



**ПРОФЕСИОНАЛНА ГИМНАЗИЯ ПО  
ЕЛЕКТРОТЕХНИКА И ЕЛЕКТРОНИКА – гр. Пловдив**

**професия код 481030 „Приложен програмист“  
специалност код 4810301 „Приложно програмиране“**

**ДИПЛОМЕН ПРОЕКТ  
за придобиване трета степен професионална квалификация**

**Тема: Електронен магазин „AutoPoint“**

**Дипломант:** Самуил Добрински

**Клас:** 12а

**e-mail:** msamuil@abv.bg

**Ръководител-консултант:** инж. Мариана Христозкова

Пловдив  
2023

# СЪДЪРЖАНИЕ

УВОД .....	1
ГЛАВА 1 – ТЕХНОЛОГИИ И КОНЦЕПЦИИ .....	3
1.1. Технологични рамки.....	3
1.2. ASP.NET Core.....	3
1.3. MVC .....	6
1.4. Базы данни .....	8
1.5. Bootstrap .....	9
1.6. Newtonsoft.Json.....	11
1.6.1. Json.....	11
1.7. EntityFramework .....	13
1.8. Script.NET .....	13
1.9. Езици .....	14
1.10. Развойна среда.....	15
1.11. API .....	15
1.12. Flutter .....	15
ГЛАВА 2 – СТРУКТУРА НА СОФТУЕРНОТО ПРИЛОЖЕНИЕ .....	17
2.1. ViewModel.....	17
2.2. Views .....	18
2.3. Controllers.....	19
2.4. Entity .....	20
2.5. DataBaseAccess .....	21
2.6. Repository .....	22
2.7. Tools.....	22
2.8. Resources .....	23

ГЛАВА 3 – СТРУКТУРА НА МОБИЛНОТО ПРИЛОЖЕНИЕ.....	25
3.1. Base.....	25
3.2. Controllers.....	25
3.3. Data .....	26
3.4. Helpers .....	26
3.5. Models.....	27
3.6. Pages .....	27
3.7. Routes.....	28
3.8. Utils.....	29
3.9. Widgets .....	29
ГЛАВА 4 ПРОГРАМНА РЕАЛИЗАЦИЯ НА ВСЕКИ ОТ МОДУЛИТЕ.....	30
4.1. База данни .....	30
4.2. Интерфейс.....	31
4.3. Интегриране на API .....	32
4.4. Разплащане .....	34
4.5. Описание на процеса на разработка на уеб апликацията.....	34
4.6. Описание на процеса на разработка на услугата (API) .....	36
4.7. Описание на процеса на разработка мобилното приложение .....	36
ГЛАВА 5 РЪКОВОДСТВО ЗА ПОТРЕБИТЕЛЯ НА УЕБ ПРИЛОЖЕНИЕТО.....	39
5.1. Навигация в приложението .....	39
5.2. Бутон Езици .....	39
5.3. Бутон Начало .....	39
5.4. Бутон Продукти.....	40
5.5. Бутон DTC четец .....	41
5.6. Бутон Двигатели.....	42

5.7. Бутон Контакти .....	43
5.8. Количка .....	44
5.9. Търсачка.....	44
5.10. Бутон Акаунт .....	45
5.11. Вход.....	45
5.12. Регистрация .....	46
5.13. Профил.....	46
ГЛАВА 6 РЪКОВОДСТВО ЗА ПОТРЕБИТЕЛЯ НА МОБИЛНОТО ПРИЛОЖЕНИЕ.....	47
6.1. Главна страница на приложението.....	47
6.2. Страници за описание на продукт.....	47
6.3. Потребителската количка с продукти.....	48
6.4. Страница за потвърдена поръчка .....	49
6.5. Страница моите поръчки.....	50
6.6. Страница желани продукти.....	50
6.7. Профилна страница.....	51
6.9. Страница за задаване на адрес .....	53
6.10. Страница промяна на профила .....	54
ЗАКЛЮЧЕНИЕ.....	55
ИЗТОЧНИЦИ .....	57
ПРИЛОЖЕНИЕ 1.....	59
ПРИЛОЖЕНИЕ 2.....	62
ПРИЛОЖЕНИЕ 3.....	63

## УВОД

С течение на времето, с разрастването на интернет услугите се увеличава и броят налични електронни магазини. Тяхното създаване е все по-достъпно дори за непрофесионалисти, които се насочват към Content Management системи, като например WordPress, OpenCart и други. За съжаление обаче, този тип електронни магазини не разполагат с всички нужни за качествена работа възможности. Тези електронни магазини обикновено са за стандартни стоки и дори в момента, търсенето на специфични продукти, например тунинг авточасти все още е изключително трудно за потребителя. Дори и да има сайтове, които предоставят подобен тип продукти, то те не са с високо качество, не са интуитивни за достъп от потребителя, а още по-малко разполагат с необходимите функционалности за търсене на съответното необходимо действие.

Цел на настоящата разработка е да разреши поставеният по-горе проблем и да разработи многофункционално уеб приложение на електронен магазин за специфични авточасти, в това число и за компоненти за тунингване на множество двигатели. Този електронен магазин трябва да съдържа :

- Достъп посредством администраторски и потребителски профил
- Работа с асортимента
- Специфични дейности

Целта е AutoPoint да бъде електронен магазин за авточасти и тунинг компоненти със специфичен дизайн и интуитивен интерфейс, в който потребителите освен специфичните за всеки един електронен магазин дейности по търсене, харесване, закупуване на продукти, да могат да извършват и допълнителни дейности. В AutoPoint трябва да бъдат имплементирани функционалности за проверка на спецификациите на определен двигател, проверка на конкретни стойности, получени от OBD модул (електронен модул за инспектиране на състоянието на автомобила). Приложението трябва да поддържа различни нива на достъп – администраторски и потребителски, чрез възможност за създаване и редактиране на профили и изграждане на съответни платежни методи.

Задачи за изработка на проекта :

1. Да се проучат необходимите за реализацията технологии и да се осъществи мотивиран избор.
2. Трябва да бъде изработен външният вид на сайта.
3. Трябва да бъде изграден самия магазин за тунинг авточасти.
4. Трябва да бъдат интегрирани дейности за проверка на двигателите и стойностите на OBD модула.
5. Трябва да се изгради система от различни нива на достъп с потребителски профили и администраторски контрол.
6. Трябва да бъде изградена система за множество разплащания.
7. В съответните потребителски и администраторски панели да има достъп за преглеждане, редактиране на отделните функционалности.
8. Да се разработи мобилно приложение, което улеснява достъпа до електронния магазин през мобилни устройства.

# ГЛАВА 1 – ТЕХНОЛОГИИ И КОНЦЕПЦИИ

## 1.1. Технологични рамки

За постигане на поставената от увода цел и за решаване на задачите е нужно първоначално да се проучат възможните технологични рамки. Съвременни качествени уеб приложения най-често се изграждат на базата на Angular, React, Content Management Systems и др. Всяка една от тях разполага със специфични позитиви и недостатъци. Например React разполага с виртуален DOM за бърза работа в документа, Angular разполага с подобрена производителност на сървъра и т.н. Въпреки това всички от тези среди имат своите минуси и лимитации. За изработката на проекта е използван ASP.NET, защото от всички среди, разполага с най-богата документация, силна back-end част и подпомогнат front-end от технологии като Razor, Blazor, AJAX и т.н.

## 1.2. ASP.NET Core



*Фигура 1 Лого на ASP.NET<sup>1</sup>*

На фигура 1 може да се види логото на ASP.NET. Това е софтуерна рамка с отворен код, разработена от Microsoft. Тя работи под Windows и се използва от софтуерните инженери за създаване на уеб апликации, услуги и динамични уебсайтове. ASP.NET Core е новата версия на ASP.NET която за разлика от предишните, работи на всички операционни системи и значително надражда възможностите им. Тази рамка е безплатна, има много поддържани езици и богата

---

<sup>1</sup> Ankita Singh, „ASP.NET: An Overview“, 22/08/2018г., прегледан на 18/01/2023г., достъпен на <https://msatechnosoft.in/blog/asp-net-architecture-life-cycle-events-web-development/>

документация, както и дава пълен контрол над работата и може да се използва както за големи, така и за малки проекти<sup>2</sup>.

ASP.NET Core предоставя следните предимства:

- Единна среда за изграждане на уеб интерфейс и уеб API.
- Подходяща е за тестване.
- Blazor ви позволява да използвате C# в браузъра заедно с JavaScript. Споделя логиката на приложението от страна на сървъра и от страна на клиента, всички написани с .NET.
- Възможност за разработка и работа под Windows, macOS и Linux.
- С отворен код и фокусиран върху общността.
- Интегриране на модерни, клиентски рамки и работни процеси за разработка.
- Вграден Dependency injection.
- Лек, високопроизводителен и модулен конвейер за HTTP заявки.
- Паралелно създаване на версии.
- Инструменти, които опростяват модерното уеб развитие<sup>3</sup>.

ASP.NET поддържа три основни модела за разработка: уеб страници, уеб формуляри и MVC (Model View Controller) и разполага с инструменти като Razor, Blazor, AJAX и т.н.

### **1.1.1. Razor**

Razor е формат за генериране на текстово съдържание, като HTML. Файловете на Razor имат cshtml или файлово разширение на razor и съдържат комбинация от C# код заедно с HTML<sup>4</sup>.

---

<sup>2</sup> Янев Румен, „Какво е ASP.NET Core и защо да я научим?“ 05/10/2020г., прегледан на 18/01/2023г., достъпен на <https://softuni.bg/blog/asp-net-core-article>

<sup>3</sup> Microsoft, „Overview of ASP.NET Core“, 15/11/2022г., прегледан на 18/01/2023г., достъпен на <https://learn.microsoft.com/bg-bg/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-3.1>

<sup>4</sup> Microsoft, „Introduction to Web Development with Blazor“, началото на 2023г., прегледан на 18/01/2023г., достъпен на <https://learn.microsoft.com/bg-bg/training/modules/blazor-introduction/2-what-is-blazor>



### 1.1.2. Blazor

Blazor е рамка с потребителски интерфейс за .NET, която може да бъде използвана за създаване на приложения. Може да изпълнява C# код директно в брауъра, използвайки WebAssembly. Работи в същата защитена затворена среда (sandbox) като JavaScript рамки като Angular, React, Vue и т.н. На практика, помощта на WebAssembly, в брауъра може да се изпълнява не само C#, а всякакъв тип код<sup>5</sup>.

### 1.1.3. AJAX

Това е техника за създаване на бързи и динамични уеб страници. AJAX позволява уеб страниците да се актуализират асинхронно чрез обмен на малки количества данни със сървъра. Това означава, че е възможно да актуализирате части от уеб страница, без да презареждате цялата страница. Класическите уеб страници (които не използват AJAX) трябва да презаредят цялата страница, ако съдържанието се промени. Примери за приложения, използващи AJAX: раздели Google Maps, Gmail, Youtube и Facebook<sup>6</sup>.

### 1.1.4. ASPX (Web Form Engine)

ASPX или Web Form Engine е машината за преглед по подразбиране за ASP.NET, която е включена в ASP.NET MVC от самото начало.

Синтаксисът, използван за писане на изглед с ASPX View Engine, е същият като синтаксиса, използван в уеб формулярите на ASP.NET. Файловете разширения също са същите като за уеб формуляри на ASP.NET (като .aspx, .ascx, .master)<sup>7</sup>.

ASPX използва "<% = %>" или "<% : %>" за изобразяване на съдържание от страна на сървъра. Пространството от имена за Webform

---

<sup>5</sup> Pragimtech, „What is Blazor“, 18/04/2020г., прегледан на 18/01/2023г., достъпен на <https://www.pragimtech.com/blog/blazor/what-is-blazor/>

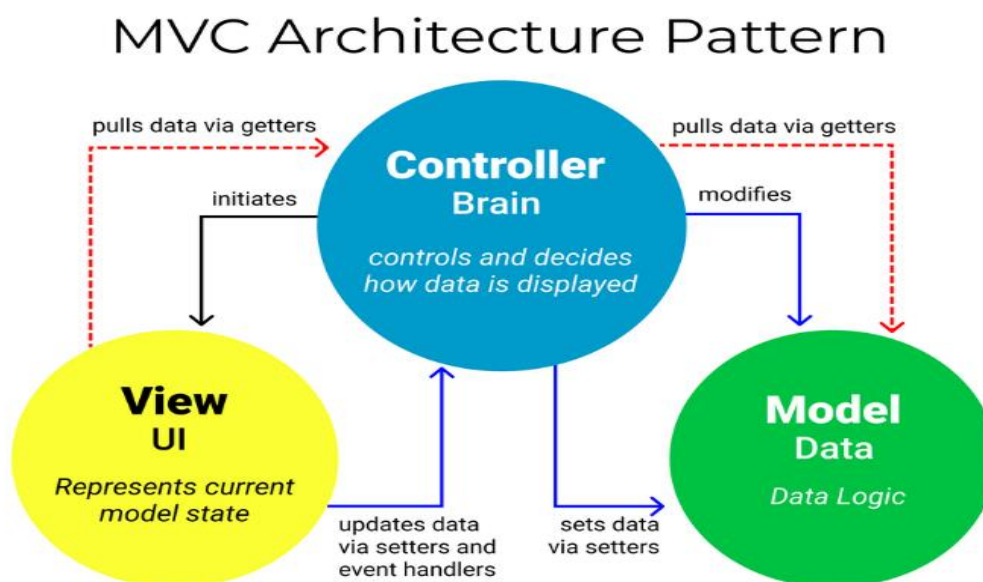
<sup>6</sup> W3schools, „ASP AJAX“, 14/07/2020г., прегледан на 18/01/2023г., достъпен на [https://www.w3schools.com/asp/asp\\_ajax.asp](https://www.w3schools.com/asp/asp_ajax.asp)

<sup>7</sup> TutorialsPoint, „ASP.NET WP – View Engines“, 25/05/2013г., прегледан на 18/01/2023г., достъпен на [https://www.tutorialspoint.com/asp.net\\_wp/asp.net\\_wp\\_view\\_engines.htm](https://www.tutorialspoint.com/asp.net_wp/asp.net_wp_view_engines.htm)

Engine е System.Web.Mvc.WebFormViewEngine. Той е сравнително по-бърз от Razor View Engine<sup>8</sup>.

### 1.3. MVC

MVC е абривиатура за „ Model – View – Controller “ или по-точно представлява архитектурен шаблон при програмния дизайн. Използва се за разделението на бизнес логиката на три части: “Model“, “View” и “Controller”. Model-ът представлява „ядрото“ на приложението. Този модел записва всички данни които ще бъдат манипулирани. View представлява изходната част на софтуера, визуализираща обработените данни и Controller-ът, който представлява библиотеката на този шаблон. Всички методи и операции за манипулиране на данните от модела биват извършвани в него<sup>9</sup>.



Фигура 2. Взаимодействие на отделните елементи на MVC архитектурата

От фигура 2 може да се види как отделните сегменти от тази архитектура взаимодействат един с друг. Първо потребителят изпраща заявка към контролера. Там тази заявка бива обработена, като данните от модела биват извлечени и

<sup>8</sup> TutorialsPoint, „ASP.NET WP – View Engines“, 25/05/2013г., прегледан на 18/01/2023г., достъпен на [https://www.tutorialspoint.com/asp.net\\_wp/asp.net\\_wp\\_view\\_engines.htm](https://www.tutorialspoint.com/asp.net_wp/asp.net_wp_view_engines.htm)

<sup>9</sup> Георги Кацаров, „Какво е ASP.NET Core и какво е MVC“, 11/06/2018г., прегледан на 18/01/2023г., достъпен на <https://softuni.bg/blog/asp-dot-net-core-and-mvc>

модифицирани, след което модифицираните данни биват изпратени обратно към съответното “View” , което потребителят вижда и може да изпрати нова заявка.

### **1.3.2. Model**

Компонентът Model съответства на цялата свързана с данните логика, с която работи потребителят. Това може да представлява или данните, които се прехвърлят между компонентите View и Controller, или всякакви други данни, свързани с бизнес логиката. Например, клиентски обект ще извлече информацията за клиента от базата данни, ще я манипулира и ще я актуализира обратно в базата данни или ще я използва за изобразяване на данни<sup>10</sup>.

### **1.3.3. View**

Компонентът View се използва за цялата логика на потребителския интерфейс на приложението. Например изгледът на клиента ще включва всички компоненти на потребителския интерфейс, като текстови полета, падащи менюта и т.н., с които взаимодейства крайният потребител<sup>11</sup>.

### **1.3.4. Controller**

Контролерите действат като интерфейс между компонентите на модела и изгледа, за да обработват цялата бизнес логика и входящи заявки, да манипулират данните с помощта на компонента на модела и да взаимодействат с изгледите, за да изобразят крайния изход. Например контролерът на клиента ще обработва всички взаимодействия и входи от изгледа на клиента и ще актуализира базата данни с помощта на модела на клиента. Същият контролер ще се използва за преглед на данните на клиента<sup>12</sup>.

---

<sup>10</sup> TutorialsPoint, „MVC Framework – Introduction“, 26/06/2022г., прегледан на 18/01/2023г., достъпен на [https://www.tutorialspoint.com/mvc\\_framework/mvc\\_framework\\_introduction.htm](https://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm)

<sup>11</sup> TutorialsPoint, „MVC Framework - Introduction“, 26/06/2022г., прегледан на 18/01/2023г., достъпен на [https://www.tutorialspoint.com/mvc\\_framework/mvc\\_framework\\_introduction.htm](https://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm)

<sup>12</sup> TutorialsPoint, „MVC Framework - Introduction“, 26/06/2022г., прегледан на 18/01/2023г., достъпен на [https://www.tutorialspoint.com/mvc\\_framework/mvc\\_framework\\_introduction.htm](https://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm)

### 1.3.5. ASP.NET MVC framework

ASP.NET MVC framework е лека презентационна рамка с възможност за тестване, която е интегрирана със съществуващите функции на ASP.NET, като главни страници, удостоверяване и т.н. В рамките на .NET тази рамка е дефинирана в колекция System.Web.Mvc. Последната версия на MVC Framework е 5.0. Използваме Visual Studio за създаване на ASP.NET MVC приложения, които могат да бъдат добавени като шаблон във Visual Studio<sup>13</sup>.

### 1.4. Базы данни

Базата данни е информация, която е създадена за лесен достъп, управление и актуализиране. Компютърните бази данни обикновено съхраняват съвкупности от записи на данни или файлове, които съдържат информация, като транзакции за продажба, данни за клиенти, финансови данни и информация за продукта. Базите данни се използват за съхраняване, поддържане и достъп до всякакъв вид данни. Те събират информация за хора, места или неща. Тази информация се събира на едно място, за да може да се наблюдава и анализира. Базите данни могат да се разглеждат като организирана колекция от информация<sup>14</sup>. Базите данни биват два вида – релационни и нерелационни. На фигура 3 е визуализиран пример с представители на различни видове бази данни.



Фигура 3 Примери за бази от данни<sup>15</sup>

<sup>13</sup> TutorialsPoint, „MVC Framework - Introduction“, 26/06/2022г., прегледан на 18/01/2023г., достъпен на [https://www.tutorialspoint.com/mvc\\_framework/mvc\\_framework\\_introduction.htm](https://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm)

<sup>14</sup> Ben Lutkovich, „database (DB)“, 15/04/2009г., прегледан на 18/01/2023г., достъпен на <https://www.techtarget.com/searchdatamanagement/definition/database>

<sup>15</sup> Sahiti Kappagantula, „Differences Between SQL & NoSQL Databases“, 14/05/2020г., прегледан на 18/01/2023г., достъпен на <https://www.edureka.co/blog/sql-vs-nosql-db/>

### 1.4.2. Релационни бази данни

Релационна база данни е тип база данни, която съхранява и предоставя достъп до отделни единици от данни, които са свързани една с друга. Релационните бази данни се основават на релационния модел, интуитивен, ясен начин за представяне на данни в таблици. В релационна база данни всеки ред в таблицата е запис с уникален идентификатор, наречен ключ. Колоните на таблицата съдържат атрибути на данните и всеки запис обикновено има стойност за всеки атрибут, което улеснява установяването на връзките между точките от данни<sup>16</sup>.

### 1.4.3. Нерелационни бази данни (NoSQL)

NoSQL базите данни (известни още като „не само SQL“) са нетаблични бази данни и съхраняват данни по различен начин от релационните таблици. NoSQL базите данни се предлагат в различни типове въз основа на техния модел на данни. Основните типове са документ, Key-Value, Wide-Column и graph. Те предоставят гъвкави схеми и се мащабират лесно с големи количества данни и големи потребителски натоварвания<sup>17</sup>.

## 1.5. Bootstrap



*Фигура 4 Лого на Bootstrap*

---

<sup>16</sup> Oracle, „What is a Relational Database (RDBMS) ?“, началото на 2023г., прегледан на 18/01/2023г., достъпен на <https://www.oracle.com/database/what-is-a-relational-database/>

<sup>17</sup> MongoDB, „What is NoSQL?“, началото на 2023г., прегледан на 18/01/2023г., достъпен на <https://www.mongodb.com/nosql-explained>

На фигура 4 е представено логото на Bootstrap, което е безплатна рамка за front-end разработка с отворен код за създаване на уебсайтове и уеб приложения. Създаден, за да даде възможност за бързо реагиране на уеб сайтове, ориентирани към мобилни устройства, Bootstrap предоставя колекция от синтаксис за дизайн на шаблони. Като рамка, Bootstrap включва основите за адаптивна уеб разработка, така че разработчиците трябва само да вмъкнат кода в предварително дефинирана мрежова система. Рамката Bootstrap е изградена върху Hypertext Markup Language (HTML), cascading style sheets (CSS) и JavaScript. Уеб разработчиците, използващи Bootstrap, могат да създават уебсайтове много по-бързо, без да губят време да се тревожат за основни команди и функции<sup>18</sup>.

Bootstrap дава възможност на уеб страница или приложение да разпознават размера и ориентацията на екрана на посетителя и автоматично да адаптират дисплея към тях. Подходът на първо място за мобилни устройства предполага, че смартфоните, таблетите и мобилните приложения за конкретни задачи са основните инструменти на служителите за извършване на работа. Bootstrap отговаря на изискванията на тези технологии в дизайна и включва UI компоненти, оформления, JavaScript инструменти и рамката за изпълнение. Софтуерът се предлага предварително компилиран или като изходен код<sup>19</sup>.

В момента в интернет има голям набор от Bootstrap шаблони, с които разработчик бързо и лесно може да имплементира дизайн на дадения уебсайт или уеб приложение. Използването на подобен тип базови ресурси е изключително достъпно и удобно. Необходимо е да бъде изтеглен шаблона и да бъдат добавени нужните CSS и JavaScript класове, без които шаблона не може да функционира. След това разработчика може да копира готовия Html код и да го постави в неговия интерфейс, като този код може да бъде изменен по всякакъв начин за да бъде направен нужния дизайн на приложението. Корекциите могат да бъдат:

- Да бъде разместено мястото на дадени сегменти – разместено място на долен колонтитул (footer), търсачка, навигационно меню и др.

---

<sup>18</sup> Andrew Zola, „What is Bootstrap?“, 08/08/2022г., прегледан на 18/01/2023г., достъпен на <https://www.techtarget.com/whatis/definition/bootstrap>

<sup>19</sup> Andrew Zola, „What is Bootstrap?“, 08/08/2022г., прегледан на 18/01/2023г., достъпен на <https://www.techtarget.com/whatis/definition/bootstrap>

- Промяна на самите сегменти – промяна на цвета, формата, размера и др.
- Да бъдат премахнати цели сегменти

## 1.6. Newtonsoft.Json

### 1.6.1. Json

JSON (JavaScript Object Notation) е олекотен формат за обмен на данни, лесно четим и лесно се пише на него, както и лесен за машините да го анализират и генерират. Базиран е на подмножество от стандарта за език за програмиране на JavaScript. JSON е текстов формат, който е напълно независим от езика, но използва конвенции, които са познати на програмистите от C-семейството езици, включително C, C++, C#, Java, JavaScript, Perl, Python и много други. Тези свойства правят JSON идеален език за обмен на данни<sup>20</sup>.

JSON е изграден върху две структури:

- Колекция от двойки key/value. В различни езици това се реализира като обект, запис, структура, речник, хеш таблица, списък с ключове или асоциативен масив.
- Подреден списък от стойности. В повечето езици това се реализира като масив, вектор, списък или последователност<sup>21</sup>.

Това са универсални структури от данни. На практика всички съвременни езици за програмиране ги поддържат под една или друга форма. Логично е формат на данни, който е взаимозаменяем с езиците за програмиране, също да се основава на тези структури<sup>22</sup>.

---

<sup>20</sup> Json.org, „Introducing JSON“, 15/04/2010г., прегледан на 18/01/2023г., достъпен на <https://www.json.org/json-en.html>

<sup>21</sup> Json.org, „Introducing JSON“, 15/04/2010г., прегледан на 18/01/2023г., достъпен на <https://www.json.org/json-en.html>

<sup>22</sup> Json.org, „Introducing JSON“, 15/04/2010г., прегледан на 18/01/2023г., достъпен на <https://www.json.org/json-en.html>

```
var jsonObj = {  
    "property1" : "value1",  
    "property2" : "value2",  
    "property3" : "value3",  
    .....  
    "property n" : "value n"  
};
```

*Фигура 5 Примерен Json обект*

От фигура 5 може да се види примерен Json обект. Вижда се как първо се задава името на обекта (property) и след това се отделят данните на този обект (value).

### **1.6.2. Newtonsoft.Json**

Newtonsoft.Json представлява Nu-Get пакет направен от Newtonsoft. Той разполага с класове и функции, които биват използвани за обработка на JSON форматът. JSON е съкращението на JavaScript Object Notation или по-точно представлява текстово базиран отворен стандарт създаден за човешки четим обмен на данни. Nu-Get пакетът поддържа функции с които може даден обект да бъде конвертиран(сериализиран) в JSON форматът както и десериализиран от форматът към обект, както и за създаване на нов обект от дадения формат.



## 1.7. EntityFramework



*Фигура 6 Лого на EntityFramework<sup>23</sup>*

На фигура 6 е изобразено логото на EntityFramework, което е Nu-Get пакет изработен от Microsoft. Entity Framework е ORM (Object-relational mapping) рамка с отворен код за .NET приложения, поддържани от Microsoft. Той позволява на разработчиците да работят с данни, използвайки обекти от специфични за домейна класове, без да се фокусират върху основните таблици и колони на базата данни, където се съхраняват тези данни. С Entity Framework разработчиците могат да работят на по-високо ниво на абстракция, когато работят с данни, и могат да създават и поддържат ориентирани към данни приложения с по-малко код в сравнение с традиционните приложения<sup>24</sup>.

## 1.8. Scrypt.NET

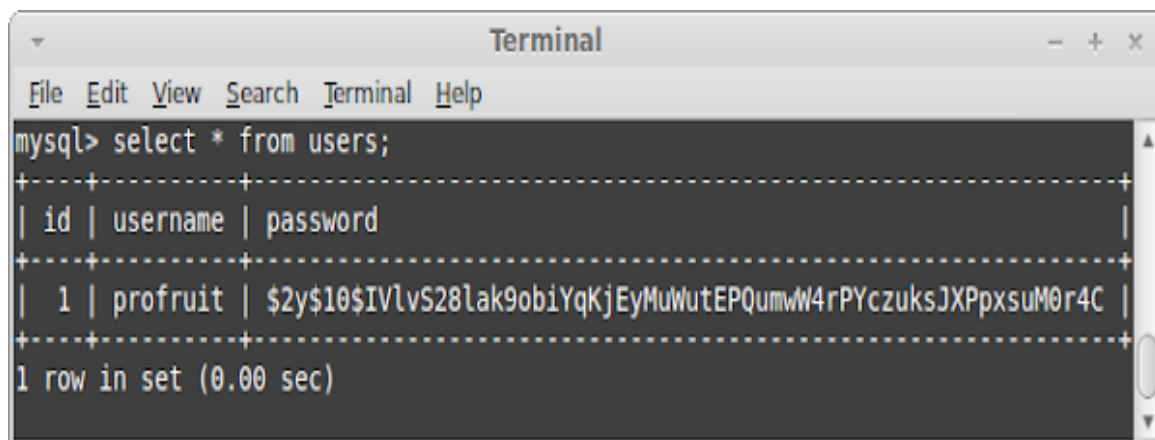
Scrypt.NET е Nu-Get пакет авторът на който е Vinicius Chiele. Този пакет разполага с алгоритъм за хеширане. Функциите и класовете му позволяват на разработчика да използва вградената му encode функция която приема даден низ и връща неговата кодирана (хеширана) версия. Този пакет се използва за да направи

---

<sup>23</sup> SimpliLeart, „What is C# Entity Framework? A Comprehensive Guide“, 10/13/2022г., прегледан на 18/01/2023г., достъпен на <https://www.simplilearn.com/tutorials/asp-dot-net-tutorial/entity-framework-in-c-sharp>

<sup>24</sup> Entity Framework Tutorial, „What is Entity Framework?“, 10/10/2018г., прегледан на 18/01/2023г., достъпен на <https://www.entityframeworktutorial.net/what-is-entityframework.aspx>

потребителските акаунти по-защитени, като кодира паролите на потребителите. Пример за хеширане може да бъде видян на фигура 7.



```
Terminal
File Edit View Search Terminal Help
mysql> select * from users;
+----+-----+-----+
| id | username | password |
+----+-----+-----+
| 1 | profruit | $2y$10$IVlvS28lak9obiYqKjEyMuWutEPQumwW4rPYczuksJXPpxsuM0r4C |
+----+-----+-----+
1 row in set (0.00 sec)
```

Фигура 7 Вид на хеширана парола<sup>25</sup>

## 1.9. Езици

Използвани програмни езици:

- C# – обектно-ориентиран език за програмиране, позволяващ разработка на приложения в .NET. Той е използван за основния backend на приложението<sup>26</sup>.
- HTML – основата на уеб съдържанието. Отговаря за дефинирането на презентационната структура на приложението<sup>27</sup>.
- CSS – език за описанието и стилизирането на уеб елементи. Шаблоните от Bootstrap, съдържащи CSS код, дават на приложението съвременен, отзивчив стил.
- JavaScript – обектно базиран, frontend, скриптов език за уеб приложения. Използва се за програмиране на поведението на уеб страници при изпълнение на условие<sup>28</sup>.

<sup>25</sup> Bingol, „Хеширане на пароли с сол в PHP 5.5“, 19/10/2013г., прегледан на 18/01/2023г., достъпен на <https://profruit.blogspot.com/2013/10/php-55.html?m=1>

<sup>26</sup> Microsoft, „A tour of the C# language“, 12/13/2022г., прегледан на 18/01/2023г., достъпен на <https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/>

<sup>27</sup> MDN, „HTML: HyperText Markup Language“, 17/01/2023г., прегледан на 18/01/2023г., достъпен на <https://developer.mozilla.org/en-US/docs/Web/HTML>

<sup>28</sup> MDN, „JavaScript“, 13/12/2022г., прегледан на 18/01/2023г., достъпен на <https://developer.mozilla.org/en-US/docs/Web/JavaScript>

- Dart – Dart е оптимизиран за клиента език за разработване на бързи приложения на всяка платформа. Неговата цел е да предложи най-продуктивния език за програмиране за мултиплатформена разработка.

### **1.10. Развойна среда**

Visual Studio, известен също като Microsoft Visual Studio и VS, е интегрирана среда за разработка за Microsoft Windows. Това е инструмент за писане на компютърни програми, уебсайтове, уеб приложения и уеб услуги. Той включва редактор на код, програма за отстраняване на грешки, инструмент за проектиране на GUI и дизайнер на схема на база данни и поддържа повечето основни системи за контрол на ревизиите. Предлага се както в безплатно издание „Community edition“, така и в платена комерсиална версия – “Profesional edition”<sup>29</sup>.

### **1.11. API**

API означава интерфейс за програмиране на приложения. В контекста на API думата Приложение се отнася до всеки софтуер с отделна функция. Интерфейсът може да се разглежда като договор за услуга между две приложения. Този договор определя как двамата комуникират помежду си чрез заявки и отговори. Тяхната API документация съдържа информация за това как разработчиците трябва да структурират тези заявки и отговори<sup>30</sup>.

### **1.12. Flutter**

Flutter е рамка за разработка на мобилни приложения с отворен код, създадена от Google за изграждане на високопроизводителни приложения за iOS и Android в една кодова база. Flutter предоставя бърз и изразителен начин за разработчиците да създават собствени приложения.

Flutter беше издаден през май 2017 г. и написан на езика C, C ++, Dart<sup>31</sup>.

---

<sup>29</sup> Computer Hope, “Visual Studio“, 06/07/2019г., прегледан на 18/01/2023г., достъпен на <https://www.computerhope.com/jargon/v/visual-studio.htm>

<sup>30</sup> Amazone, “What is an Api?”, Началото на 2023г., прегледан на 20/01/2023г., достъпен на <https://aws.amazon.com/what-is/api/>

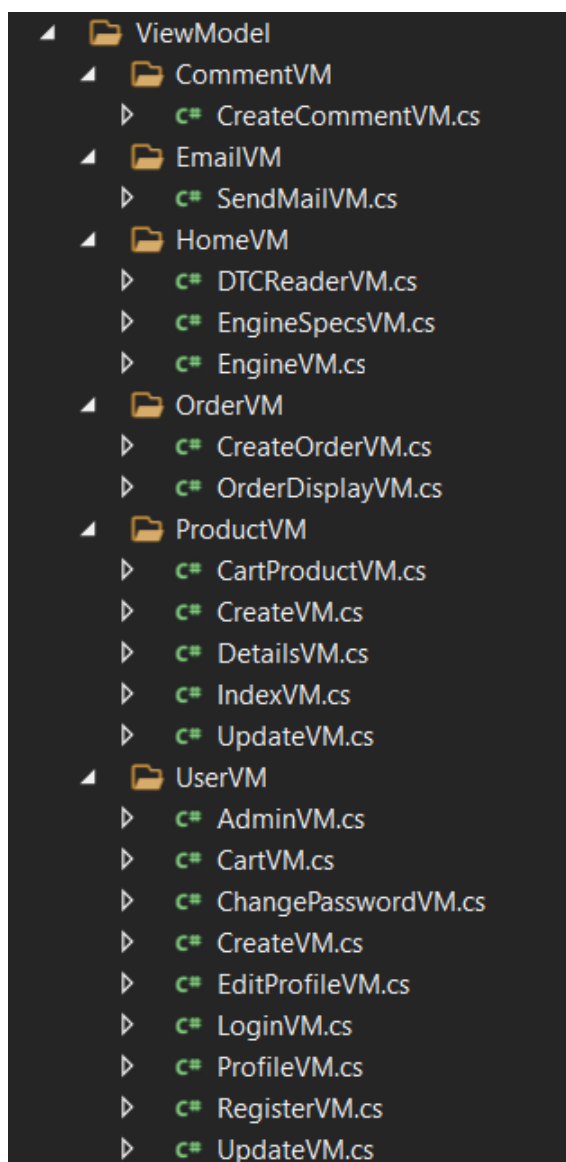
<sup>31</sup> Субодх Дармуан, “Основи на Flutter: защо да използваме Flutter за разработка на мобилни приложения”, 18/01/2020г., прегледан на 24/04/2023г., достъпен на <https://cynoteck.com/bg/blog-post/flutter-basics-why-use-flutter-for-mobile-app-development/>



## ГЛАВА 2 – СТРУКТУРА НА СОФТУЕРНОТО ПРИЛОЖЕНИЕ

### 2.1. ViewModel

Папката ViewModel съдържа всички модели които ще бъдат използвани за пренос на данни по страниците (Views). В контролерите тези модели биват инициализирани, запълвани с данни и изпращани до страниците. Освен това също самите страници могат да върнат в контролера цели модели, заредени с данни от form – ите, които съдържат в тях. Без тях обменът на данни между страниците и контролерите е невъзможен.

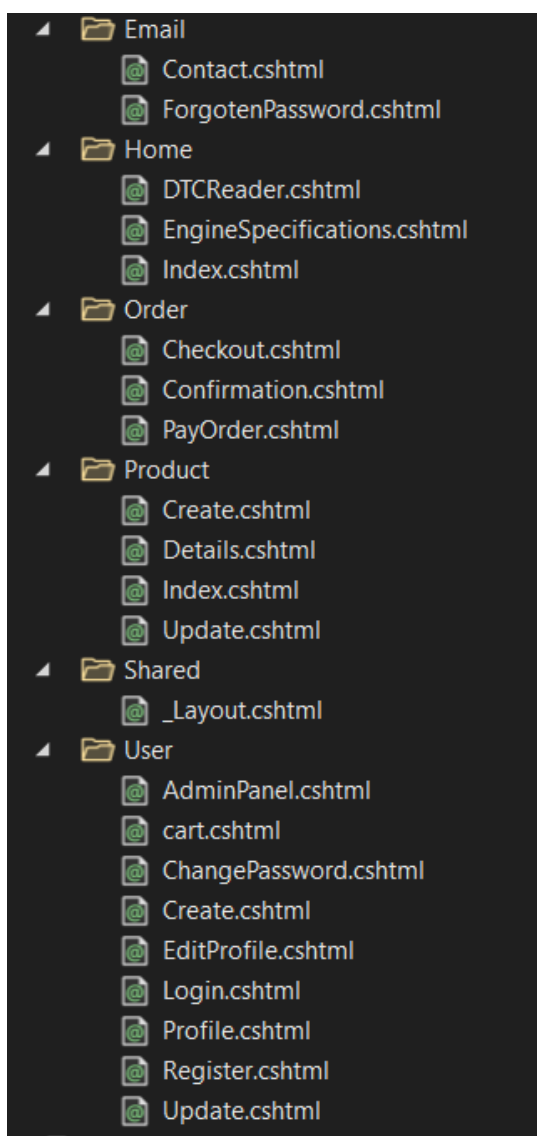


Фигура 8 Разпределение на папката ViewModel

От фигурата по-горе може да се види структурата на ViewModel-ите, или това представлява папка която съдържа множество подпапки, които служат като разделители за отделните контролери. Тези папки разделители съдържат всички модели, които биват използвани в дадения контролер.

## 2.2. Views

В тази папка се съдържат всички файлове с разширение .cshtml или по-точно всички страници с изграденият им чрез Html, CSS и JavaScript интерфейс. Това е една от папките в MVC архитектурата, като без тази папка уебсайтът или уеб приложението няма да има интерфейс (изглед на приложението).

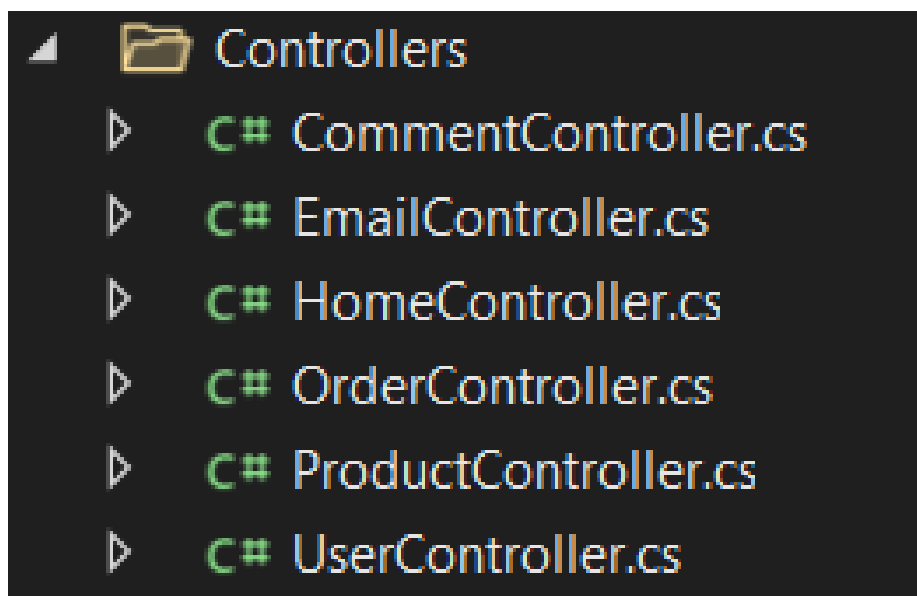


Фигура 9 Разпределение на файловете в папката Views

От фигурата по-горе се вижда структурата на този сегмент от архитектурата, като той представлява както при ViewModel-ите папка, съдържаща подпапки които отделят интерфейсите за отделните контролери. Чрез това разделение на подпапки е улеснено ориентирането в проекта. Без тази организация програмата не може да функционира коректно, защото контролерът търси нужния изглед точно посредством това папково разделение. Имената на тези папки трябва да съвпада с името на контролера и имената на самите изгледи трябва да съвпадат с имената на action-ите (действията) в съответния контролер. Ако тези имена не съвпадат страницата няма да бъде намерена и разработчика или потребителя ще получи грешка 404 (страницата не е намерена).

### 2.3. Controllers

Контролерът е последният член от архитектурата MVC. Този компонент извършва цялата логика на програмата, създава и запълва моделът и изпраща потребителят към определена страница. Контролерът се съдържа главно от методи тип IActionResult които след извършване на определени действия за манипулиране на данните връщат потребителя в определената страница (View). Всеки метод в този контролер трябва да съответства на дадена страница.



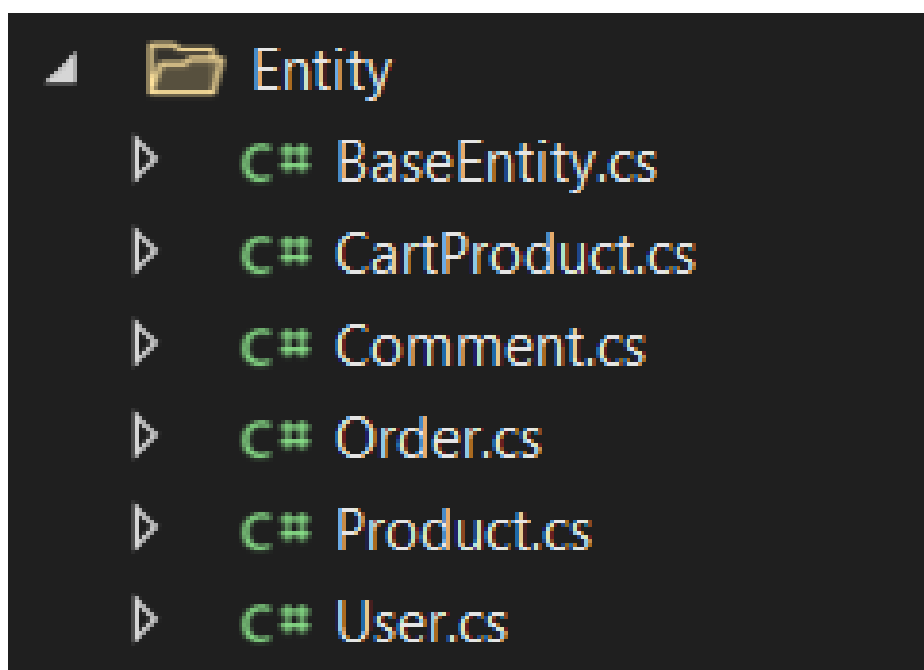
*Фигура 10 Разпределение на файловете в папката Controllers*

От фигура 10 може да се види какво представлява този сегмент в MVC архитектурата. В една програма контролерите биват разделени според това с какво правят. Те представляват файлове с разширение .cs които наследяват базовия клас

Controller за да могат да извършват работата на контролер. Контролерът е мозъка който извършва операциите в архитектурата.

## 2.4. Entity

В папката се съдържа описанието на всички обекти в уеб приложението. Тези обекти ще бъдат използвани от базата данни за това са отделени в отделна папка и се използват в CRUD (най-основната и използвана бизнес логика) операциите на приложението, които осъществяват действията по създаване, променяне, извличане и изтриване от базата данни. Без тези Entity-та програмата няма да може да записва данните в базата данни.



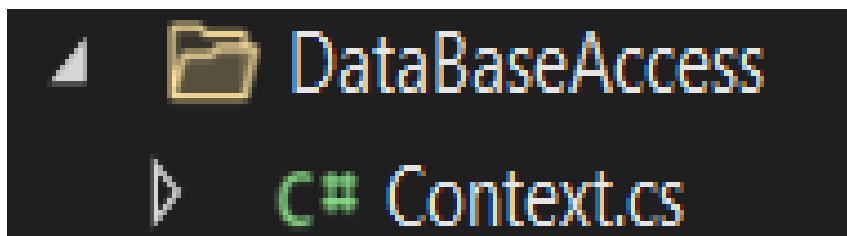
*Фигура 11 Разпределение на обектите в папката Entity*

От фигурата може да се види структурата на папката. Тя съдържа както ключовите за приложението обекти (коментари, поръчки, продукти, потребители и продукти по количките на потребителите), така и базовият обект (BaseEntity), който базов обект няма да има отделна таблица в базата данни а е се използва за да бъде наследяван от всички останали обекти от папката. Този клас съдържа само параметъра ID, който параметър е наследен от всички Entity-та за да могат да са индексирани в таблиците на базата.



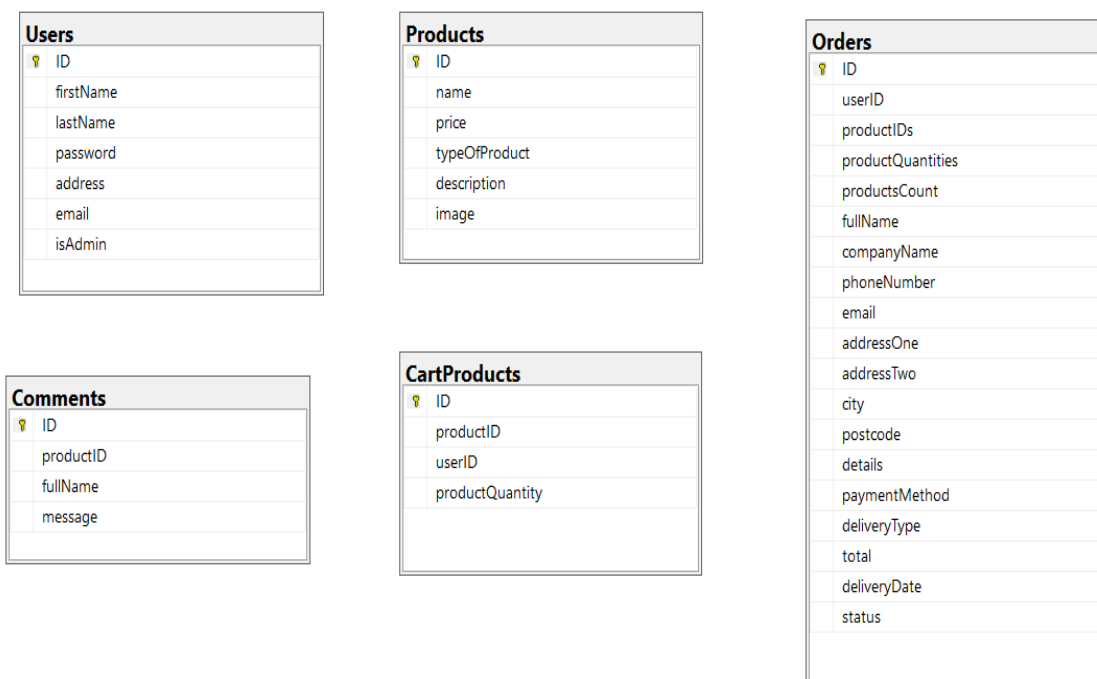
## 2.5. DataBaseAccess

Папката DataBaseAccess съдържа файлът, който инициализира самата база данни. Структурата на папката може да се види на фигура 12, а програмната реализация в Приложение 2.



Фигура 12 Разпределение на файловете в папката DataBaseAccess

Използвайки тази папка се инициализира не само самата база, а и всички таблици в нея, като на фигура 13 може да бъде видяна ER диаграмата на тази база от данни.

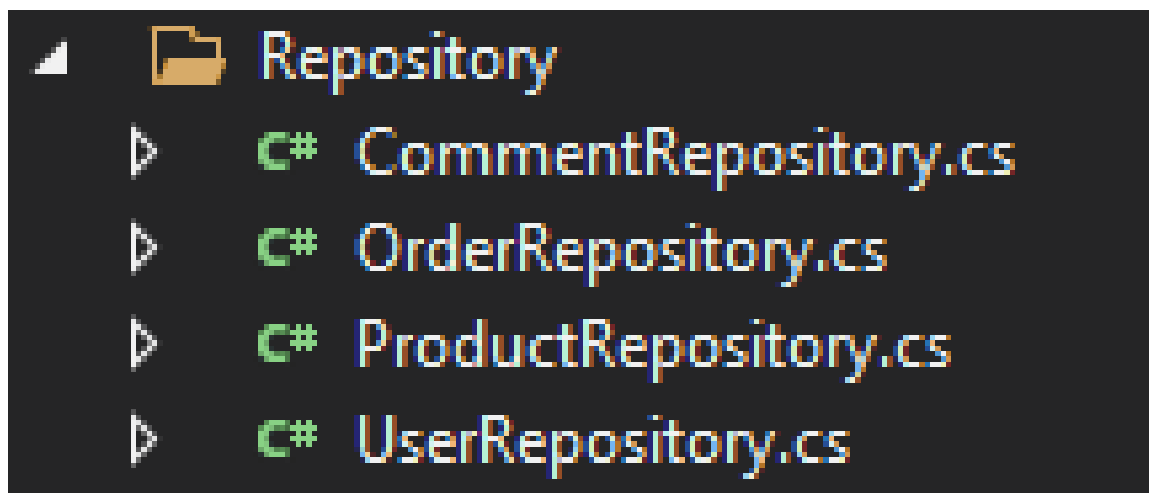


Фигура 13 ER диаграма на базата данни

От фигура 13 може да се проследят характеристиките на обектите, съдържащи се в папката Entity.

## 2.6. Repository

Папка Repository съдържа всички методи и функции, които манипулират данните в приложението. Това е единствената папка, която извиква и използва Context класът от папката DataBaseAccess. Папка Repository бива извиквана от всички други папки, които извършват логика, като Controller, Tools и т.н. От фигура 14 по-долу може да се види структурата на папката.

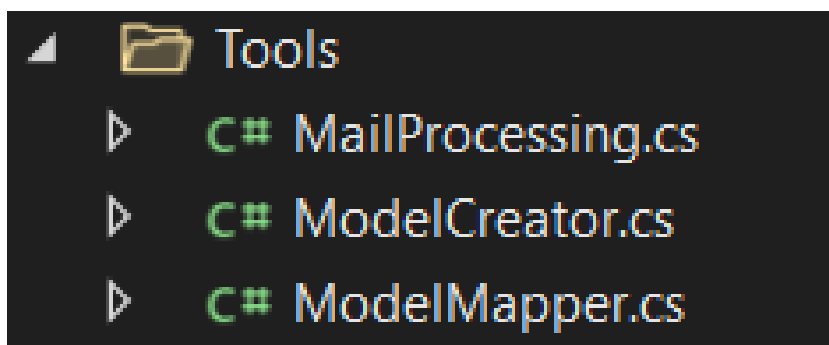


Фигура 14 Структура на папката Repository

Структурата представлява папка с 4 отделни класа за всеки обект от папката Entity. Отделният клас извършва манипулация само на един обект от Entity-тата. Това е много важна папка, която е мозъка на програмата и поддържа структурата на цялата програма.

## 2.7. Tools

Папка Tools съдържа класове инструменти, които подпомагат на програмата с различни видове действия.



Фигура 15 Структура на папка Tools

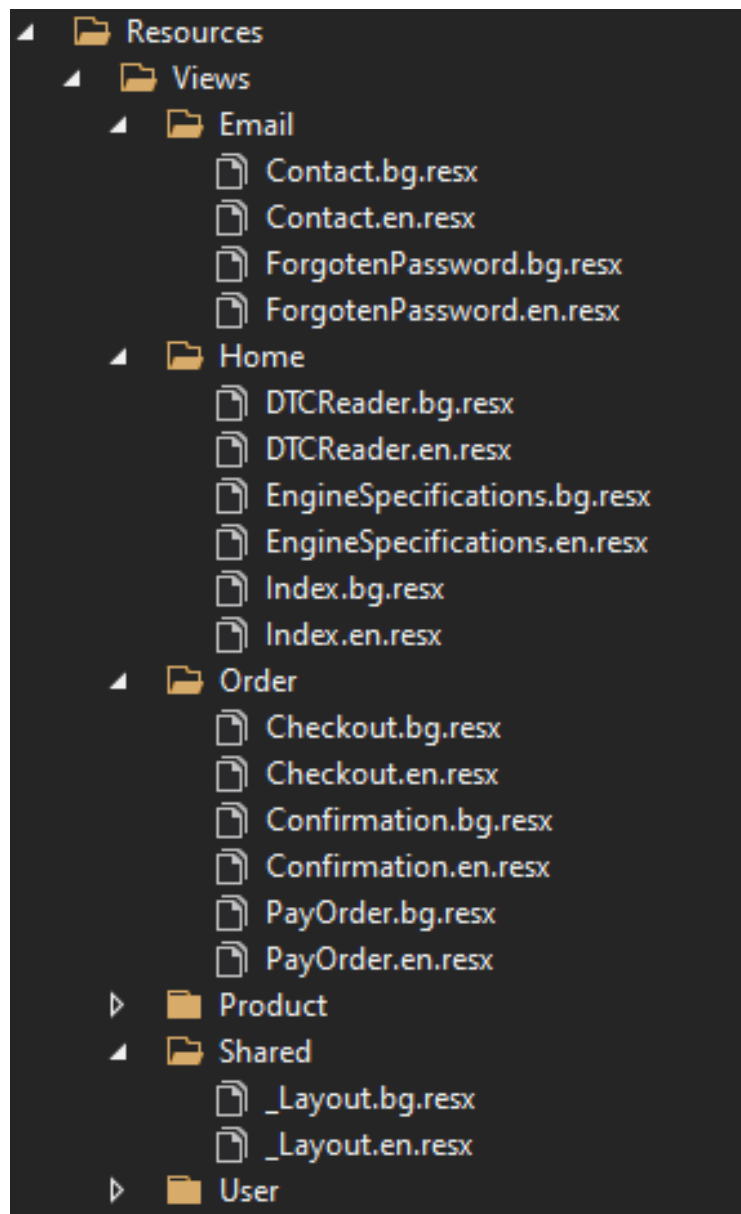
От фигура 15 се вижда структурата на папката. Тази структура се състои от три класа:

- MailProcessing.cs, който съдържа методи, чрез, които програмата изпраща имейли на потребителите. В конструктора на този клас се създава SMTP (Simple Mail Transfer Protocol) клиент, който приема хоста на електронната поща outlook, изходния порт 587, потребителския профил, от който ще се изпраща пощата и други настройки нужни за правилното функциониране на този клиент.
- ModelCreator.cs, който се занимава със създаването на моделите, които се изпращат в интерфейсите. Целта на класа е да се отдели създаването на модели в самите контролери, като вместо да се създава моделът се извиква даден метод от този клас, подават се нужните данни и метода връща готов модел с всички данни в него, който може директно да се използва от контролера.
- ModelMapper.cs, занимаващ се с превръщането на получените модели в обекти от папка Entity. Този клас, както и ModelCreator.cs, цели изчистването на голямото количество код от контролера. Класът съдържа методи, които приемат даден модел и го превръщат в готов, пълен с информация обект от тип Entity, който е готов за използване от контролерът.

## 2.8. Resources

Папка Resources съдържа текста на всички страници написани на български и английски.

От фигура 16 може да се види структурата на папката.



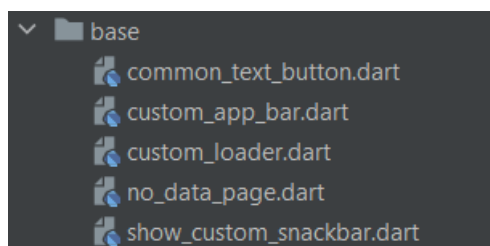
*Фигура 16 Структура на папка Resources*

Структурата на папката съответства на папката View, за да може текста да бъде намерен в точния файл с точния текст за даденото поле. Файловете в тази структура са от тип .resx и на всеки изглед от View съответстват 2 ресурсни файла от тази папка (един на български и един на английски).

## ГЛАВА 3 – СТРУКТУРА НА МОБИЛНОТО ПРИЛОЖЕНИЕ

### 3.1. Base

В папката base се съдържат само най-базовите widget-и (Джаджи). На фигура 17 се вижда структурата на папката.

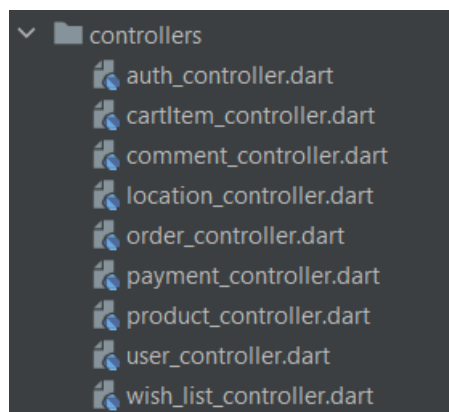


Фигура 17 Структура на папка base

В папката има 5 базови джаджи използвани навсякъде в процеса на изработка на външният вид на мобилната апликация, като тези джаджи представляват готов елемент от външният вид на приложението и само биват извикани на определеното място.

### 3.2. Controllers

Папката controllers извършва цялата логическа дейност на приложението. Тази папка съдържа контролери, които обработват цялата информация на приложението. На фигура 18 се вижда структурата на дадената папка.



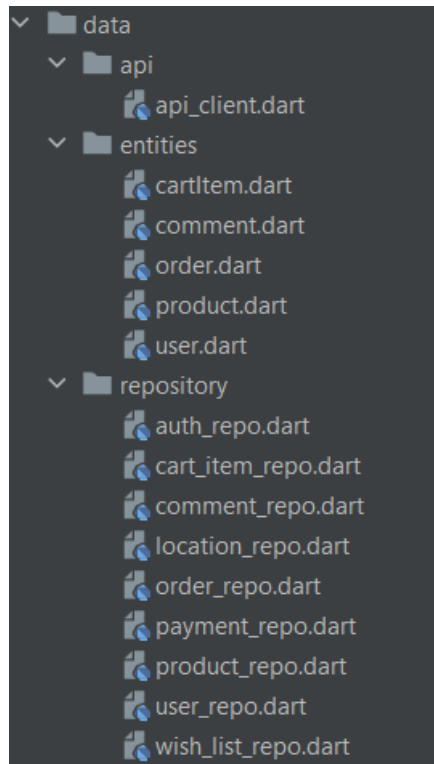
Фигура 18 Структура на папката Controller

В папката всеки контролер се използва за точно определено действие, като по този начин се намаля обема на контролерите, но за сметка на това тяхната бройка

се увеличава. От папката се вижда как всеки контролер си има определена работа: един за аутентикацията на приложението, друг за продукти, поръчки и други.

### 3.3. Data

Папката Data съдържа три под папки, като цялата папка съдържа най-важните данни на приложението. От фигура 19 се вижда и структурата на папката и нейните под папки.



Фигура 19 Структура на папката Data

В папката api има един клас, който представлява нашият api клиент, чрез който изпращаме всички заявки за информация, към дадената услуга.

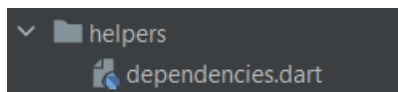
В папка entity се намират всички обекти, запазвани и обработвани от базата данни.

В папката repository за хранилищата на всички методи и функции използвани от контролерите.

### 3.4. Helpers

Папката helpers съдържа само един клас, който е от изключителна важност понеже той инициализира нашите контролери, хранилища, използваният api client

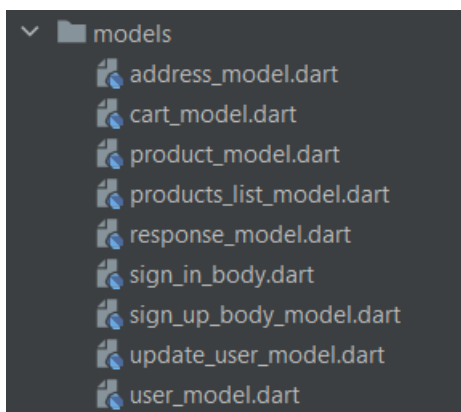
и класът `SharedPreference`, който ни помага да запазваме информация в хранилището на самото устройство.



Фигура 20 Структура на папката *Helpers*

### 3.5. Models

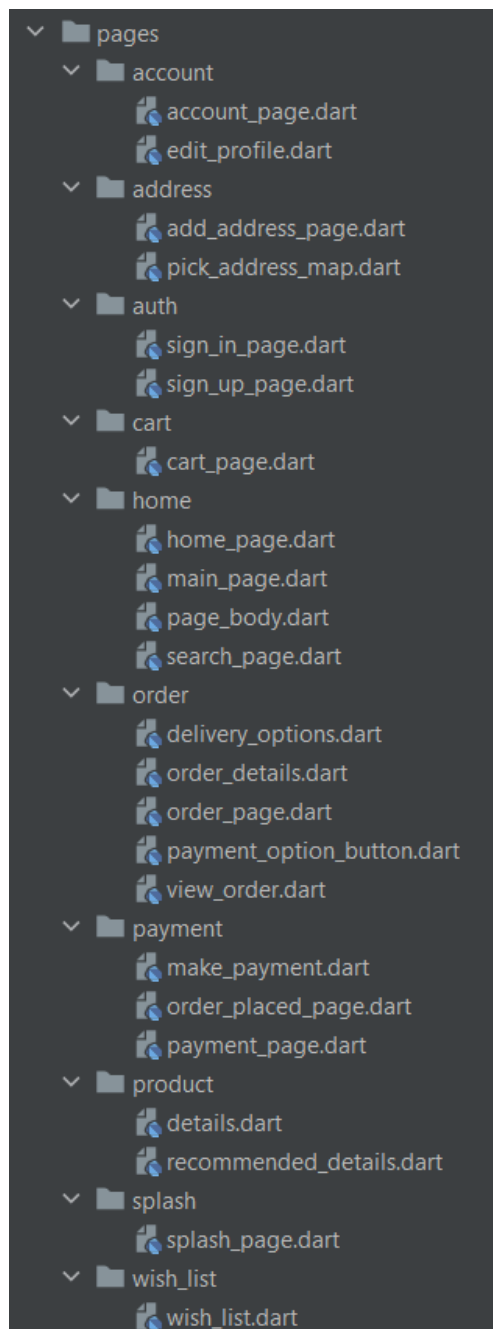
Папката `models` съдържа всички модели използвани за изобразяване, прехвърляне и обработване на информацията от контролерите и хранилищата. От фигура 21 се вижда и структурата на папката.



Фигура 21 Структура на папката *Models*

### 3.6. Pages

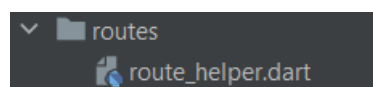
Папката `pages` съдържа множество под папки. Тази папка съдържа всичките напълно структурирани страници на приложението. В тези страници биват извиквани `widget`-ите в папка `base` и `widgets`, които формират самата страница на приложението. От фигура 22 може да се види файловата структура на папката.



Фигура 22 Структура на папката Pages

### 3.7. Routes

Папката routes съдържа клас с всички пътища към страниците от папка pages. Този клас бива извикван от GetX (State Management tool). От фигура 23 се вижда и файловата структура на папката.

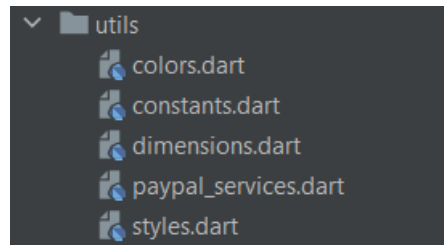


Фигура 23 Структура на папката Routes



### 3.8. Utils

Папката `utils` представлява инструменти създадени от разработчика или от дадена услуга използвани за улесняване на работата на приложението. От фигура 24 се вижда структурата на папката с нейните класове.



Фигура 24 Структура на папката *Utils*

Класа `colors` съдържа цветовете използвани от разработчика за да даде вид на страниците на мобилната апликация.

Класа `constants` съдържа главни низове, които не подлежат на промяна и биват извиквани на определени места в приложението.

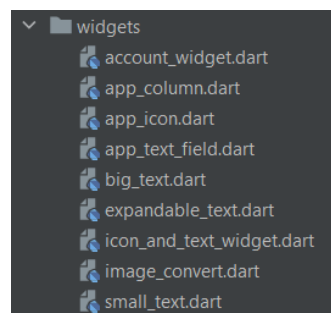
Класа `dimensions` съдържа множество пропорции на различни величини според големината на екрана на използваното устройство.

Класа `paypal_services` съдържа готов код от страницата на PayPal, който се използва при инициализиране на плащане.

Класа `styles` съдържа различни стилове размери и дебелени на шрифтове.

### 3.9. Widgets

Папката `widgets`, както `base` съдържа джаджи, но джаджите от тази папка не са толкова базови а са по – скоро изградени само за самото приложение. От фигура 25 се вижда структурата на папката.



Фигура 25 Структура на папката *Widget*

## ГЛАВА 4 ПРОГРАМНА РЕАЛИЗАЦИЯ НА ВСЕКИ ОТ МОДУЛИТЕ

### 4.1. База данни

За реализация на базата данни е използван SSMS (SQL Server Management System) и Nu-Get пакета EntityFramework. Реализацията е code first, тоест става през кода. Първо се изтегля пакетът EntityFramework и се създават Entity-тата (обектите за базата данни). След тези стъпки се създава класът Context който изглежда по следния начин :

```
public class Context : DbContext
{
    public DbSet<User> Users { get; set; }
    public DbSet<Product> Products { get; set; }
    public DbSet<Comment> Comments { get; set; }
    public DbSet<CartProduct> CartProducts { get; set; }
}

public DbSet<Order> Orders { get; set; }

public Context() : base("Data
Source=SQL8002.site4now.net;Initial
Catalog=db_a91afa_autopoint;User
Id=db_a91afa_autopoint_admin;Password=Sami5550125")
{
    Users = this.Set<User>();
    Products = this.Set<Product>();
    Comments = this.Set<Comment>();
    CartProducts = this.Set<CartProduct>();
    Orders = this.Set<Order>();
}
}
```

За да работи класът трябва да наследява DbContext. Класът съдържа няколко DbSet-а, които репрезентират таблиците от базата и конструктор който наследява

от супер класът и в base на конструктора се слага Connection string-a. Когато Context класът е готов, трябва да се отвори Package manager console, който се намира на View/Other Windows/Package Manager Console и да се въведат следните команди в тази конзола:

- enable-migrations която команда създава папката с миграциите и конфигурационния файл
- add-migration (име на миграцията) което създава миграция
- update-database която команда обновява базата с последната налична миграция

След подаването на тези команди, базата данни бива създадена и в мениджмънт системата с всичките таблици, първични ключове и други.

## **4.2. Интерфейс**

За създаването на интерфейса на цялото приложение и използван Bootstrap template. Откриването на такъв шаблон става много лесно. В интернет пространството са налични множество шаблони, като шаблона имплементиран в приложението, е взет от сайта ThemeWagon. Изтегления шаблон съдържа Html, CSS, JavaScript и други ресурсни файлове. За имплементацията се преместват нужните файлове в wwwroot папката на проекта. След това, ако шаблона няма да подлежи на корекции Html файловете директно се изпращат в папката View. В настоящото приложение, при създаването на интерфейса, всички Html страници биват или променяни или изградени от нулата. Промените се отнасят до смяна на цветовете, шрифтове, размери, местоположения на фрагменти и други подобни действия.

css	Папка с файлове					20.7.2020 г. 23:47
fonts	Папка с файлове					20.7.2020 г. 23:47
img	Папка с файлове					20.7.2020 г. 23:47
js	Папка с файлове					20.7.2020 г. 23:47
Karma Shop-doc	Папка с файлове					20.7.2020 г. 23:47
scss	Папка с файлове					20.7.2020 г. 23:47
.DS_Store	DS_STORE файл	4 КБ	He	15 КБ	78%	20.7.2020 г. 23:47
blog	Opera GX Web Document	5 КБ	He	35 КБ	87%	20.7.2020 г. 23:47
cart	Opera GX Web Document	4 КБ	He	21 КБ	83%	20.7.2020 г. 23:47
category	Opera GX Web Document	6 КБ	He	36 КБ	86%	20.7.2020 г. 23:47
checkout	Opera GX Web Document	5 КБ	He	20 КБ	80%	20.7.2020 г. 23:47
confirmation	Opera GX Web Document	4 КБ	He	12 КБ	74%	20.7.2020 г. 23:47
contact	Opera GX Web Document	4 КБ	He	13 КБ	73%	20.7.2020 г. 23:47
contact_process	PHP Source File	1 КБ	He	2 КБ	58%	20.7.2020 г. 23:47
elements	Opera GX Web Document	7 КБ	He	36 КБ	82%	20.7.2020 г. 23:47
index	Opera GX Web Document	5 КБ	He	40 КБ	89%	20.7.2020 г. 23:47
login	Opera GX Web Document	4 КБ	He	11 КБ	72%	20.7.2020 г. 23:47
prepros-6.config	CONFIG файл	2 КБ	He	7 КБ	71%	20.7.2020 г. 23:47
single-blog	Opera GX Web Document	6 КБ	He	35 КБ	86%	20.7.2020 г. 23:47

Фигура 26 Файлове съдържащи се в Bootstrap шаблона

От фигура 17 се вижда примерен Bootstrap шаблон. За имплементацията му папките css, fonts, img и js биват преместени в споменатата wwwroot папка. Файлове като blog, cart checkout и други са Html-a, или скелетът за изграждането на интерфейса. Чрез съдържанието на тези файлове се изграждат всички страници от приложението.

### 4.3. Интегриране на API

За създаването на страниците DTC reader, Engines, както и за заплащането, е нужно интегрирането на специфични API. Преди интеграция трябва да бъде открит подходящ за работата API. Има много сайтове, предоставящи достъп до своя услуга, но интегрираните в приложението са взети от rapidAPI. След намирането на тези API-та, те трябва да бъдат използвани в кода, като в примера:

```
var client = new HttpClient();
var request = new HttpRequestMessage
{
    Method = HttpMethod.Get,
```

```

        RequestUri = new Uri("https://car-
code.p.rapidapi.com/obd2/" + errorCode),
        Headers =
        {
            { "X-RapidAPI-Key",
"534846abf0mshb184e5d747fecebp11d6aajsn7d9451e74bc2" },
            { "X-RapidAPI-Host", "car-
code.p.rapidapi.com" },
        },
    };
    using (var response = client.Send(request))
    {
        //here we get the response and convert
it into a view model
        response.EnsureSuccessStatusCode();
        var body =
response.Content.ReadAsStringAsync();

        DTCReaderVM model =
JsonConvert.DeserializeObject<DTCReaderVM>(body.Result);

        //we get returned to the view
        return View(model);
    }

```

За интеграцията първо се създава HttpClient, който ще изпраща заявката до самият API. След него се създава самата заявка, като при изграждането на HttpClient-а задаваме какъв е HTTP методът, линкът за API-а и хедърите, предоставени от rapidAPI. Като заявката е успешно създадена, следва изпращането ѝ. В резултат се получава отговор от API-а в Json формат. Този отговор се десериализира в модел чрез Nu-Get пакета Newtonsoft.Json и впоследствие този модел се подава на страницата.

#### 4.4. Разплащане

Платежният метод, интегриран в програмата е PayPal. Той поддържа както плащане по тяхната система, така и платежни методи като ApplePay, GooglePay, PayLater и плащане чрез дебитна карта. За да бъде интегриран този платежен метод, трябва първо да се създаде developer акаунт в PayPal. След успешното му създаване потребителя получава ключ за използване на външният API на PayPal. От имплементацията на Приложение 3 може да се види и автогенерирания код, който разработчика просто трябва да постави в Html кода на страницата.

В първия script таг трябва да се постави получения от PayPal ключ. Втория script инициализира вградените бутони за разплащане на определеното място от потребителя. За задаване на това място трябва да бъде поставен този код който PayPal предоставят :

```
<div class="payment_item">
                                <div id="smart-button-
container">
                                <div style="text-align:
center;">
                                <div id="paypal-
button-container"></div>
                                </div>
                                </div>
                                </div>
```

Освен с инициализацията на бутоните, скрипта разполага и с методите onApprove и onError. При натискане на платежните бутони потребителя бива изпратен чрез външен API към платежната система на PayPal, където записва неговата информация. След успешно направено заплащане, потребителя бива върнат в сайта и формата за разплащане бива изпратена с платежния метод.

#### 4.5. Описание на процеса на разработка на уеб апликацията

Работата по разработката започва с изграждането на интерфейса. За постигането е изтеглен bootstrap шаблон чиито CSS, JavaScript, Fonts и снимки биват преместени в wwwroot папката на приложението. Без тези файлове интерфейсът на приложението няма да работи. След поставянето на нужните файлове Html, кодовете на шаблона биват изменени и пригодени за спецификите на уеб приложението. Като целият интерфейс е готов, следва създаването на

файловата архитектура на приложението. За тази изработка първо се създават папките ViewModel, Views, и Controllers, ако вече не са създадени в приложението при генериране на проект посредством стартов шаблон. Тези три папки отговарят за MVC архитектурата на приложението. След създаването им е нужно да се добавят папките Entity и DataBaseAccess, които се занимават с структурата на обектите в приложението и това как и къде ще бъдат запазени. Последните две папки, които трябва да се създадат са Repository и Tools, като от тях зависи функционалността на приложението. До момента приложението съдържа готов интерфейс и файлова архитектура. Следва имплементирането на базата данни. За тази имплементация първо се създават обектите които ще бъдат съхранявани (потребители, продукти, коментари и други.) в папката Entity. След създаването им се разработва клас Context в папката DataBaseAccess. Този Context ще извършва запазването, променянето, извличането и изтриването на обекти от базата данни. Класът съдържа DbSet полета за всички обекти от папка Entity, като тези сетове кореспондират на всички таблици в приложението. Изработва се конструктор, който съдържа Connection string-a (низ който определя сървъра и самата база с която да се върже програмата) и вътре в този конструктор се инициализират сетовете. След създаването му в Packet Manager Console се подават следните три реда код:

- `enable-migrations` – този код създава папката migrations и конфигурационният ѝ файл
- `add-migration name` – този код създава миграция
- `update-database` – този код ъпдейтва базата данни спрямо последно добавената миграция.

След създаването и инициализирането на базата данни следва да се създаде цялата логика на приложението или по точно изграждането на класовете в папките Repository и Tools. След създаване в контролерите на тези класове и функции остава да се осъществи техническа работа по превеждане на сайта. За целта се създава папка Resources, която съдържа ресурсни файлове като за 1 страница кореспондират 2 ресурсни файла (един на Български и един на Английски). След създаването и попълването на ресурсите променяме културата на приложението от `program.cs`, инжектираме `localizer` във всички страници и заменяме информацията с

името на инстанцията от тези ресурсни файлове. След като всичко от горе изброените е готово, следва проверка дали сайта функционира коректно и хостването му.

#### **4.6. Описание на процеса на разработка на услугата (API)**

ASP.NET Core предлага възможността и за създаване на уеб услуга. За създаването на тази услуга първо трябва да си създадем класът `data` който съдържа нашите обекти от базата данни и инициализирането на връзката към дадената база. Класовете могат да бъдат копирани от уеб апликацията а инициализацията е авто генерирана от самата среда.

След създаването на тази папка се създава и папката `controllers`, където в зависимост таблиците от базата са създадени и определените контролери. При създаване на контролер той трябва да наследява `ControllerBase` и над дефинирането на класа да се сложат анотации с път към дадения контролер и анотация `ApiController`, дефинираща че даденият клас е за услуга (API).

Вътре в класа се дефинират всички заявки които ще се създадат в бъдеще или се използват понастоящем. При дефиниране на заявката, която представлява метод, над нея се слага анотация `HttpGet`, `HttpPost`, `HttpPut`, `HttpDelete` и други, в зависимост какво действие извършва дадения метод. Освен това и в самата анотация се дефинира пътя към заявката и евентуалните ѝ параметри.

В тялото на метода извършваме дадена операция, като накрая винаги трябва да връща резултат, който бива код за статус, обект, двете обединени в едно и други.

#### **4.7. Описание на процеса на разработка мобилното приложение**

За изработка на мобилното приложение е използван Flutter, заради това че написан за една платформа, кода може да бъде компилиран като iOS или Android апликация, уеб приложение или като Desktop апликация. Освен това е избрана тази среда, защото е най-бърза спрямо всички останали среди предлагащи код компилиран като iOS и Android приложение едновременно.

Езикът Dart използван в тази рамка е базиран на C# и java. Процеса на изработка започва със създаването на файловата структура на приложението, както



е показана в глава 3. След създаването на всички папки избираме кой state management tool ще използваме.

В AutoPoint Mobile този инструмент е GetX. Когато инструмента е избран, неговото dependency бива извикано във файла pubspec.yaml. Впоследствие се извиква командата pub get, за да може GetX да бъде добавен в проекта. Веднъж добавен този state management tool бива използван навсякъде в приложението, като предназначението му е да променя състоянието на страниците, да инициализира контролери, хранилища и други важни обекти.

След имплементирането на инструмента трябва да бъдат направени два много важни класа – dependencies и route\_helper. Dependencies съдържа само init метод който слага всички контролери, хранилища и използваният API клиент в GetX, като всички тези структури след добавянето им могат вече да имат само една инстанция, направена през Get. Класът RouteHelper съдържа пътищата към всички страници на мобилното приложение.

Като създадем двата важни за GetX класа, следва създаването на обектите от подпапка entity. За тяхната изработка се създават същите полета като тези в AutoPoint уеб приложението, но освен това на тези обекти в мобилното приложение биват изградени toJson и fromJson методи които конвертират обекта на json резултат и обратно.

Като обектите са готови се насочваме да създадем API клиента, където се съдържат всички извиквания на API заявки или към вътрешният API или към външни сървъри. Този клас допълва (extends) GetConnect и имплементира GetxService. Съдържа две полета – token и baseUrl, където baseUrl е линкът към нашият вътрешен API, а tokenът е низ получен след извършването на заявки който удостоверява нашата аутентикация. След като тези две полета са дефинирани вътре в класът се дефинират всички заявки, като този клас ще бъде дефиниран във всяко хранилище.

След изграждането на клиента се насочваме към създаване на нужните хранилища. За това след създаването на класа се дефинира създаденият API клиент и в зависимост от това, дали хранилището ще записва данни в хранилището на даденото устройство, се дефинира и класът SharedPreferences който улеснява това

запазване на устройството с 3 главни метода (set, get, delete). След като сме дефинирали нужните за хранилището класове вътре извършваме цялата логика която ще бъде извиквана в кореспондиращият на хранилището контролер.

След създаването на хранилищата създаваме кореспондиращите контролери. Тези контролери имплементират `GetxService` и допълват (extends) `GetController`. Вътре в контролера дефинираме хранилището и всички останали променливи и методи използвани по страниците.

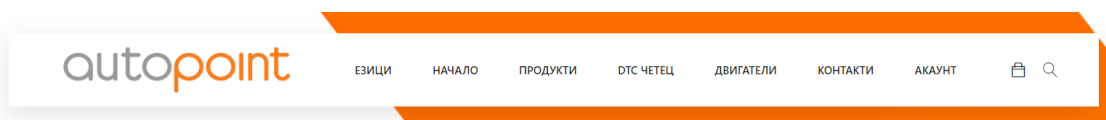
Като контролерите са готови може да се създадът класовете в папката `utils`. Тези класове служат като инструмент за различни аспекти на приложението. `Colors` съдържа само дефинирани цветове, `constants` е константния клас където се съдържат плаващите низове, `dimensions` съдържа размери създадени с определени пропорции така размер 20 например да бъде един и същ на мобилни устройства с различна големина на екрана. Освен това има класът `styles`, който съдържа дефинирани шрифтове размери и всякакви променливи използвани при текстообработката и последният инструмент `paupal_services`, който съдържа два метода които ни помагат да инициализираме разплащане в приложението. Събрани заедно тези инструменти помагат при изработката на страниците във всички аспекти на обработка.

След като сме създали всички тези класове започва създаването на страниците с имплементирането на логиката навсякъде по тези страници. В процеса на изработката също така биват изработени и класовете в папките `base` и `widgets`, които са джаджи отделени за да бъдат преизползвани.

## ГЛАВА 5 РЪКОВОДСТВО ЗА ПОТРЕБИТЕЛЯ НА УЕБ ПРИЛОЖЕНИЕТО

### 5.1. Навигация в приложението

Навигационното меню на уеб приложението е изложено на фигура 18, като сайта може да бъде достъпен на <http://siteloginauto-001-site1.atempurl.com>.

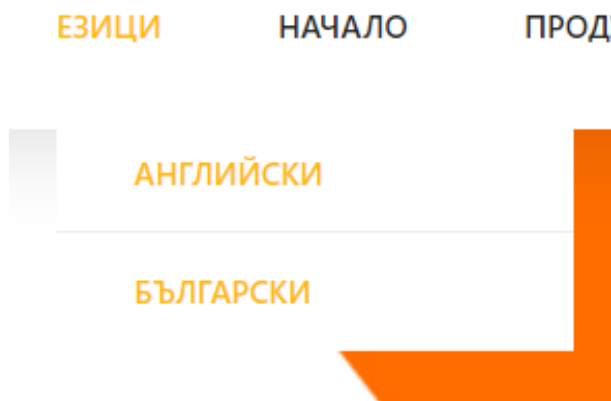


Фигура 27 Навигационно меню на уеб приложението

До логото на сайта са всички страници, които могат да бъдат достъпени, както и търсачката с икона на лупа и количката с икона на чанта.

### 5.2. Бутон Езици

При преместване на мишката до бутона езици се появява падащото меню показано на фигура 19.

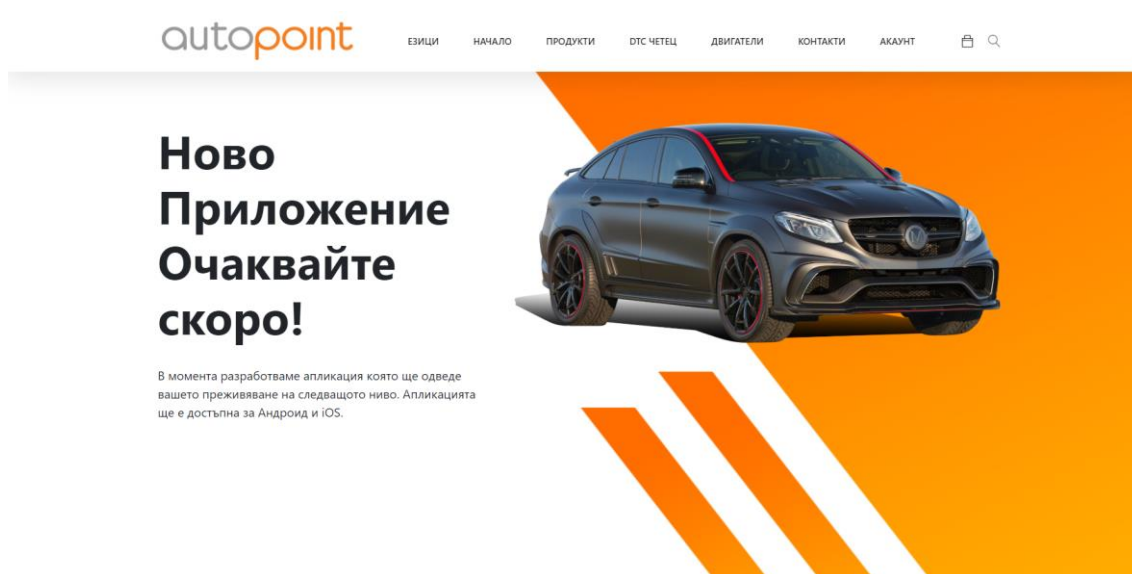


Фигура 28 Падащото меню на бутона езици

В падащото меню потребителя може да избере какъв да бъде езикът на страницата като са достъпни два: Български и Английски.

### 5.3. Бутон Начало

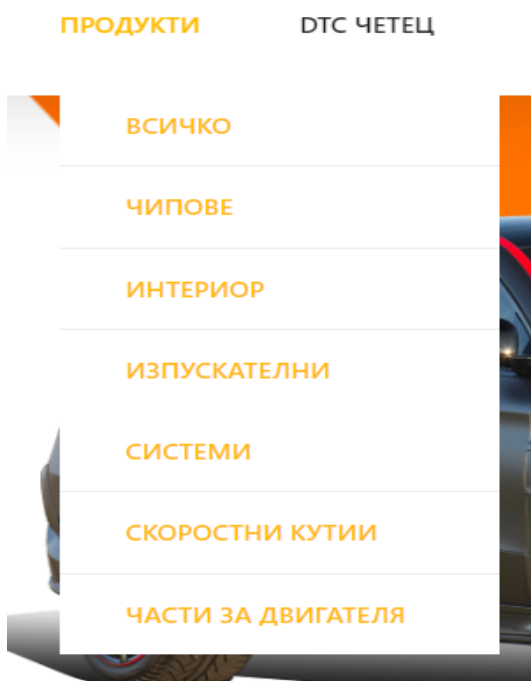
При натискане на бутона потребителя бива върнат към главната страница на приложението както е показано на фигура 20.



Фигура 29 Заглавна страница на уебсайта

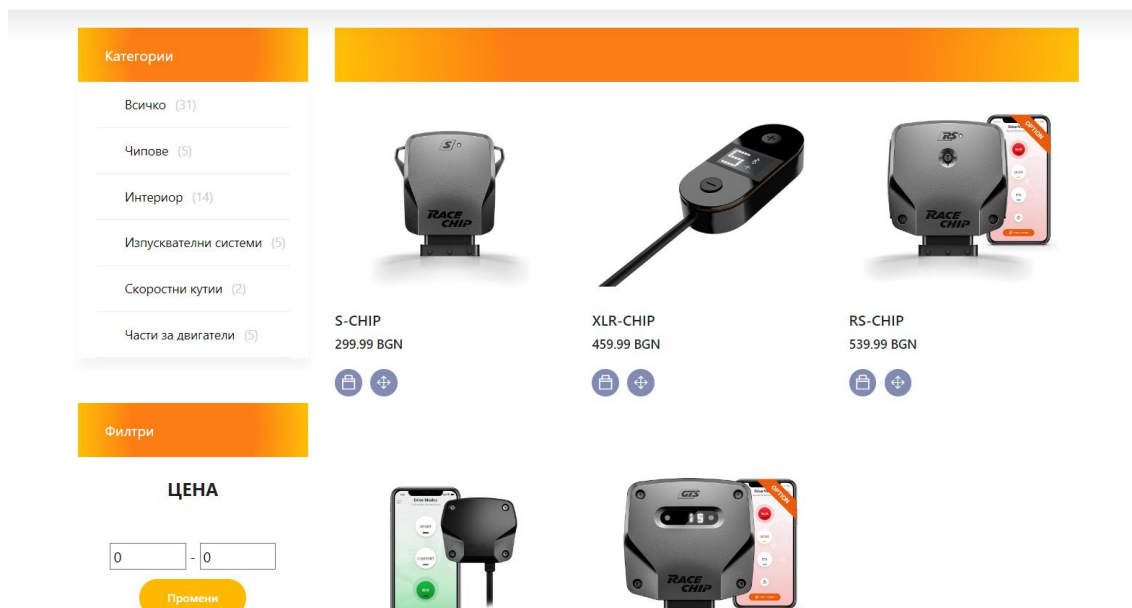
#### 5.4. Бутон Продукти

При плъзгане на мишката към Продукти, потребителя вижда падащото меню от фигура 21.



Фигура 30 Падащо меню на страница продукти

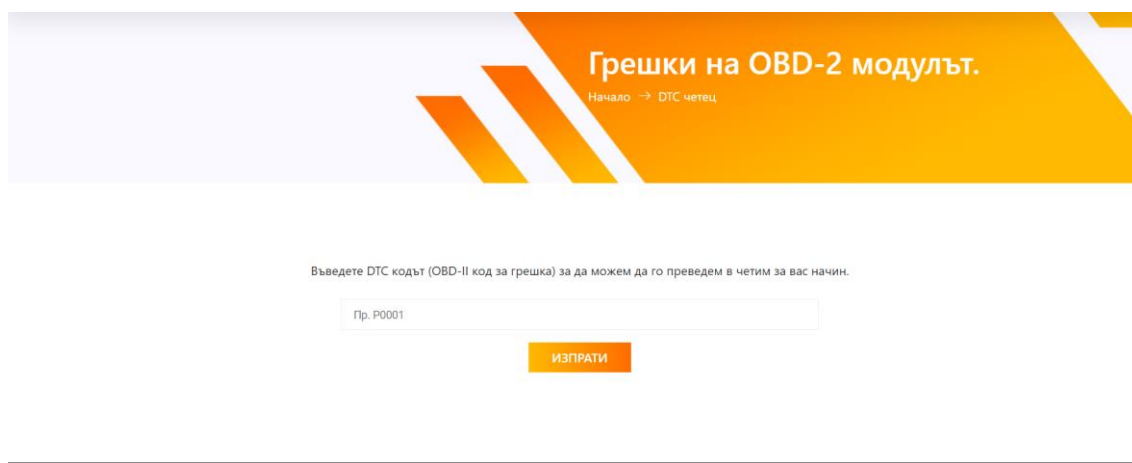
При натискане на някоя от опциите потребителят бива изпратен към страницата с продуктите, филтрирани според избраната категория, както се вижда на фигура 22.



Фигура 31 Католаго с продукти на сайта

## 5.5. Бутон DTC четец

При натискане на Бутона потребител бива отведен до страницата на фигура 23.



Фигура 32 Страница DTC четец

На тази страница потребител може да въведе стойността получена от притежаваният от него OBD модул и при успешно въведен код, потребителя получава резултата от търсенето на кода както е показан в фигура 24.

Въведете DTC кодът (OBD-II код за грешка) за да можем да го преведем в четим за вас начин.

P0305

ИЗПРАТИ

01. Код: P0305

02. Описание: Cylinder 5 Misfire Detected

03. Причини:

- a. Mechanical motor failure
- b. Wiring harness
- c. Ignition system/fuel supply circuit
- d. Injector
- e. Coolant temperature sensor/air flow meter
- f. Engine management calculator

*Фигура 33 Резултат получен при въвеждане на код от OBD модул*

## 5.6. Бутон Двигатели

При натискане на бутона потребителя бива изпратен в страницата както е показано на фигура 25.

**ВЗЕМЕТЕ СПЕЦИФИКАЦИИ ЗА КОНКРЕТЕН АВТОМОБИЛЕН ДВИГАТЕЛ**

Тази функция поддържа само двигатели направени 2020 година от изброените марки :

Mazda, Mercedes, Toyota, Ford, Volkswagen, Fiat, Kia, Hyundai, Audi, BMW, Honda, Volvo, Chevrolet.

марка

модел

ТЪРСЕНЕ

*Фигура 34 Страница Двигатели*

При въвеждане на марка и модел, потребител получава резултат като на фигура 26.

Искате ли да [опитате отново?](#)

01.	Двигател / Рама ID 8163
a.	Тип двигател gas
b.	Вид гориво regular unleaded
c.	Цилиндри: I4
d.	Конски сили(HP) 187
e.	Конски сили (RPM) 6000
f.	Въртящ момент(LBS) 186
g.	Въртящ момент(RPM) 4000
h.	Клапани 16
i.	Синхронизация на клапаните Variable
j.	Тип камера: Double overhead cam (DOHC)
k.	Тип задвижване front wheel drive
l.	Предаване 6-speed shiftable automatic
02.	Двигател / Рама ID 8164
a.	Тип двигател gas
b.	Вид гориво regular unleaded
c.	Цилиндри: I4
d.	Конски сили(HP) 187
e.	Конски сили (RPM) 6000
f.	Въртящ момент(LBS) 186

Фигура 35 Резултат от търсене на двигател

## 5.7. Бутон Контакти

При натискане на бутона потребителя е изпратен към контактната страница.

**Пловдив, България**  
 Ул. Битоля 24Б

**(+359) 889 5361 85**  
 Понеделник до петък от 9ч. до 18ч.

**AutoPoint2023@outlook.com**  
 Изпратете ни съобщение когато поискате!

**ИЗПРАТЕТЕ СЪОБЩЕНИЕТО**

Фигура 36 Страница Контакти

На фигура 27 се вижда страницата като от ляво на формата за изпращане на въпрос са телефона за контакт, местоположението на магазина и имейла за контакт. При натискане на изпратете съобщението потребителя изпраща реален имейл на администраторите на сайта с неговия въпрос.

## 5.8. Количка

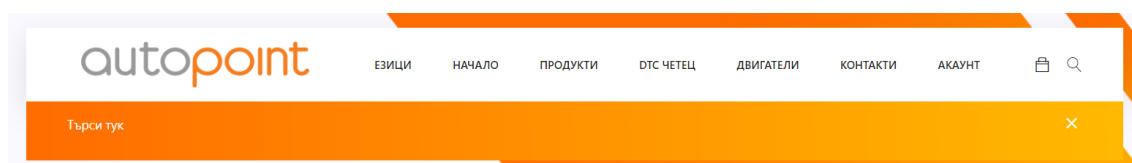
При натискане на иконата на торба потребителя отива в своята количка. На фигура 28 може да се види как изглежда самата количка.

Продукт	Бройка	Цена
Общо		0 BGN
Доставка	Безплатна доставка: 0 BGN	
<div>НАЗАДПРОДЪЛЖЕТЕ ДО ОГЛЕД</div>		

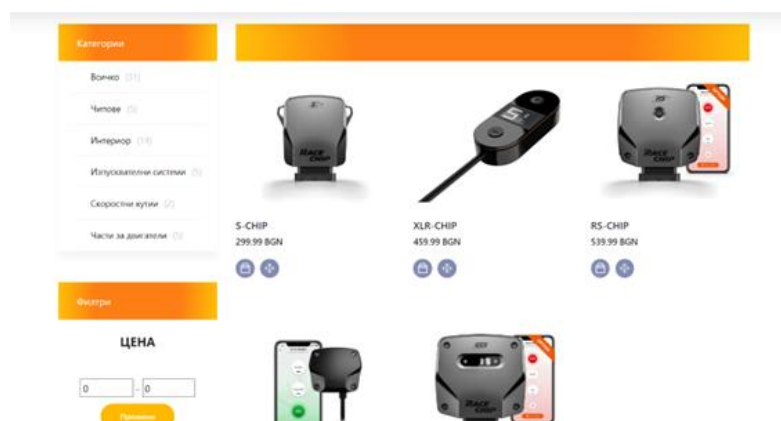
Фигура 37 Потребителската количка

## 5.9. Търсачка

При натискане на лупата се появява поле за въвеждане на търсен продукт, както се вижда на фигура 29. След потърсване на продукт потребителя бива изпратен в страницата с продуктите, които съдържат търсената дума в името си. Резултатът от търсенето е показан на фигура 30.



Фигура 38 Търсачката на приложението

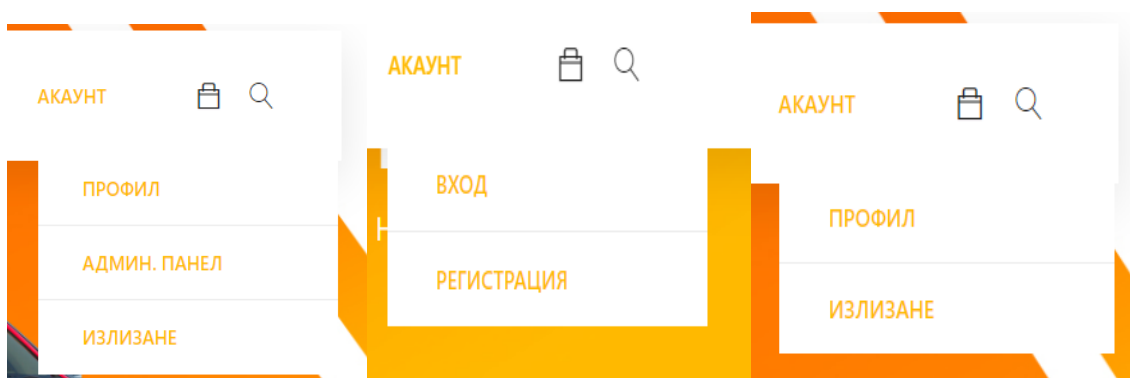


Фигура 39 Резултат от търсенето на продукти



## 5.10. Бутон Акаунт

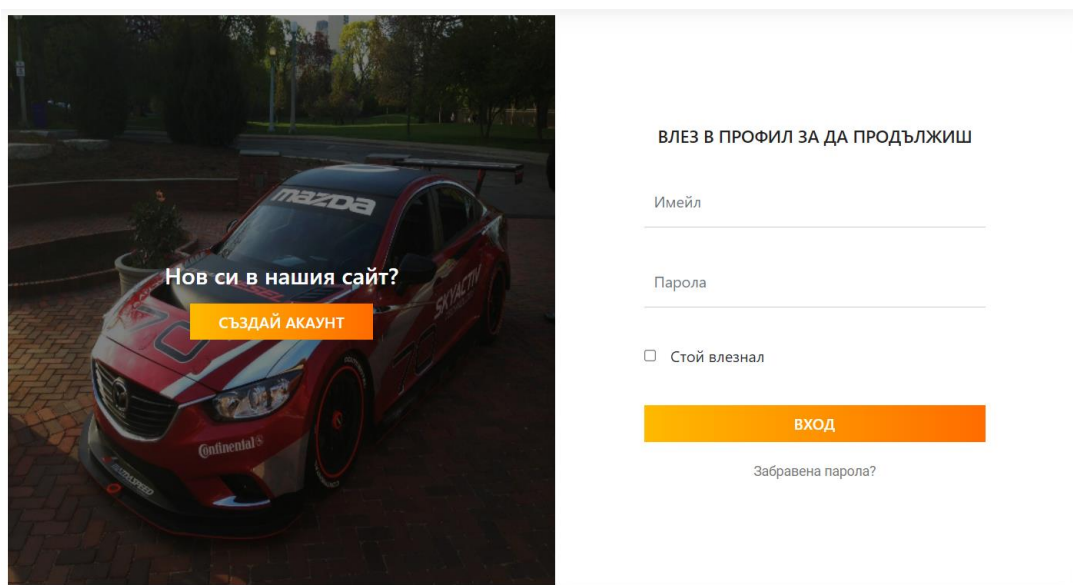
При плъзгане на мишката към бутона се показва падащото меню, което в зависимост дали потребителя си е влезнал в акаунта показва различни функции. Различните функции могат да се видят на фигура 31. На лявата снимка са функциите на администратор, на средната изглед без вписан потребител и в дясно на регистриран и включил се потребител.



Фигура 40 Бутон Акаунт

## 5.11. Вход

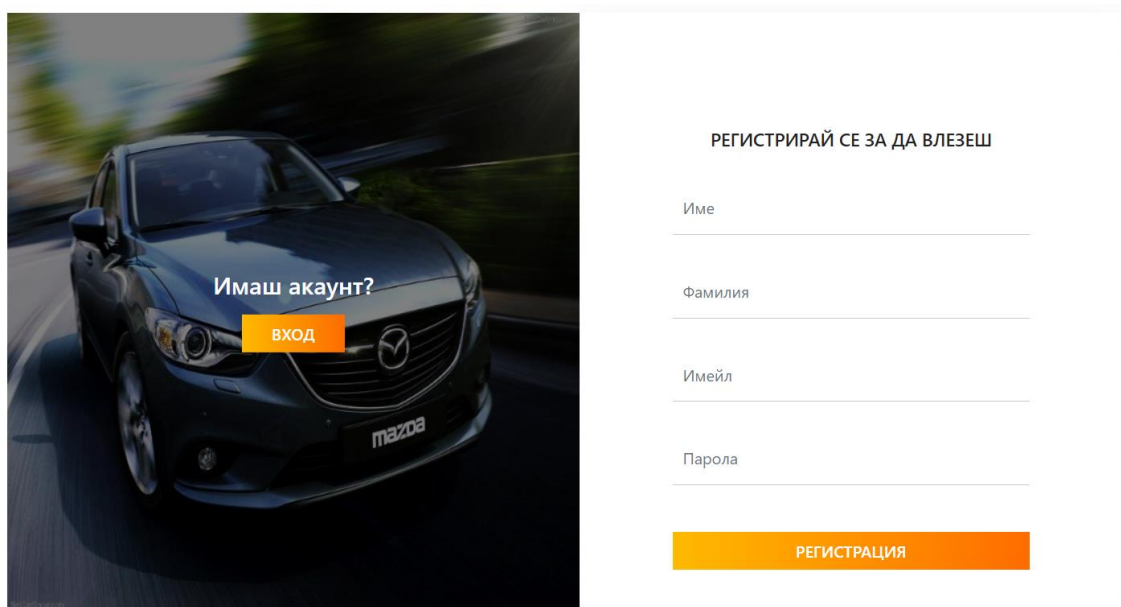
Това е страницата, в която потребителя се вписва в акаунта си. Тази страница има опция да запомни потребителя, опитващ се да влезе, да го изпрати в страницата за забравена парола или в страницата за регистрация. На фигура 32 се вижда страницата.



Фигура 41 Страница за Вход

## 5.12. Регистрация

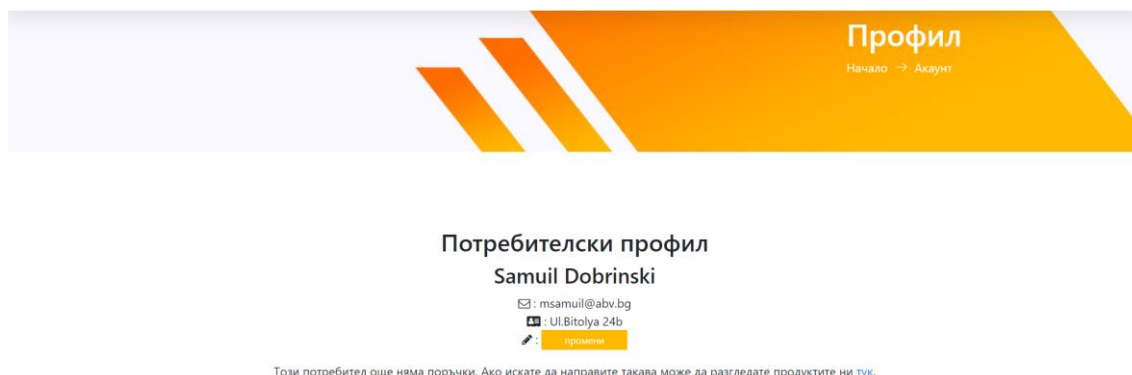
Страницата за регистрация съдържа форма за записване и бутон който препраща потребителя към страницата за вход ако има акаунт. Страницата се вижда на фигура 33.



Фигура 42 Страница за регистрация

## 5.13. Профил

При натискане на бутона за влизане в профила, потребителят отива в страница като на фигура 34, където е изписана неговата информация и има опцията да промени информацията по акаунта.

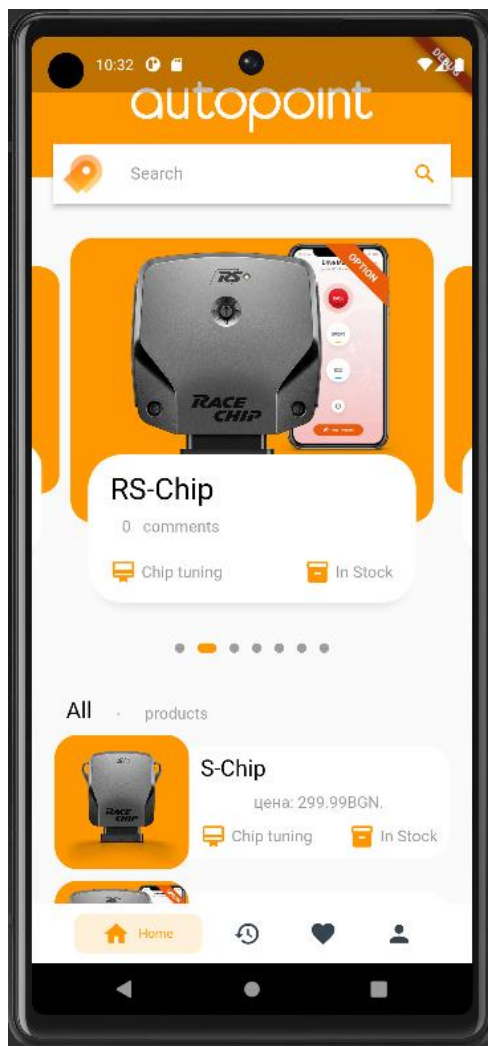


Фигура 43 Профилна страница

## ГЛАВА 6 РЪКОВОДСТВО ЗА ПОТРЕБИТЕЛЯ НА МОБИЛНОТО ПРИЛОЖЕНИЕ

### 6.1. Главна страница на приложението

На фигура 44 се вижда главната страница на приложението. На главната страница се намира търсачката, всички продукти и навигационното меню на приложението.

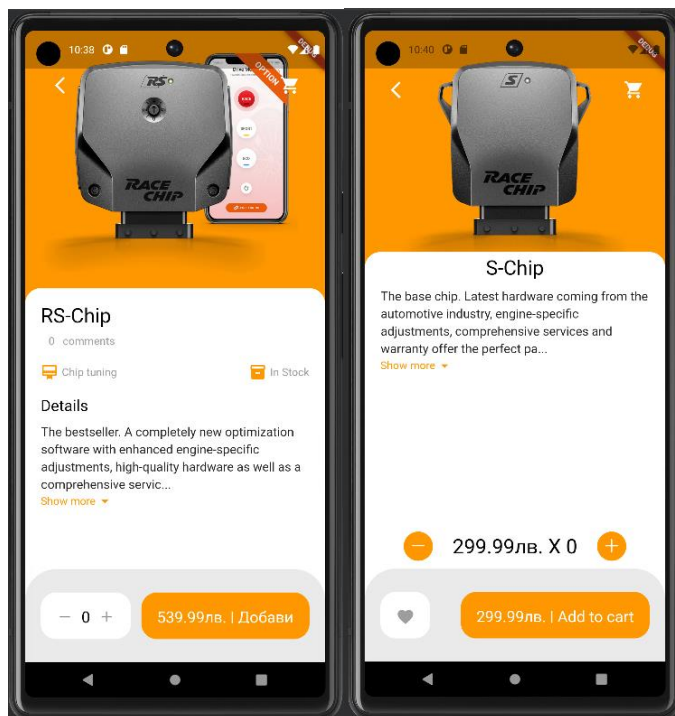


Фигура 44 Начална страница на мобилното приложение

### 6.2. Страници за описание на продукт

От фигура 45 се виждат двете страници за описание на продукт. В тази страница е изобразен продукта с неговата категория дали го има в наличност, детайлите, както и меню отдолу от където може да се добави дадена бройка на продукта в количката. В едно от двете менюта има бутон с формата на сърце, който

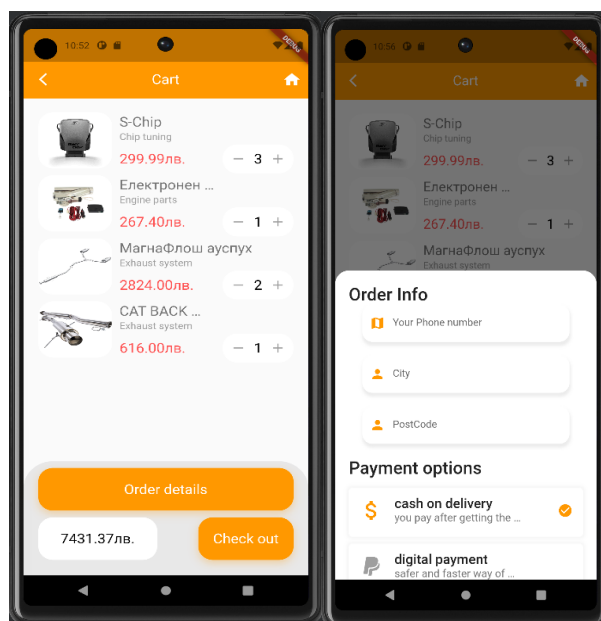
бутон ако свети в оранжево индикира че продукта е запазен в листа с желани продукти.



Фигура 45 Страници за детайли на продукт

### 6.3. Потребителската количка с продукти

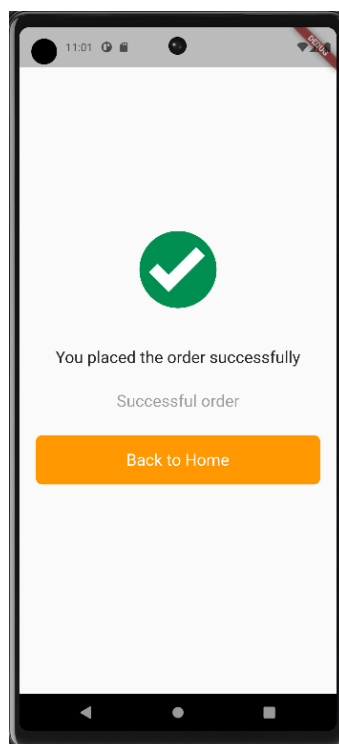
От потребителската количка потребителя вижда всички продукти в нея с техните цени и в какво количество ги е добавил, като количеството може да бъде променено от самата количка. От долу на екрана се намира менюто където се вижда общата цена на количката бутон за заплащане и бутон с детайли на поръчката. При натискане на бутоната с детайли излиза меню където потребителя трябва да попълни нужната информация. След попълването и може успешно да си поръча продуктите. На фигура 46 се вижда количката.



Фигура 46 количката на потребител

#### 6.4. Страница за потвърдена поръчка

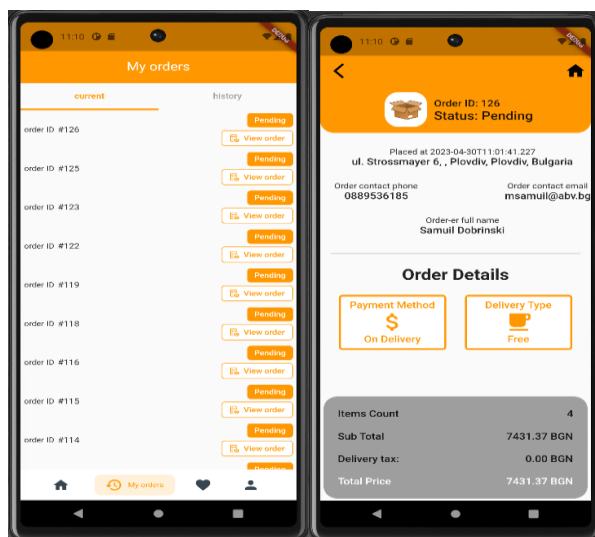
На фигура 47 се вижда страницата за потвърдена поръчка.



Фигура 47 Страница за потвърдена поръчка

## 6.5. Страница моите поръчки

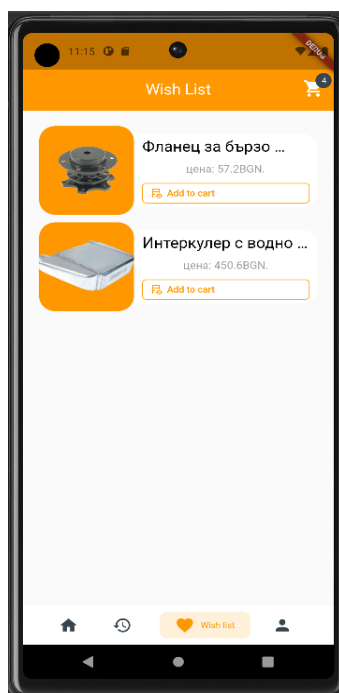
Това е страницата с историята на поръчките. В тази страница има две менюта. Първото е с поръчките които още не са изпратени а втората с вече изпратените. При натискане на бутона прегледай се отваря страницата с детайли на поръчката. На фигура 48 се виждат двете страници.



Фигура 48 Страница моите поръчки и страницата детайли на поръчка

## 6.6. Страница желани продукти

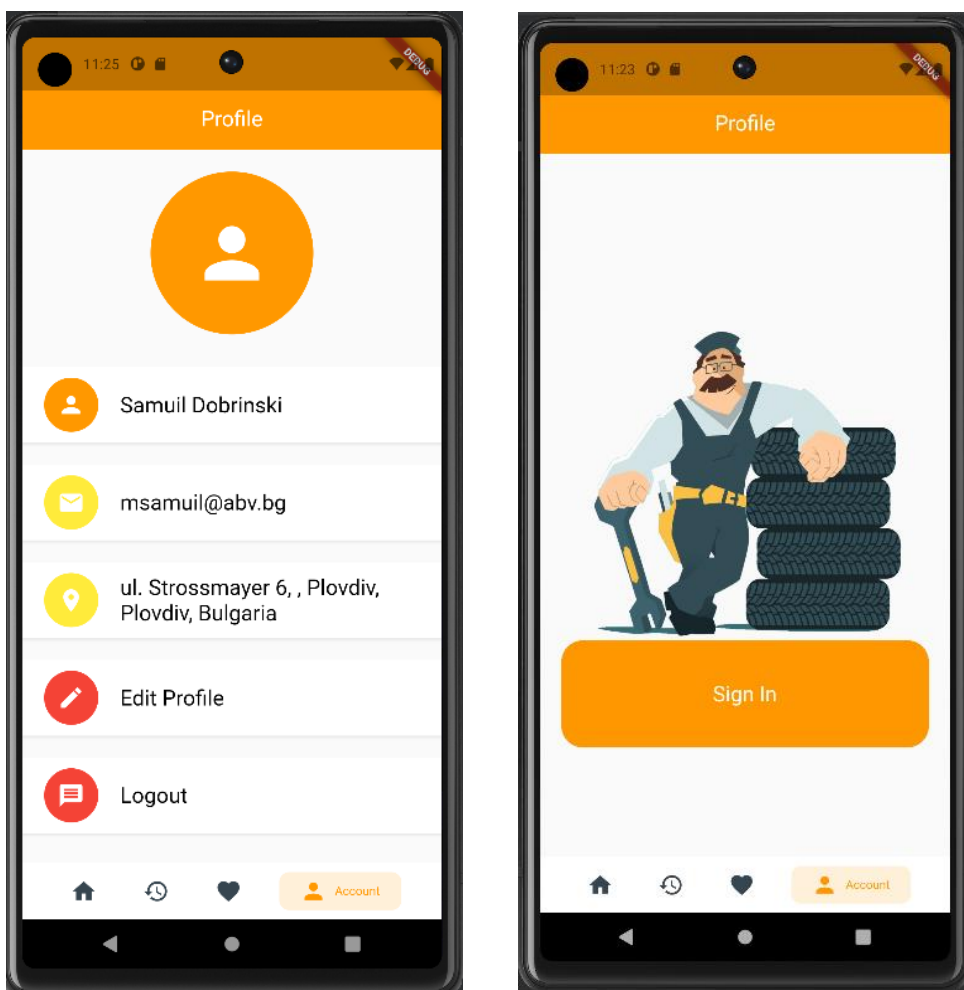
На фигура 49 се вижда страницата желани продукти, където се намират всички продукти добавени в този лист от потребителя. Освен че ги вижда потребителя може с едно натискане на бутона да добави някой от желаните продукти в количката.



*Фигура 49 Страница желани продукти*

## **6.7. Профилна страница**

На фигура 50 се вижда профилната страница където са изобразени данните на потребителя. От това меню потребителя може да си промени адреса като натисне на него или цялата информация от промени профила. Също така разполага с бутон за излизане от профила.

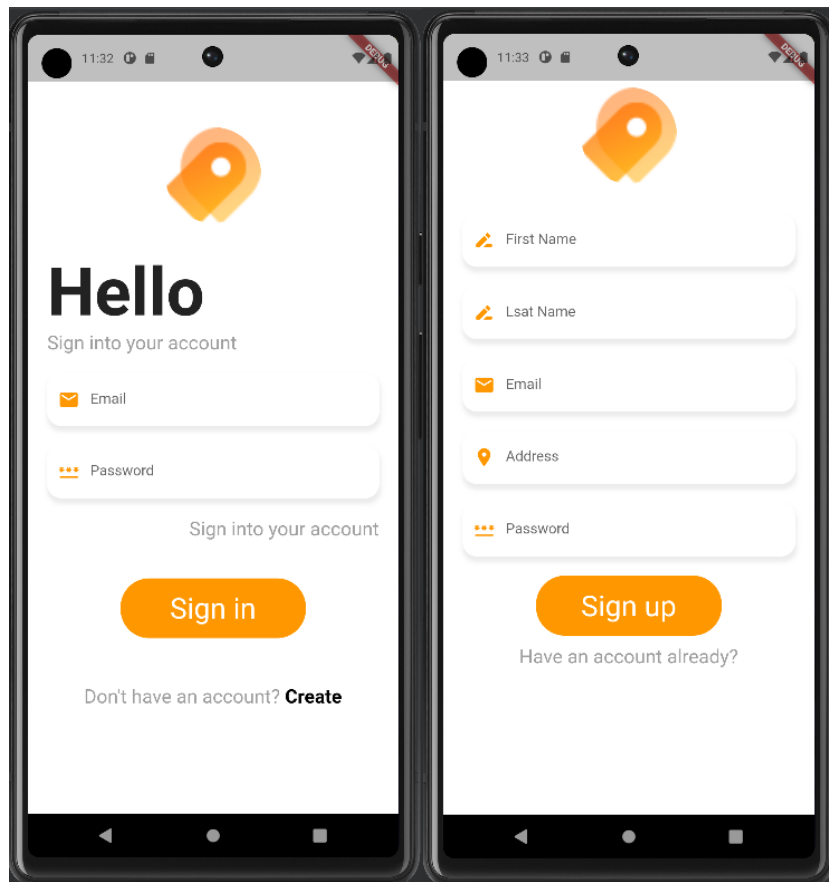


*Фигура 50 Профилна страница на анонимен и на влезнал потребител*

#### 6.8. Страници вход и регистрация

На фигура 51 се виждат страниците за вход и регистрация от където потребителя може да си създаде профил или ако има такъв да влезе в него.

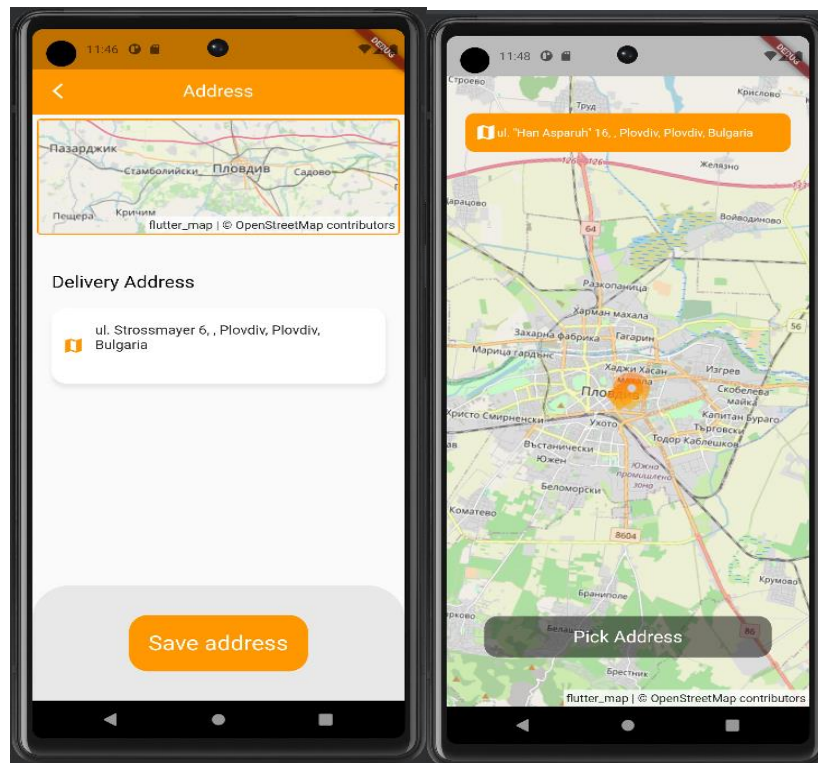




*Фигура 51 Страница вход и страница регистрация*

## **6.9. Страница за задаване на адрес**

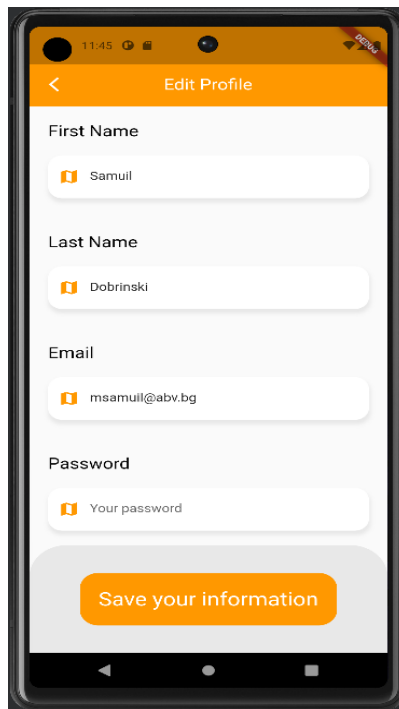
На фигура 52 се виждат страницата за подаване на адрес и картата на която можеш да си избереш адрес. От страницата с картата се задава адреса а от другата страница се запазва, като ако потребителя не харесва подадения адрес винаги може да го подаде ръчно написан.



Фигура 52 Страница за задаване и избиране на адрес

#### 6.10. Страница промяна на профила

На фигура 53 се вижда страницата за промяна на информация. От тук потребителя може да смени своето име, парола или имейл адрес.



*Фигура 53 Страница за промяна на профила*

## **ЗАКЛЮЧЕНИЕ**

В резултат на извършената работа по проучване на източници и използване на технологични решения, може да се твърди че основните поставени пред разработката задачи са изпълнени. В този аспект се разглеждат следните конкретни решения:

- Изработване външният вид на сайта.
- Реализация функционалностите на магазина за тунинг авточасти.
- Интегриране дейности за проверка на двигателите и стойностите на OBD модула.
- Изграждане на система от различни нива на достъп с потребителски профили и администраторски контрол.
- Изграждане на система за множество разплащания.
- Създаване на възможност в съответните потребителски и администраторски панели да има достъп до преглеждане и редактиране на отделните функционалности.
- Изработено е мобилно приложение, което улеснява достъпа до електронния магазин през мобилни устройства.

В допълнение на тези задачи, по време на разработка на приложението биват изградени и множество допълнителни функционалности:

- Бутон за смяна на езика.
- Раздел за детайлите на продукт, където потребител може да напише коментар за него.
- Работеща SMTP услуга, която изпраща реални имейли до потребителите и улеснява връзката с тях.
- Допълнителна опция в администраторския панел която променя статуса на чакащите за одобрение продукти.

В заключение може да се каже, че поставената пред разработката цел – проектиране и изграждане на електронен магазин за специфични тунинг авточасти е изпълнена. Разбира се, в бъдеще магазинът може да бъде доразвит с добавяне на допълнителни функционалности – разработка на мобилно приложение, поддържано от Android и IOS, създаване на чат асистент, който чрез изкуствен интелект да дава отговор на въпросите на даден клиент и евентуално модификация на изградените модули.

## ИЗТОЧНИЦИ

- Георги Кацаров, „Какво е ASP.NET Core и какво е MVC“, 11/06/2018г., прегледан на 18/01/2023г., достъпен на <https://softuni.bg/blog/asp-dot-net-core-and-mvc>
- Янев Румен, „Какво е ASP.NET Core и защо да я научим?“, 05/10/2020г., прегледан на 18/01/2023г., достъпен на <https://softuni.bg/blog/asp-net-core-article>
- Microsoft, „Overview of ASP.NET Core“, 15/11/2022г., прегледан на 18/01/2023г., достъпен на <https://learn.microsoft.com/bg-bg/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-3.1>
- Microsoft, „Introduction to Web Development with Blazor“, началото на 2023г., прегледан на 18/01/2023г., достъпен на <https://learn.microsoft.com/bg-bg/training/modules/blazor-introduction/2-what-is-blazor>
- Pragimtech, „What is Blazor“, 18/04/2020г., прегледан на 18/01/2023г., достъпен на <https://www.pragimtech.com/blog/blazor/what-is-blazor/>
- W3schools, „ASP AJAX“, 14/07/2020г., прегледан на 18/01/2023г., достъпен на [https://www.w3schools.com/asp/asp\\_ajax.asp](https://www.w3schools.com/asp/asp_ajax.asp)
- TutorialsPoint, „ASP.NET WP – View Engines“, 25/05/2013г., прегледан на 18/01/2023г., достъпен на [https://www.tutorialspoint.com/asp.net\\_wp/asp.net\\_wp\\_view\\_engines.htm](https://www.tutorialspoint.com/asp.net_wp/asp.net_wp_view_engines.htm)
- TutorialsPoint, „MVC Framework – Introduction“, 26/06/2022г., прегледан на 18/01/2023г., достъпен на [https://www.tutorialspoint.com/mvc\\_framework/mvc\\_framework\\_introduction.htm](https://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm)
- Ben Lutkovich, „database (DB)“, 15/04/2009г., прегледан на 18/01/2023г., достъпен на <https://www.techtarget.com/searchdatamanagement/definition/database>
- Oracle, „What is a Relational Database (RDBMS) ?“, началото на 2023г., прегледан на 18/01/2023г., достъпен на <https://www.oracle.com/database/what-is-a-relational-database/>
- MongoDB, „What is NoSQL?“, началото на 2023г., прегледан на 18/01/2023г., достъпен на <https://www.mongodb.com/nosql-explained>

- Andrew Zola, „What is Bootstrap?“, 08/08/2022г., прегледан на 18/01/2023г., достъпен на <https://www.techtarget.com/whatis/definition/bootstrap>
- Json.org, „Introducing JSON“, 15/04/2010г., прегледан на 18/01/2023г., достъпен на <https://www.json.org/json-en.html>
- Entity Framework Tutorial, „What is Entity Framework?“, 10/10/2018г., прегледан на 18/01/2023г., достъпен на <https://www.entityframeworktutorial.net/what-is-entityframework.aspx>
- Microsoft, „A tour of the C# language“, 12/13/2022г., прегледан на 18/01/2023г., достъпен на <https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/>
- MDN, „HTML: HyperText Markup Language“, 17/01/2023г., прегледан на 18/01/2023г., достъпен на <https://developer.mozilla.org/en-US/docs/Web/HTML>
- MDN, „JavaScript“, 13/12/2022г., прегледан на 18/01/2023г., достъпен на <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
- Computer Hope, „Visual Studio“, 06/07/2019г., прегледан на 18/01/2023г., достъпен на <https://www.computerhope.com/jargon/v/visual-studio.htm>
- Amazone, „What is an Api?“, Началото на 2023г., прегледан на 20/01/2023г., достъпен на <https://aws.amazon.com/what-is/api/>
- <https://www.youtube.com/>
- <https://stackoverflow.com/>
- <https://learn.microsoft.com/en-us/training/>

## ПРИЛОЖЕНИЕ 1

Program.cs файлът на програмата :

```
var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllersWithViews();

builder.Services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)

    .AddCookie(options => { options.LoginPath =
"/User/Login"; options.ExpireTimeSpan =
TimeSpan.FromMinutes(20); }); //

builder.Services.AddLocalization(opt => { opt.ResourcesPath
= "Resources"; }); //

builder.Services.AddMvc()

    .AddViewLocalization

(Microsoft.AspNetCore.Mvc.Razor.LanguageViewLocationExpanderFormat.Suffix)

    .AddDataAnnotationsLocalization(); //

builder.Services.Configure<RequestLocalizationOptions>(
    options =>
    {
        var supportedCultures = new List<CultureInfo>
        {
```

```

        new CultureInfo("en"),

        new CultureInfo("bg"){ NumberFormat = new
NumberFormatInfo(){NumberDecimalSeparator = "." }

    }

};

options.DefaultRequestCulture = new
RequestCulture("bg");

options.SupportedCultures = supportedCultures;
options.SupportedUICultures = supportedCultures;

});//
var app = builder.Build();
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();

app.UseAuthentication();//

app.UseRequestLocalization(app.Services.GetRequiredService<
IOptions<RequestLocalizationOptions>>().Value);//

```



```
app.UseAuthorization();
```

```
app.MapControllerRoute(
```

```
    name: "default",
```

```
    pattern: "{controller=Home}/{action=Index}/{id?}");
```

```
app.Run();
```

## ПРИЛОЖЕНИЕ 2

Context.cs, който инициализира връзката между приложението и базата данни :

```
public class Context : DbContext
{
    public DbSet<User> Users { get; set; }
    public DbSet<Product> Products { get; set; }
    public DbSet<Comment> Comments { get; set; }
    public DbSet<CartProduct> CartProducts { get; set; }
}

public DbSet<Order> Orders { get; set; }

public Context() : base("Data
Source=SQL8002.site4now.net;Initial
Catalog=db_a91afa_autopoint;User
Id=db_a91afa_autopoint_admin;Password= ")
{
    Users = this.Set<User>();
    Products = this.Set<Product>();
    Comments = this.Set<Comment>();
    CartProducts = this.Set<CartProduct>();
    Orders = this.Set<Order>();
}
}
```

### ПРИЛОЖЕНИЕ 3

Скрипта, който осъществява плащания с PayPal:

```
<script src="https://www.paypal.com/sdk/js?client-  
id=Ae3tHpr-  
yczbhy6qceXfoIr5lNOTuk8SJdELlOd9cqEMEVPB7WSA4RNf_y-  
pYuxD6zxI0bFoPpYCH2tz&enable-funding=venmo&currency=EUR"  
data-sdk-integration-source="button-factory"></script>
```

```
<script>
```

```
function initPayPalButton() {  
  
    var total =  
    (parseFloat(document.getElementById("total-  
price").value.replace(",","."))/2).toFixed(2);
```

```
    paypal.Buttons({  
  
        style: {  
  
            shape: 'rect',  
  
            color: 'gold',  
  
            layout: 'vertical',  
  
            label: 'paypal',
```

```
    },
```

```
    createOrder: function(data, actions) {  
  
        return actions.order.create({
```

```

        purchase_units:
[{"amount":{"currency_code":"EUR","value":total}}]

        });

    },

    onApprove: function(data, actions) {

        return
actions.order.capture().then(function(orderData) {

            // Full available details

            console.log('Capture result',
orderData, JSON.stringify(orderData, null, 2));

            // Show a success message within this
page, e.g.

            const element =
document.getElementById('paypal-button-container');

            element.innerHTML = '';

            element.innerHTML = '<h3>Thank you for
your payment!</h3>';

            document.getElementById("payment-
method").value="paypal";

            document.getElementById("checkout-
form").submit();

        });

```

```
    },  
  
    onError: function(err) {  
        console.log(err);  
    }  
}).render('#paypal-button-container');  
  
}  
  
initPayPalButton();  
</script>
```