл с разширение **.spec.ts** – в него компонентът се създава и инициализира.
* Файл с разширение **.ts** – в него можем да пишем frontend-ската логика, която искаме (валидации, пренасочване, повикване на функции и така нататък)

# 2.2.3 Service-ите в Angular

Service-ите в Angular са основнен начин за обмяна на данни между frontend-a и backend-a на дадено уеб приложение. Важно е тяхното използване, защото то подпомага лесното преизползване на вече написан код, което на свой ред е една от основните черти на оптимизацията.

В service-ите ние се свързваме с backend-a чрез така наречените **HTTP** **заявки**, като най-често използваните са следните:

* **GET**: Извлича информация или данни от сървъра за определен ресурс. Това е най-често използваният метод на HTTP заявка и често се използва за извличане на уеб страници, изображения и други видове файлове.
* **POST**: Подаване на данни към сървъра за обработка, обикновено като част от подаване на формуляр или за създаване на нов ресурс.
* **PUT**: Актуализира съществуващ ресурс на сървъра с нови данни.
* **DELETE**: Премахва ресурс от сървъра.

Именно чрез тези заявки разработчиците правят връзката между frontend-a и backend-a.



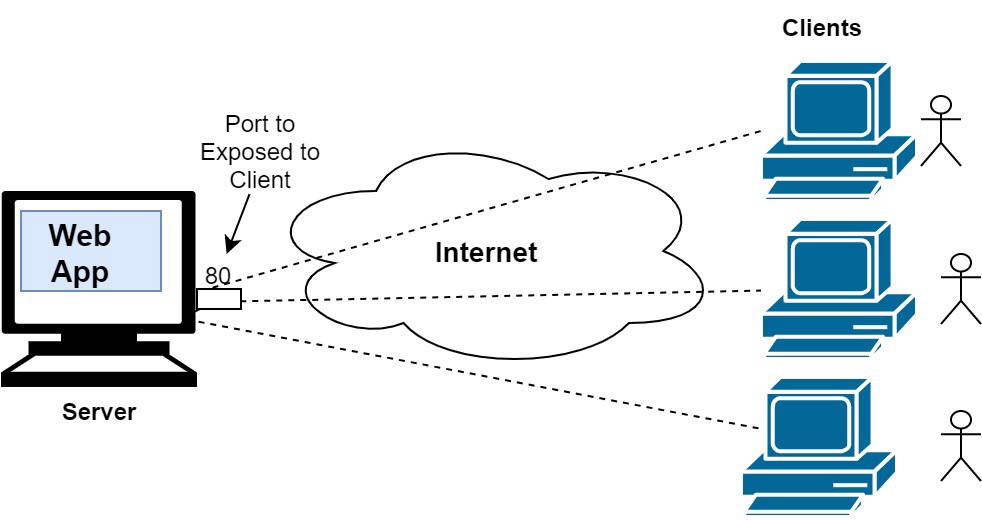
Фигура 15: Примерен service в Angular

# Глава 3. Проектиране и имплементация

Сега ще обясня как е проектирано и имплементрано уеб приложението DeltaCar. Ще разгледаме логиката в backend-a, data-access layer-a, миграциите, моделите, frontend-а, как се осъществява връзката между клиентската и сървърната част и др.

## 3.1 Архитектура на системата

Архитектурата на проекта ще е client-server архитектура, която изглежда по следният начин:



Фигура 16: Client-Server архитектура

Client-server архитектурата е модел на софтуерен дизайн, където приложението е разделено на две основни компонента - клиент и сървър. Клиентът е софтуер, който изпълнява заявки към сървъра, за да получава информация или да извършва операции. Сървърът е софтуер, който отговаря на заявките на клиента и обработва информацията, която е необходима за тяхната изпълнение.

Системата работи като клиентите комуникират със сървъра чрез мрежата, като изпращат заявки за информация или изпълнение на операции. Сървърът получава тези заявки, обработва ги и изпраща отговорите обратно към клиентите. Това позволява на различни клиенти да работят едновременно с един и същи сървър, като споделят обща информация и ресурси.

Предимствата на този модел включват улеснена съвместна работа и управление на данни, както и по-лесно скалиране на приложенията. В допълнение, този модел предоставя по-голяма сигурност на данните, тъй като те са съхранени централизирано на сървъра и могат да бъдат защитени с подходящи механизми за достъп и сигурност.

## 3.2 Имплементация на системата

В следващите страници ще покажа нагледно как е имплементирана системата, както в Backend-a, така и във Frontend-a.

## 3.2.1 Имплементация на Backend системата

Първа стъпка в имплементацията на backend-a е инициализацията и конфигурацията на базата данни, след това се правят моделите, първи миграции, след това се правят repository-тата и controller-ите. В repository-тата е логиката, а в controller-ите са самите endpoint-ове и отговорите на сървъра.

## 3.2.1.1 Инициализация на базата и конфигурация

Инициализацията на базата е сравнително лесна, понеже Entity Framework генерира не малка част от кода, който е нужен. Това представля ва DatabaseContext файлът:

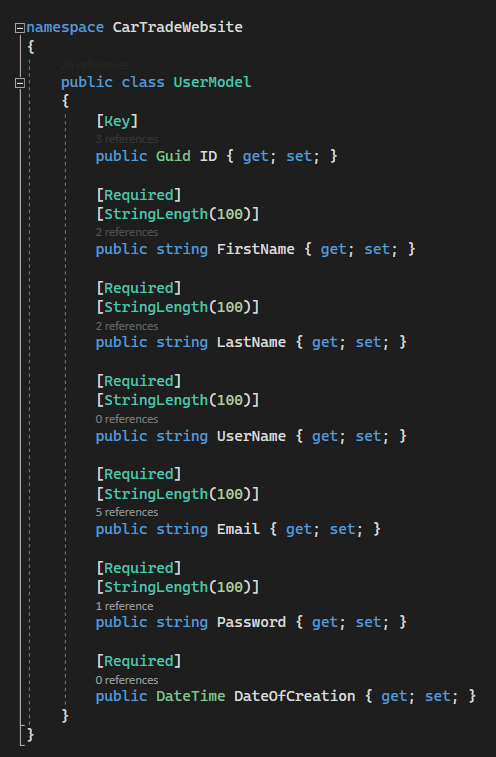


Фигура 17: Database Context-a

## 3.2.1.2 Модели

Моделите са ни необходими, за да зададем на Entity Framwork скелета, по който да създаде базата данни, която ни е необходима. В тези модели съществуват различни анотации, които служат за различни неща, като ограничение на дължината на записа, означаване на ID-то и тн.

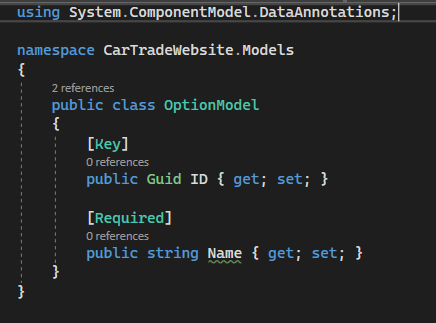
В DeltaCar разполагаме с 4 модела – модел на потребителя (UserModel.cs), модел на публикацията (PostModel.cs), модел на различните опции, които можем да имаме в публикацията (OptionModel.cs), както и модел на изображенията в публикациите (ImageModel.cs). Те представляват следното:



Фигура 18: Модел на потребител

Този модел се използва за представяне на потребител в системата за търговия на автомобили. Включва следните property-та:

* **ID** - уникален идентификатор за потребителя. Това property е маркирано с атрибута [*Key*], който указва, че това е ключово поле в базата данни.
* **FirstName** - собствено име на потребителя. Това е задължително поле, което трябва да има дължина до 100 символа. Това е поле от тип string.
* **LastName** - фамилно име на потребителя. Това е задължително поле, което трябва да има дължина до 100 символа. Това е поле от тип string.
* **UserName** - потребителско име на потребителя. Това е задължително поле, което трябва да има дължина до 100 символа. Това е поле от тип string.
* **Email** - имейл адрес на потребителя. Това е задължително поле, което трябва да има дължина до 100 символа. Това е поле от тип string.
* **Password** - парола за достъп до профила на потребителя. Това е задължително поле, което трябва да има дължина до 100 символа. Това е поле от тип string.
* **DateOfCreation** - дата на създаване на профила на потребителя. Това е задължително поле, което е от тип DateTime.



Фигура 19: Моделът на опцията

Този модел се използва за представяне на потребител в системата за търговия на автомобили. Включва следните property-та:

## Този модел се използва за представяне на опция на автомобил. Той съдържа две property-та:

## ID: Това е уникалният идентификатор на опцията, който е дефиниран като GUID (Globally Unique Identifier). Този property е маркиран с [*Key*] атрибута.

## Name: Това е името на опцията, която има автомобилът. Този property е маркиран с [Required] атрибута, който указва, че това поле е задължително за попълване и не може да бъде оставено празно.



Фигура 20: Модел на изображение

## Този модел се използва за представяне на изображение. Той съдържа две property-та:

## ID: Това е уникалният идентификатор на опцията, който е дефиниран като GUID (Globally Unique Identifier). Този property е маркиран с [Key] атрибута.

## Name: Това е името на опцията, която има автомобилът. Този property е маркиран с [Required] атрибута, който указва, че това поле е задължително за попълване и не може да бъде оставено празно.

## 

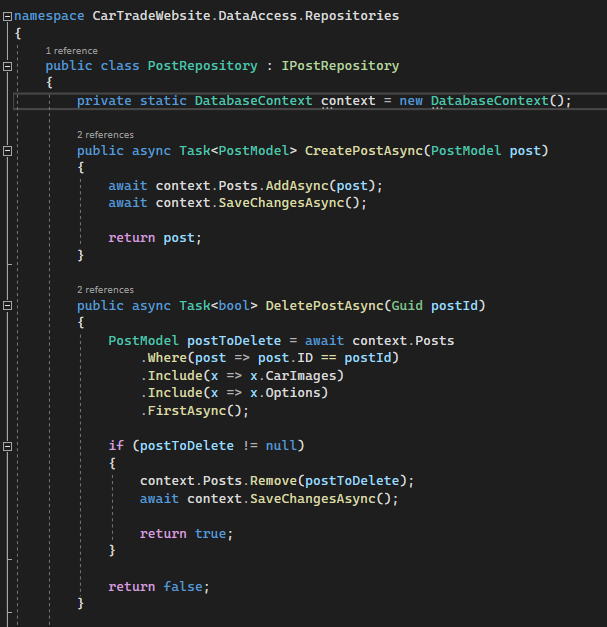
Фигура 21: Модел на публикация

## Този модел съдържа информацията на публикацията. Той съдържа следните property-та

* **ID** - това property е от тип "Guid" и се използва като ключ на модела.
* **Title** - Заглавието на обявата. Той е задължителен (Required) и има ограничение на дължината на символите до 50 (StringLength(50)).
* **CarImages** - това property е от тип "IEnumerable<ImageModel>" и представлява списък от изображения на автомобила. Той е задължителен (Required).
* **CoverImageURL** - URL адреса на основното изображение на автомобила. Той е задължителен (Required).
* **CarMake** - Марката на автомобила. Той е задължителен (Required) и има ограничение на дължината на символите до 50 (StringLength(50)).
* **CarModel** - Моделът на автомобила. Той е задължителен (Required) и има ограничение на дължината на символите до 50 (StringLength(50)).
* **FuelType** – Тип на горивото, използвано от автомобила. Той е задължителен (Required) и има ограничение на дължината на символите до 50 (StringLength(50)).
* **EngineLayout** - Разположението на двигателя на автомобила. Той е задължителен (Required) и има ограничение на дължината на символите до 50 (StringLength(50)).
* **Mileage** - Пробегът на автомобила. Той е задължителен (Required).
* **YearOfProduction** - Годината на производство на автомобила. Той е задължителен (Required).
* **Color** - Цветът на автомобила.
* **EngineDisplacement** – Обемът на двигателя на автомобила, който трябва да бъде въведен задължително и не може да бъде по-дълъг от 50 символа.
* **TransmissionType**: Това е типът на скоростната кутия на автомобила, който също трябва да бъде въведен задължително и не може да бъде по-дълъг от 50 символа.
* **Description**: Това е описание на автомобила, което е задължително и не може да бъде по-дълго от 1000 символа.
* **Location**: Това е местоположението на автомобила, което е задължително и не може да бъде по-дълго от 1000 символа.
* **Price**: Това е цената на автомобила, която е задължителна и трябва да бъде въведена като десетично число.
* **Options**: Това е колекция от опции, налични за автомобила, като например електрически стъкла, климатик и други. Това е задължително property и трябва да съдържа поне една опция.
* **DateOfCreation**: Това е датата на създаване на обявата за продажба на автомобила, която е задължителна и трябва да бъде въведена като обект от тип DateTime.

## 3.2.1.3 Repository-та

В repository-тата ще слагаме логиката, на която се базира цялото приложение. Там ще обработваме данни от базата, ще правим нови записи и ще изтриваме такива.



Фигура 22: Repository за публикация

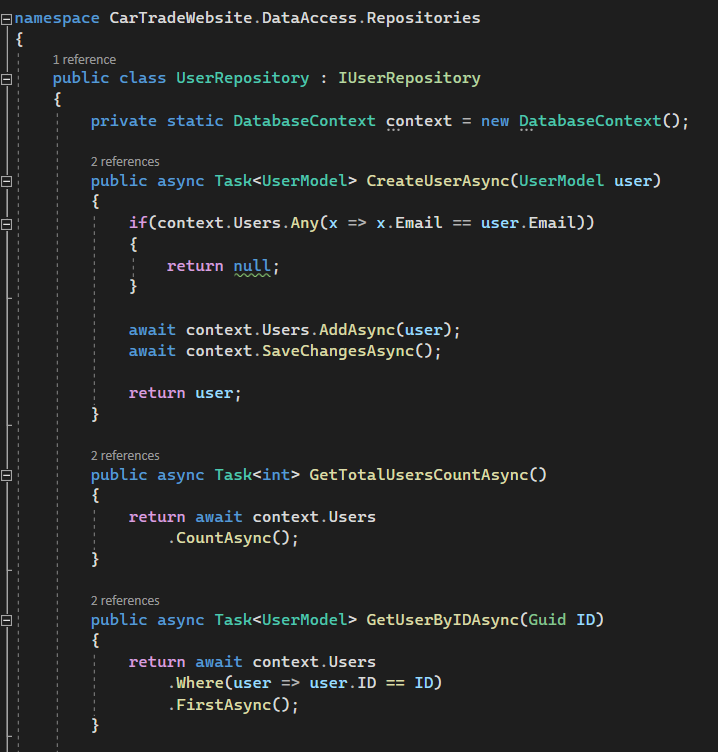
В repository-то на публикацията ще имаме различни методи:

* Добавяне на публикация
* Редактиране на публикация
* Изтриване на публикация
* Взимане на публикация по ID
* Взимане на всички публикации
* Взимане на брой на всички публикации

# 

Фигура 23: Repository на автентикацията

Това е Repository-то, което е отговорно за връщането на **JWT Token**, който служи за автентикация в приложението от клиентската страна. Имаме само един метод, който служи за Login в апликацията.



Фигура 24: Repository на потребител

Това е Repository-то, което отговаря за потребителите. То съдържа следните методи:

* Добавяне на потребител
* Редактиране на потребител
* Изтриване на потребител
* Достъпване на потребител по ID
* Достъпване на всички потребители
* Достъпване на броят на всички потребители



Фигура 25: Repository за търсачката

Това е Repository-то, което е отговорно за търсачката. В него има различни методи, които търсят по различни параметри: Заглавие, описание, марка, модел и др.

## 3.2.1.4 Контролери

Контролерите са отговорни за комуникацията на приложението с backend логиката. Пример ще дам с контролера за потребителя, като останалите работят на същия принцип.



Фигура 26: Контролер за потребител

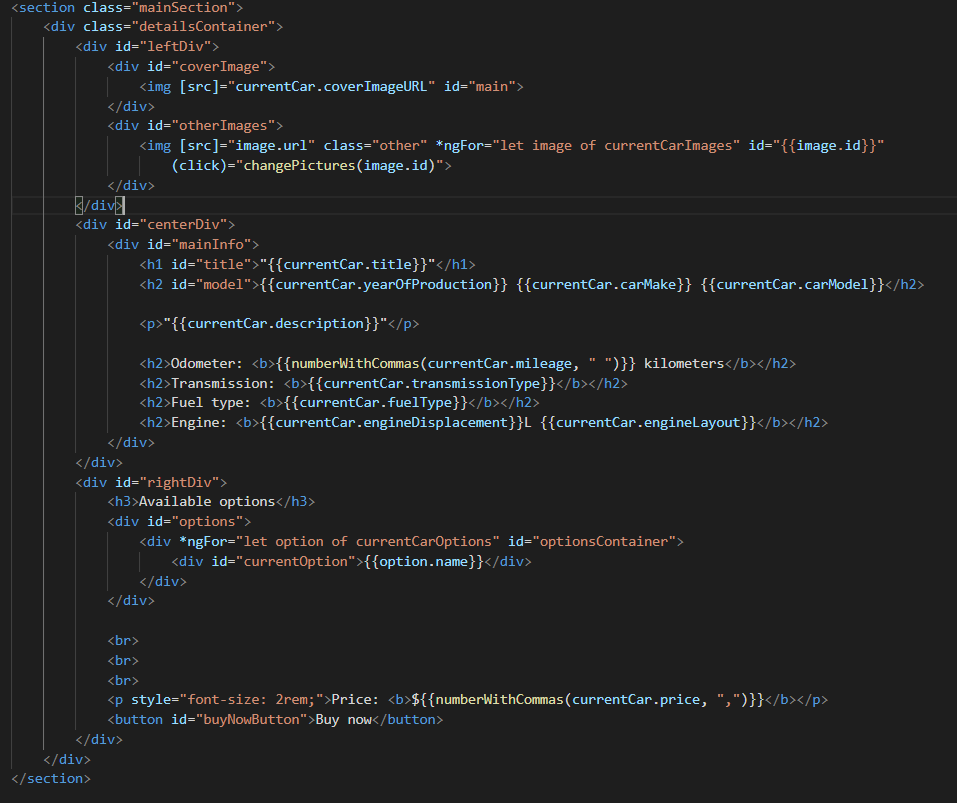
В този контролер комуникацията се изършва в различни методи, които имат същата цел като методите в Repository-тата.

## 3.2.2 Имплементация на Frontend системата

Frontend-а се състои от компоненти, които се свързват помежду си чрез Routing-ът, който е вграден в Angular, service-ите служат за комуникация с backend-a, използвайки т.нар Response модели.

## 3.2.2.1 Компоненти

Нека разгледаме как работи един компонент – в случая този компонент ще бъде страницата, съдържаща описанието на обявите.

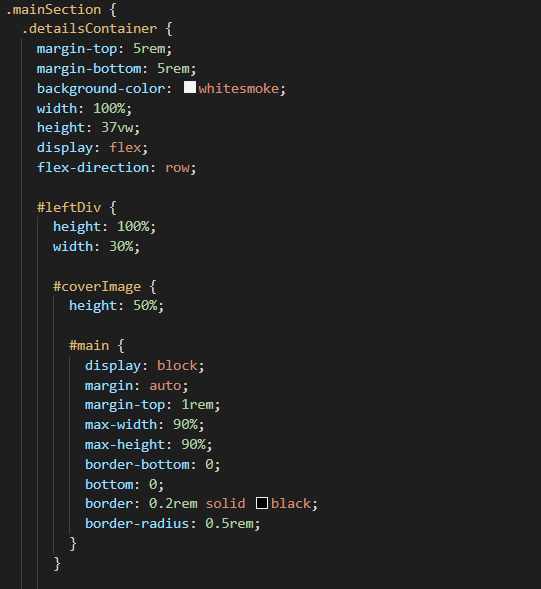


Фигура 27: HTML в компонента

Това е фрагмент от HTML код за страница с подробности за обявата. Той съдържа раздел с три div-a за показване на подробности за автомобила, изображения и налични опции.

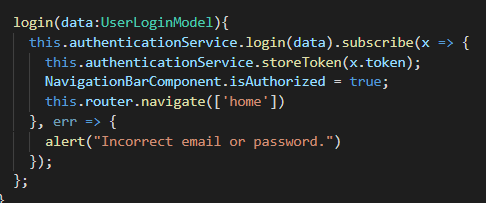
Лявото подразделение показва изображенията на автомобила, като основното изображение се показва по-голямо, а другите изображения са показани в по-малки миниатюри. В централния div се показва информация за автомобила, като например заглавие на обявата, модел, описание, показания на километража, вид на трансмисията, вид на горивото и данни за двигателя. В дясното подразделение се показват наличните опции за автомобила и бутон "Buy now".

А ето така изглежда CSS файлът към тази страница:



Фигура 28: CSS на страницата

Логиката, която помага на frontend-а да работи, се намира в .ts файлът – за пример давам логиката зад Login-a в приложението:



Фигура 29: Логиката в Login компонента

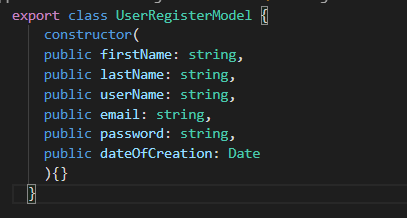
## 3.2.2.2 Service layer и модели

В т.нар service layer ние осъществяваме връзката между frontend-a и backend-a – това постигаме със заявки към различни endpoint-ове, които сме задали предварително в backend-a.

# 

Фигура 30: Service-са за автентикация

В тези service-и ние използваме модели, за да знаем как да запишем информацията, която ни се изпраща от backend-a. Пример ще дам с модела за потребител:



Фигура 31: Модел за регистрация на потребител

Както в backend-a, така и тук имаме необходимите property-та:

* **firstName**: от тип string - съдържа първото име на потребителя.
* **lastName**: от тип string - съдържа фамилията на потребителя.
* **userName**: от тип string - съдържа потребителското име на потребителя.
* **email**: от тип string - съдържа имейл адреса на потребителя.
* **password**: от тип string - съдържа паролата на потребителя.
* **dateOfCreation**: от тип Date - съдържа датата на създаване на акаунта на потребителя.

## 3.2.2.3 Routing в Angular

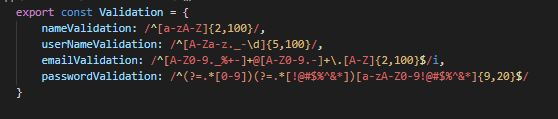
# 

Фигура 32: Route path-овете

В този пример, има следните маршрути:

* **path: 'home'** - който съответства на *HomeComponent* и може да бъде достъпен само ако потребителят е автентикиран, като това е постигнато чрез **canActivate** опцията и **AuthenticationGuard**.
* **path: 'search'** - който съответства на *SearchComponent* и може да бъде достъпен само ако потребителят е автентикиран, като това е постигнато чрез **canActivate** опцията и **AuthenticationGuard**.
* **path: 'register'** - който съответства на *RegisterComponent* и може да бъде достъпен от всеки потребител.
* **path: 'login'** - който съответства на *LoginComponent* и може да бъде достъпен от всеки потребител.
* **path: 'posts/:ID'** - който съответства на *DetailsComponent* и може да бъде достъпен само ако потребителят е автентикиран, като това е постигнато чрез **canActivate** опцията и **AuthenticationGuard**. Използва се :ID за да се предаде идентификатор на публикацията като параметър.
* **path: ‘‘** - който съответства на *LandingPageComponent* и може да бъде достъпен от всеки потребител.
* **path: 'new-listing'** - който съответства на *AddListingComponent* и може да бъде достъпен само ако потребителят е автентикиран, като това е постигнато чрез **canActivate** опцията и **AuthenticationGuard**.
* **path: '\*\*'** - който съответства на всички други пътища и пренасочва към началната страница, дефинирана с празен път - path: ''.

Нека хвърлим и един бърз поглед на валидациите, които са използвани за формите в приложението:



Фигура 33: Валидации

Виждаме, че имаме различни валидации, които са под формата на Regex – имаме валидация за име, за потребителско име, за имейл и за парола. Те се използват както при регистрацията на потребител в приложението, така и при неговата автентикация.

# Глава 4. Ръководство за потребителя

Source Code-ът на проекта може да бъде открит на следното Github хранилище:

[https://github.com/AlexanderSvl/CarTradeWebsite](https://github.com/AlexanderSvl/CarTradeWebsite%20)

За да стартирате приложението, трябва да изпълните следните стъпки:

**Backend**: Отворете файла с разшиерение .*sln* и стартирайте проекта от **Visual Studio.**

**Frontend**: Изпълнете командата *npm* *install* в папката /*src,* за да инсталирате необходимите *npm* пакети, и след това изпълнете командата *ng serve*, за да стартирате приложението.

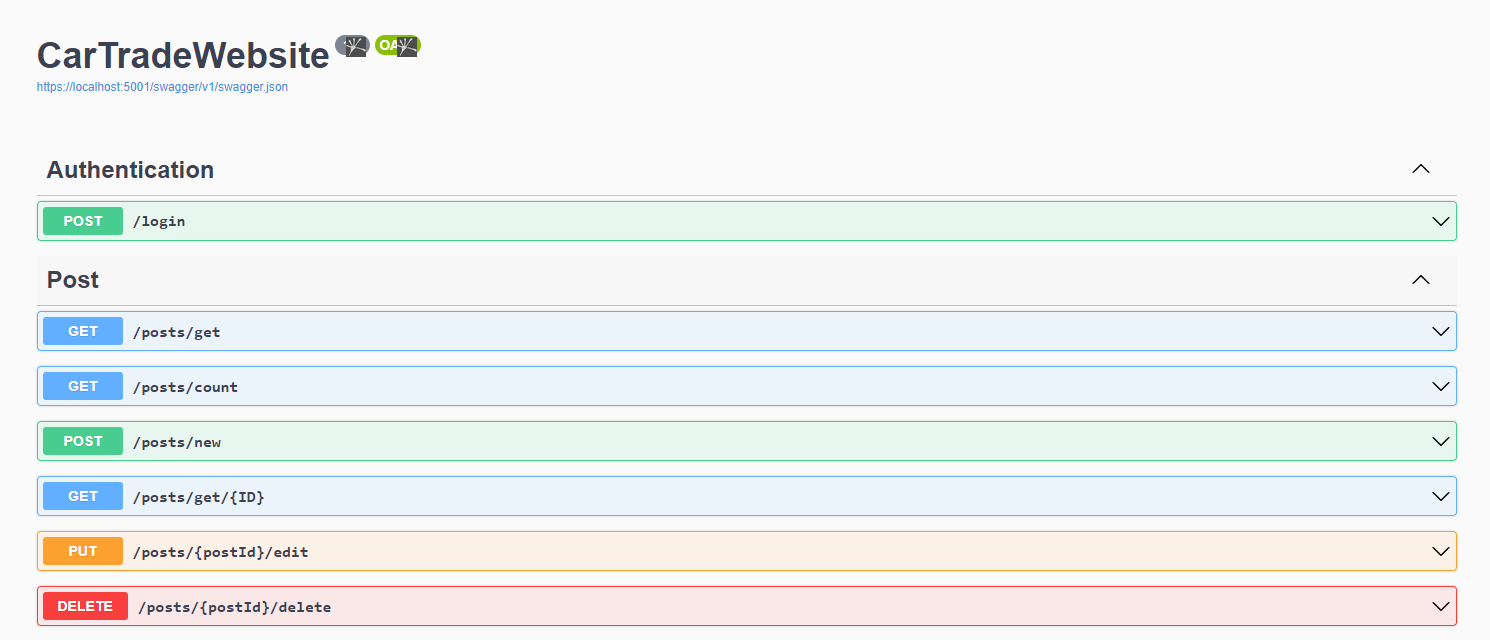
## 4.1 Swagger

Swagger UI, част от Swagger, е инструмент с отворен код, който генерира уеб страница, която документира API-то. Това представяне на API-то в потребителския интерфейс е удобно за потребителя и лесно за разбиране, като цялата сложност на логиката остава зад екрана.

Цялата идея на Swagger е да улесни визуализацията на едно уеб приложение за разработчиците, когато то е все още в етап на раработка на backend-a – така се улеснява изпращането на заявки, получаването на отговор от сървъра и тн.

Документацията на Swagger е обширна и инструментът е лесен и приятен за използване, поради което винаги го включвам, когато разработвам уеб приложение.

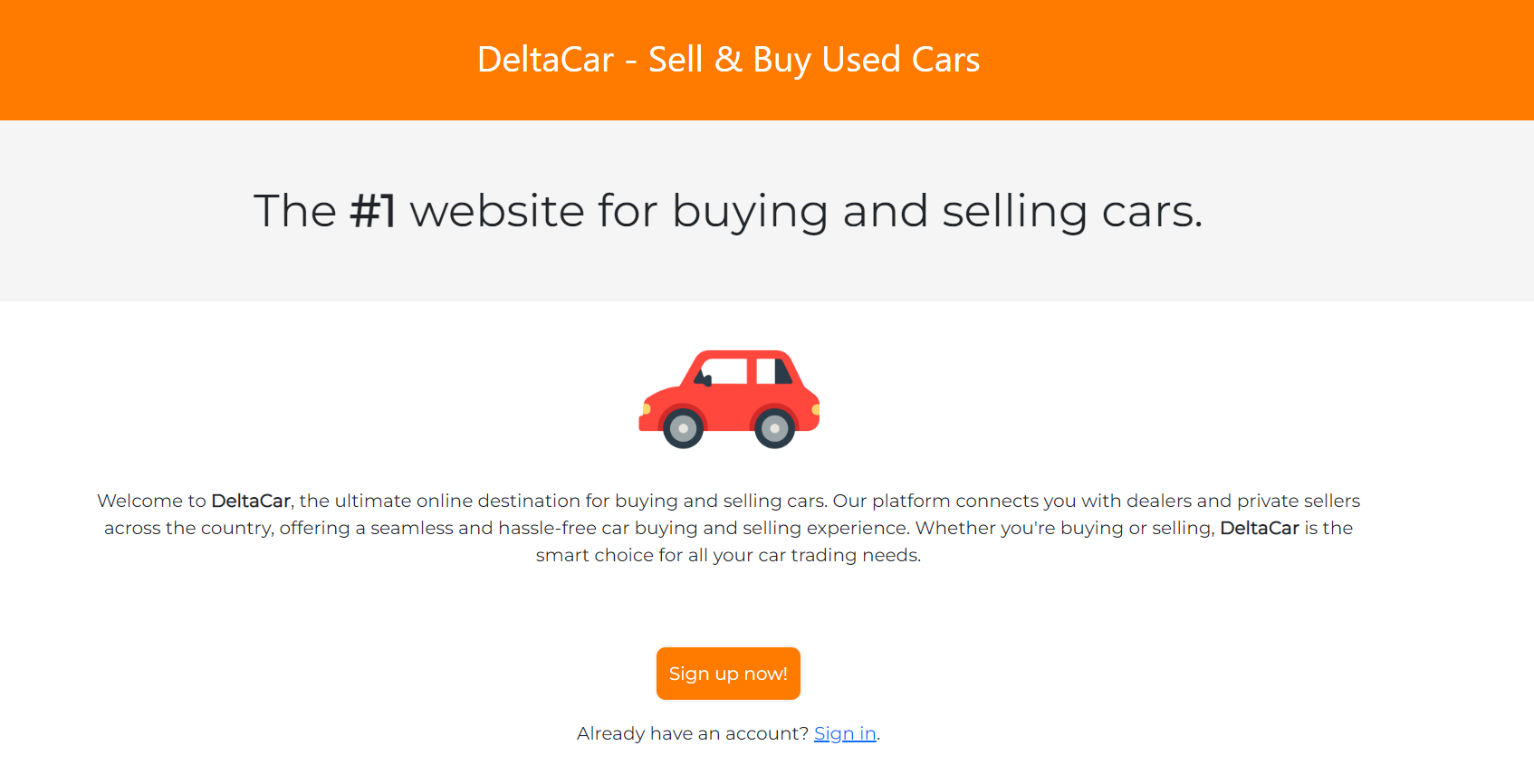
В текущия проект Swagger изглежда по следния начин:



Фигура 34: SwaggerUI

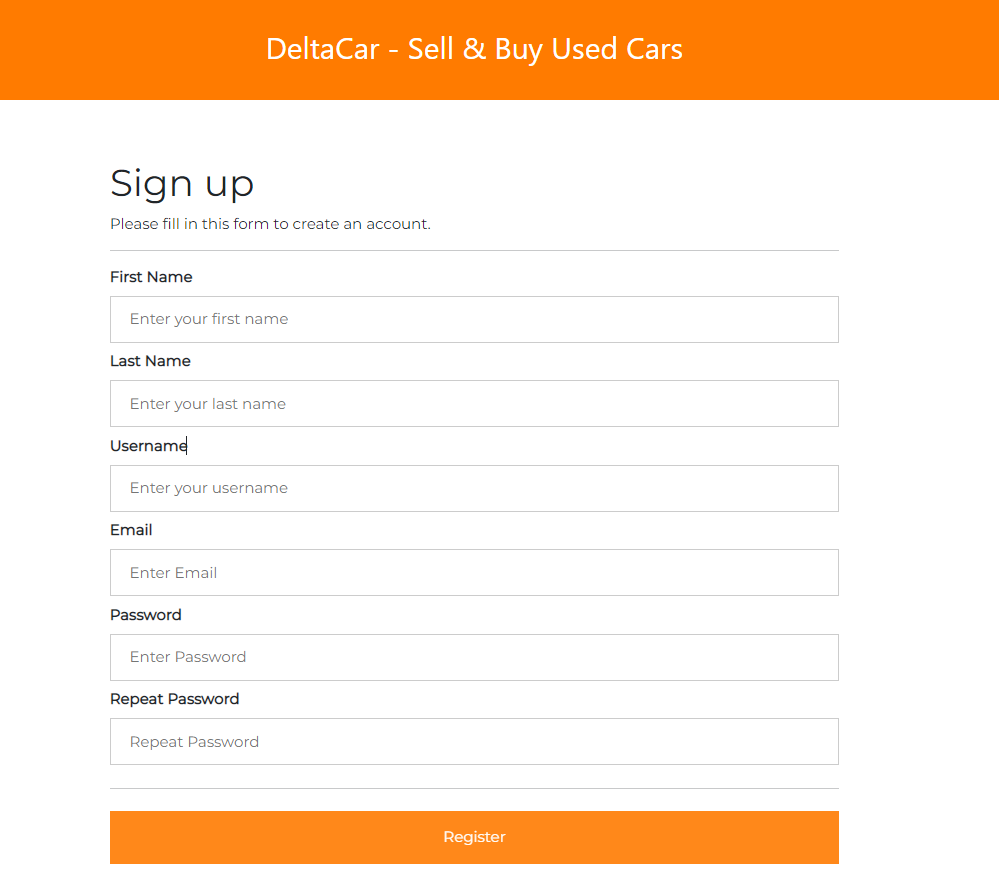
## 4.2 Frontend

След стартиране на frontend-a, ще попаднете на началната страница на уеб приложението:



Тук имате възможност да прочетете за сайта и неговата мисия.

След натискане на бутона в оранжево, ще имате възможност да се регистрирате или автентицирате в приложението:

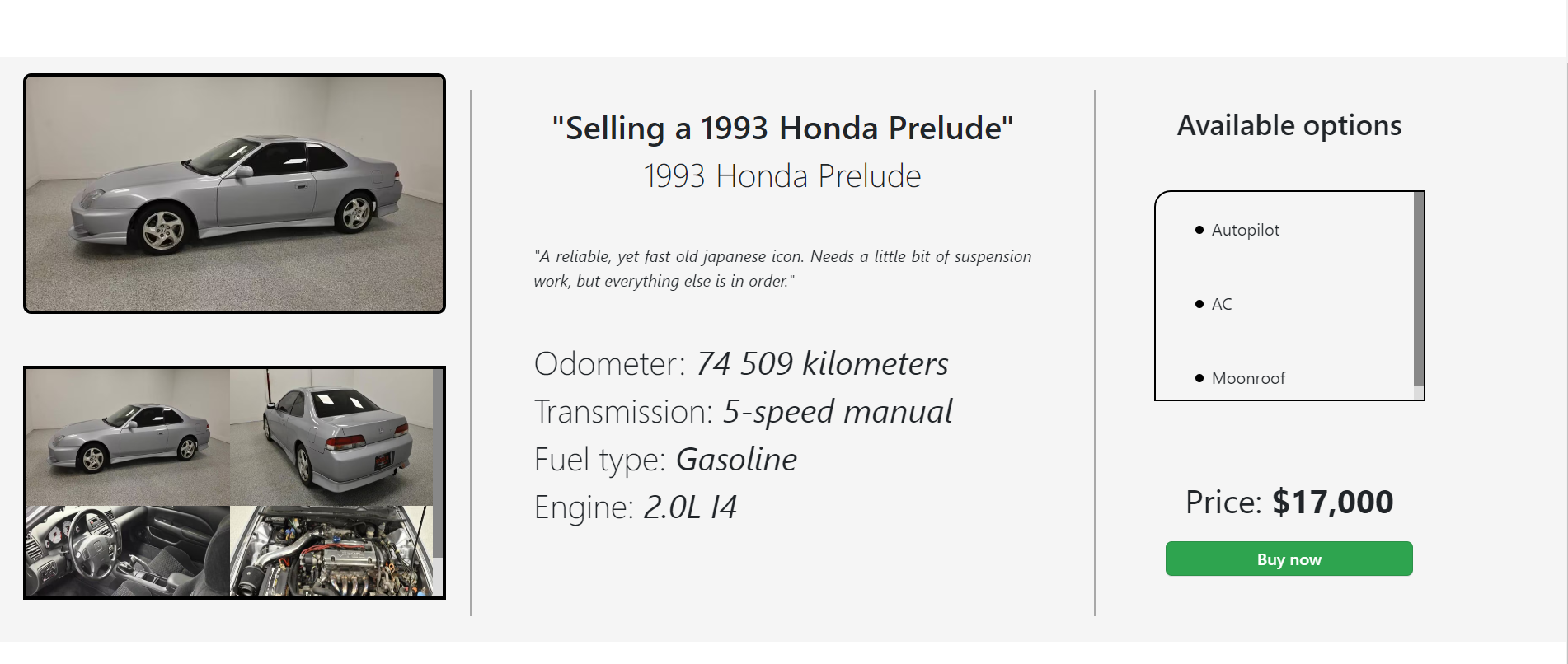


След успешна регистрация или автентикация, ще бъдете изпратени на началната страница на сайта, където ще можете да разгледате обявите, които са налични.

В горната част на екрана се намира т.нар Navigation bar, който позволява лесното навигиране из различните страници в приложението:



Една обява изглежда по следния начин:



Виждаме, че информацията е достатъчна, снимките са хубави, имаме цена и бутон за покупка.

# 

# Заключение

Уеб приложението DeltaCar е разработено добре, като са следвани всички приети принципи в програмирането. Кодът е лесен за разбиране, добре описващ себе си. Страниците са лесни за усвояване, UI-я и UX-a са на ниво.

Като цяло смятам, че DeltaCar е един достоен проект и ще бъде продължен да бъде разработван, като някои от бъдещите feature-и ще са:

* **Бърз и усъвършенстван Search**
* **Pager на обявите**
* **Разглеждане на обяви по дата на добавяне**
* **Връзка с продавач / автокъща**

И други...

# Информационни източници

* <https://angular.io/docs>
* <https://learn.microsoft.com/en-us/ef/core>
* <https://stackoverflow.com>
* <https://github.com/>
* <https://swagger.io/docs>

# Рецензия на дипломен проект

|  |  |  |  |
| --- | --- | --- | --- |
| **Тема на дипломния проект** |  | | |
| **Ученик** |  | | |
| **Клас** |  | | |
| **Професия** |  | | |
| **Специалност** |  | | |
| **Ръководител- консултант** |  | | |
| **Рецензент** |  | | |
| **Критерии за допускане до защита на дипломен проект** | | **Да** | **Не** |
| Съответствие на съдържанието и точките от заданието | |  |  |
| Съответствие между тема и съдържание | |  |  |
| Спазване на препоръчителния обем на дипломния проект | |  |  |
| Спазване на изискванията за оформление на дипломния проект | |  |  |
| Готовност за защита на дипломния проект | |  |  |
| Силни страни на дипломния проект |  | | |
| Допуснати основни слабости |  | | |
| Въпроси и препоръки към дипломния проект |  | | |

**ЗАКЛЮЧЕНИЕ:**

Качествата на дипломния проект дават основание ученикът/ ученичката ................................................... да бъде допуснат/а до защита пред членовете на комисията за подготовка, провеждане и оценяване на изпит чрез защита на дипломен проект- част по теория на професията.

.........05.2023г. Рецензент:................................

град София