****

**Природо-математическа гимназия**

**„Васил Друмев“**

**Гр. Велико Търново**

**ДИПЛОМЕН  ПРОЕКТ**

за придобиване на трета степен на професионална квалификация

Професия: „Приложен програмист“

Специалност: „Приложно програмиране“

**Тема: Изграждане на комплекс от методи, средства и подходи при разработване на приложение**

**„СЕРВИЗ ЗА ТЕЛЕФОНИ „PHONE-SERVICE““**

Ученик:                                                         Ръководител-консултант:

Боян Росенов Божанов   Милен Василев

# СЪДЪРЖАНИЕ

Съдържание

[СЪДЪРЖАНИЕ - 2 -](#_Toc163461117)

[Увод - 3 -](#_Toc163461118)

[Теоретична част - 4 -](#_Toc163461119)

[1. Технологии за разработка на уеб-базирани приложения - 4 -](#_Toc163461120)

[1.1 Особености на уеб-базираните приложения - 4 -](#_Toc163461121)

[1.2 Основни технологии за разработка на уеб-базирани приложения - 4 -](#_Toc163461122)

[Практическа част - 11 -](#_Toc163461123)

[1. Концепция - 11 -](#_Toc163461124)

[1.1 Задание - 11 -](#_Toc163461125)

[1.2 Допълнителни разработки - 12 -](#_Toc163461126)

[2. Реализация - 12 -](#_Toc163461127)

[2.1 База данни - 12 -](#_Toc163461128)

[2.2 Потребителски интерфейс и бизнес логика - 20 -](#_Toc163461129)

[Заключение - 38 -](#_Toc163461130)

[Използвани източници - 39 -](#_Toc163461131)

[Приложения - 40 -](#_Toc163461132)

# Увод

Съвременните сервизи за мобилни телефони се нуждаят от ефективни и удобни решения за управление на своите операции, включително поддръжка на информация за предлаганите услуги и контрол на процеса на ремонтиране. В този контекст представям документацията за проекта "СЕРВИЗ ЗА ТЕЛЕФОНИ „Phone-service“" - система, предназначена за оптимизиране на дейността на сервизи за мобилни телефони.

Целта на дипломната работа е да се разработи и документира комплексна информационна система, която да обслужва потребителите, служителите и администраторите на такива сервизи.

В основата на "СЕРВИЗ ЗА ТЕЛЕФОНИ „Phone-service“" е двукомпонентна архитектура, състояща се от база данни и приложение със слой, комуникиращ с нея. Проектът предоставя разнообразни функционалности, включително управление на потребители, ремонти, поръчки и други.

В настоящата документация ще бъде представена подробна информация за проектирането и функционалностите на "СЕРВИЗ ЗА ТЕЛЕФОНИ „Phone-service“", както и реализацията на различните компоненти и техните взаимодействия. Ще бъде разгледано използването на валидация на данни и падащи менюта, съобразено с изискванията на проекта.

Тази документация е съставена с цел да предостави ясен обзор на проекта ""СЕРВИЗ ЗА ТЕЛЕФОНИ „Phone-service“" и да допринесе за разбирането на неговото предназначение и възможности. В следващите раздели ще бъдат представени подробности за функционалностите и реализацията на системата, както и заключение на извлечените поуки и резултати от проекта.

За реализацията на този проект са използвани HTML, CSS, Bootstrap, JavaScript, C#, ASP.NET Core, Entity Framework Core.

# Теоретична част

## Технологии за разработка на уеб-базирани приложения

### Особености на уеб-базираните приложения

Уеб-базираните приложения се отличават с редица особености, които ги правят широко използвани и предпочитани за различни видове софтуерни приложения. Техните основни характеристики включват достъпност от всякакви устройства, които имат интернет връзка, включително компютри, таблети и смартфони. Това ги прави изключително удобни за потребителите, които могат да използват приложенията по всяко време и от всяко място. Освен това, уеб приложенията са лесни за разпространение и актуализация, тъй като не изискват инсталация на потребителския компютър, а новите версии се разпространяват автоматично. Основният език за програмиране за създаване на уеб приложения е HTML (HyperText Markup Language), като за визуалния дизайн се използват CSS (Cascading Style Sheets), а за логиката на приложението - различни програмни езици и фреймуърки като JavaScript, PHP, Python, ASP.NET и други. Уеб-базираните приложения също така се отличават с гъвкавост, което позволява лесното им разширяване и адаптиране към нуждите на потребителите и бизнеса.

### Основни технологии за разработка на уеб-базирани приложения

* + 1. **HTML**

HTML (HyperText Markup Language) е стандартен език за програмиране, който се използва за създаване на структура и съдържание на уеб страници. Той е основен елемент от уеб разработката и позволява на разработчиците да дефинират различни елементи и техни стилове.

HTML играе ключова роля в определянето на основната структура на уеб страници. Той дефинира елементи като заглавия (<h1>, <h2>… <h6>), параграфи (<p>), секции (<section>), списъци (<ul>, <ol>), таблици (<table>) и други, които оформят визуалния интерфейс на приложението. HTML елементите са групирани по двойки и имат отварящ таг <h1> и затварящ таг </h1>.

Чрез HTML се създават форми (фиг. 1), които позволяват на потребителите да въвеждат данни. Тези форми се използват за регистрация на потребители, вход в системата, създаване на поръчки и други действия.

HTML позволява създаването на интерактивни елементи като връзки (link), бутони (button) и менюта (menu), които подобряват навигацията и удобството на потребителите.



Фигура 1 - Форма за създаване на модел ревю.

Изображения и медия: Чрез HTML можем да вградим изображения, видео и аудио файлове в уеб страниците. Това допълнително обогатява съдържанието на страниците и ги прави по-атрактивни за потребителите.

HTML включва семантични елементи като `<header>`, `<footer>`, `<nav>`, `<article>` и други, които не само подобряват четливостта на кода, но и помагат за SEO (Search Engine Optimization) оптимизацията на уеб страници.

* + 1. **CSS**

CSS (Cascading Style Sheets) е език за стилове, който играе ключова роля в стилизирането на уеб страници. Този език предоставя удобен механизъм за прилагане на различни стилове към HTML елементите, което дава възможност за пълен контрол върху техните визуални аспекти.

Един от най-важните аспекти на CSS е неговата способност да дефинира външния вид на HTML елементите. Това включва определяне на цветове, шрифтове, размери, разположение и други визуални атрибути, които създават уникални и привлекателни уеб дизайни.

Със CSS можем да контролираме не само визуалния стил на елементите, но и техния макет и позициониране на уеб страницата. Това включва задаване на размери, отстъпи, отстояния, рамки и други части на оформлението, които подобряват употребата и привличат вниманието на потребителите.

Използвайки селектори, CSS ни позволява да идентифицираме и стилизираме конкретни елементи на уеб страницата. Те могат да бъдат елементарни, като името на тага, класове, идентификатори, псевдокласове и други, което ни дава гъвкавостта да стилизираме елементите според нуждите на проекта.

Освен това, CSS позволява създаването на адаптивни дизайни, които се приспособяват към различни устройства и екрани. Това се постига чрез “Media Queries”, които задават различни стилове в зависимост от различни параметри на устройството, като широчина на екрана или ориентацията на устройството. Така уеб страниците могат да бъдат оптимизирани за различни размери на екрана и да предложат приятно изживяване на потребителите.

* + - 1. **Bootstrap**

Bootstrap представлява интегриран фреймуърк за уеб разработка, предлагащ разнообразни компоненти, стилове и скриптове, които значително улесняват създаването на модерни уеб приложения. Гъвкавата грид система на Bootstrap позволява лесно разпределяне на елементите на страницата в различни колони и редове, това води до създаване на responsive /отзивчиви/ макети, които се приспособяват към различни екрани.

Фреймуъркът предлага широк набор от готови компоненти като навигационни менюта, форми, бутони, карти, модални прозорци и много други. Тези компоненти се интегрират лесно в уеб приложението и могат да бъдат персонализирани според конкретните изисквания на проекта.

Bootstrap включва и различни CSS и JavaScript плъгини, които допълнително разширяват функционалността на уеб приложенията. Например модални прозорци, карусели, валидация на форми и други, които значително подобряват интерактивността и удобството за потребителите.

С оглед на използването на “Media Queries” и класове за адаптивен дизайн, Bootstrap 5 е напълно отзивчив и се приспособява към различни размери на екрани и устройства. Това осигурява уеб приложенията да изглеждат добре и да функционират ефективно на всички устройства.

* + 1. **Java Script**

JavaScript (JS) представлява високо ниво език за програмиране, използван за придаване на интерактивност и динамично поведение на уеб страниците. Той предлага мощни възможности за манипулиране на HTML и CSS, обработка на събития и комуникация със сървъра.

Един от основните аспекти на JavaScript е манипулирането на DOM (Document Object Model) на уеб страницата. Този процес позволява динамично добавяне, промяна и изтриване на елементи и тяхното съдържание. JavaScript може да създава нови елементи, да променя стиловете им, да обработва събития и да извършва други действия.

Също така, JavaScript позволява обработката на различни събития като кликвания, навигация с мишката, въвеждане на данни от потребителя и други. Това позволява създаването на интерактивни уеб приложения, които динамично реагират на действията на потребителите.

Освен това, JavaScript позволява взаимодействие със сървъра чрез изпращане на HTTP заявки. Това се осъществява чрез AJAX технологията, която позволява зареждане на данни и актуализация на съдържанието на страницата без презареждане на цялата страница.

* + - 1. **jQuery**

jQuery е библиотека за JavaScript, която предоставя удобни инструменти за манипулиране на DOM, обработка на събития и изпращане на AJAX заявки към сървъра. Тази библиотека има кратък и лесен за разбиране синтаксис, който значително опростява разработката на уеб приложения и намалява нуждата от повторно използване на код.

Един от ключовите аспекти на jQuery е неговият мощен механизъм за селектиране, който позволява бързо и лесно намиране на елементите в DOM дървото, използвайки CSS подобни селектори.

Примери:

* $("p") // Избира всички <p> елементи на страницата
* $(".myClass") // Избира всички елементи с клас "myClass"
* $("#myId") // Избира елемент с идентификатор "myId"
* $("[type=’text’]") // Избира всички елементи с атрибут type, чиято стойност е "text"

Освен това, jQuery предоставя обширен набор от методи за манипулиране на DOM елементите. Тези методи позволяват добавяне, премахване, модификация и други манипулации върху елементите, което прави кода по-четим и ефективен.

Когато става въпрос за обработка на събития, jQuery осигурява удобен начин за това. Той предоставя лесни и интуитивни методи за обработка на различни събития като кликвания, движения на мишката, въвеждане на данни и други. Това улеснява добавянето на интерактивност към уеб приложенията.

* + 1. **C#**

C# е мощен и елегантен език за програмиране, разработен от Microsoft, който се използва за разработване на разнообразни видове софтуерни приложения, включително уеб, десктоп и мобилни приложения.

Той е обектно-ориентиран език, който поддържа основните принципи на ООП като наследяване, капсулиране и полиморфизъм, което прави кода по-структуриран, лесен за поддръжка и разширяем.

C# се използва за създаване на логиката на приложението, включително обработка на заявките от клиентите, манипулиране на данни от базата данни, изпращане на отговори към клиентите и други. Със C# се създава и бизнес логика, като валидация на входните данни, изчисления, обработка на бизнес правила и други.

Той също така се използва за интеграция с бази данни, като например Microsoft SQL Server, и други външни услуги, което позволява управление на данни и взаимодействие с външни системи.

* + 1. **ASP.NET Core**

ASP.NET Core е отворена и мултиплатформена уеб рамка, разработена от Microsoft, която предоставя среда за изграждане на съвременни и издръжливи уеб приложения. Тази рамка е проектирана да бъде гъвкава и модулна, като позволява на разработчиците да избират само тези компоненти, които са им необходими за техните приложения.

ASP.NET Core е създаден да работи с .NET Core, мултиплатформеният и отворен фреймуърк за разработка на софтуерни приложения. Това осигурява гъвкавост и мобилност, позволявайки разработчиците да използват предимствата на .NET платформата независимо от операционната система, на която работят.

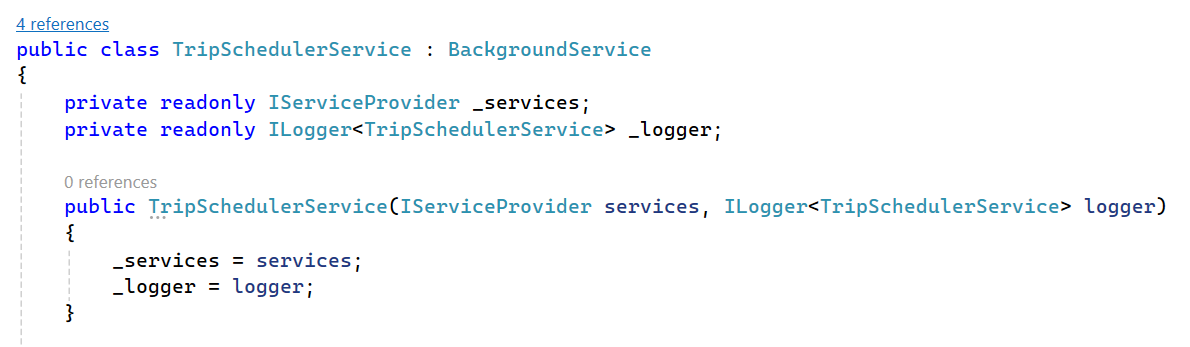
С висока производителност и способност за разширения, ASP.NET Core е оптимизиран за изграждане на уеб приложения, които издържат на висок натоварване и изискват бърз отговор. Рамката включва вградена поддръжка за сигурност, като предлага вградени механизми за управление на идентичността и защита от различни видове атаки.

* + - 1. **BackgroundService**

BackgroundService е клас, предоставен от ASP.NET Core, който позволява изпълнението на фонови задачи в рамките на уеб приложенията. Тези задачи могат да бъдат дългосрочни и да се изпълняват извън основния поток на приложението, без да се нарушава неговата производителност.

Класът BackgroundService се наследява и използва за дефиниране на задължения, които се изпълняват в бекграунд. Например, фонови задачи като обработка на заявки, изпращане на имейли, синхронизация с външни услуги или почистване на данни могат да бъдат автоматизирани с помощта на този клас.

Работата с BackgroundService е лесна благодарение на интеграцията му с инфраструктурата за хостване в ASP.NET Core, като така приложението може да стартира фонови задачи при своя стартиране и да ги прекрати при изключване.



Фигура 2 – Пример за инициализация на BackgroundService.

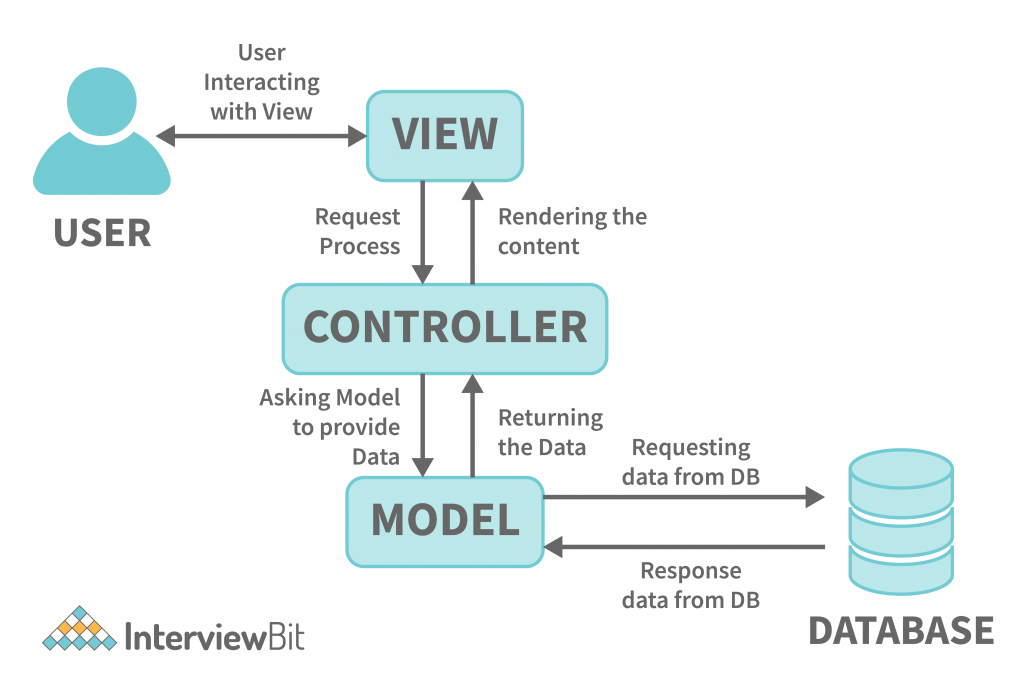
* + - 1. **MVC**

MVC (Model-View-Controller) е архитектурен модел (фиг. 3) за проектиране на софтуерни приложения, който цели да раздели приложението на три основни компонента: модел, изглед и контролер. Тази архитектурна модел помага за подобряване на поддръжката, разширяемостта и възможността за повторно използване на кода.

Моделът представлява частта на приложението, която съдържа логиката за обработка на данни. Това включва извличане на данни от базата данни, обработка на информацията и предоставяне на необходимите данни за изгледите и контролерите.

Изгледът представлява частта на приложението, която се грижи за визуалното представяне на информацията. Това включва HTML шаблоните, които се използват за генериране на уеб страниците, както и CSS стиловете за оформяне на външния вид на страниците.

Контролерът представлява частта на приложението, която приема заявките от потребителите, обработва ги и извиква съответната логика за обработка на данните в моделите. След това контролерът избира подходящия изглед за представяне на резултатите на заявката.



Фигура 3 – Схема на MVC.

* + 1. **Entity Framework Core**

Entity Framework Core (EF Core) представлява модерен ORM (Object-Relational Mapping) инструмент, създаден от Microsoft, който предоставя удобен и лесен начин за взаимодействие с бази данни в .NET приложенията. Той е лек и гъвкав фреймуърк, който дава възможност на разработчиците да работят с данните в базите данни, като използват обектно-ориентирания подход.

Работейки по принципа на ORM, EF Core позволява на програмистите да манипулират данните в базите данни, като използват обекти и класове, вместо да се занимават с писане на SQL заявки директно. Този подход значително улеснява работата с данните и прави кода по-четим и по-лесен за поддръжка.

Разработчиците могат да дефинират модел на данни с помощта на EF Core, който отразява структурата и връзките между обектите в базата данни. Този модел може да бъде дефиниран с използване на атрибути или конвенции, които автоматично генерират схемата на базата данни.

За конфигуриране на връзката с базата данни, EF Core предоставя възможност за настройка на параметрите като типа на базата данни, връзката със сървъра, потребителските имена и пароли и други.

LINQ (Language Integrated Query) се използва от EF Core за създаване на заявки към базата данни. Това позволява на разработчиците да пишат заявки към данните по естествен и декларативен начин, като използват обекти и методи, които са част от езика на програмиране.

Също така, EF Core предлага вграден механизъм за миграции на базата данни, който улеснява управлението и актуализацията на структурата на базата данни чрез код-базирани миграции.

# Практическа част

## Концепция

### Задание

* + 1. **Функционалност**

Целта на дипломния проект е да се направи система, която служи за подпомагане дейността на сервиз за мобилни телефони, която да се използва за поддържане на информация за предлаганите услуги и следене на процеса на ремонта в сервиза.

Задачи, които трябва да бъдат решени:

* В системата да има четири вида потребители: администратор, служители, клиенти и гости (посетители). Първите три вида да са регистрирани и да имат достъп до съответната функционалност само след вход в системата. Гостите да могат само да разглеждат предлаганите услуги.
* Сайтът да има един администратор. Той да е първоначално зададен и да отговаря за данните за потребителите. Същият да разполага с изглед, в който вижда ролята на всеки един потребител. Чрез него той да може да я променя. Администраторът да не може да повиши друг потребител в администратор. Освен това, той трябва да има възможности за тресне по потребителско име, собствено име, фамилия или филтриране на потребителите по роли.
* Администраторът да разпределя постъпилите поръчки за ремонти на служителите. Информацията за предлаганите ремонти да е достъпна за четене от всички потребители и да може да се променя само от администратора.
* Видовете ремонти да се характеризират само с название. Имената да са зададени предварително и никой няма право да ги променя.
* Марката на телефоните да се характеризира само с название. Имената да са предварително въведени и не могат да се променят от никого.
* Моделите да са също предварително зададени и да не подлежат на промяна. Всеки модел принадлежи към някоя марка. Също така за всеки модел първоначално да е заложена информация какви ремонти са допустими.
* Служителите трябва да въвеждат информацията за поръчката. След въвеждане статусът на поръчката автоматично се определя от системата на „незапочната". Клиентът може да следи статуса по всяко време. След като една поръчка е завършена, той има възможност да даде оценка за качеството на услугата. Един клиент не може да има повече от една поръчка едновременно. Цената на поръчката се определя от цената на ремонта и типа на поръчката. Ако е нормална, цената й съвпада с тази на ремонта; ако е бърза, към цената на ремонта се добавят 50% и ако е експресна, цената се удвоява. Служителите и администраторът могат да сортират поръчките по дата и да ги филтрират по тип и статус.
* Типовете поръчки да се характеризират с название, да се въведат предварително и да не могат да се променят.
* Статусите на поръчките да имат само название. Също както типовете са зададени предварително и не подлежат на промяна.
* Всяка форма в приложението трябва да е валидирана, т.е. не се приемат нелогични данни, като например отрицателни цени или празни символни низове. Където е удачно да се използва падащо меню за избор между възможности.

### Допълнителни разработки

Като допълнително към условието е разработена начална страница, която да подканва посетителите на сайта да си направят регистрация. Има секция с оценени поръчки. Показват се предлаганите услуги за ремонти и съответните телефони, на които могат да се извършат.

За регистрираните потребители е направена страница, с която могат да проверят колко евентуално ще им струва поправката на телефон.

Служителите и клиентите могат да виждат техните заявки в отделни панели.

## Реализация

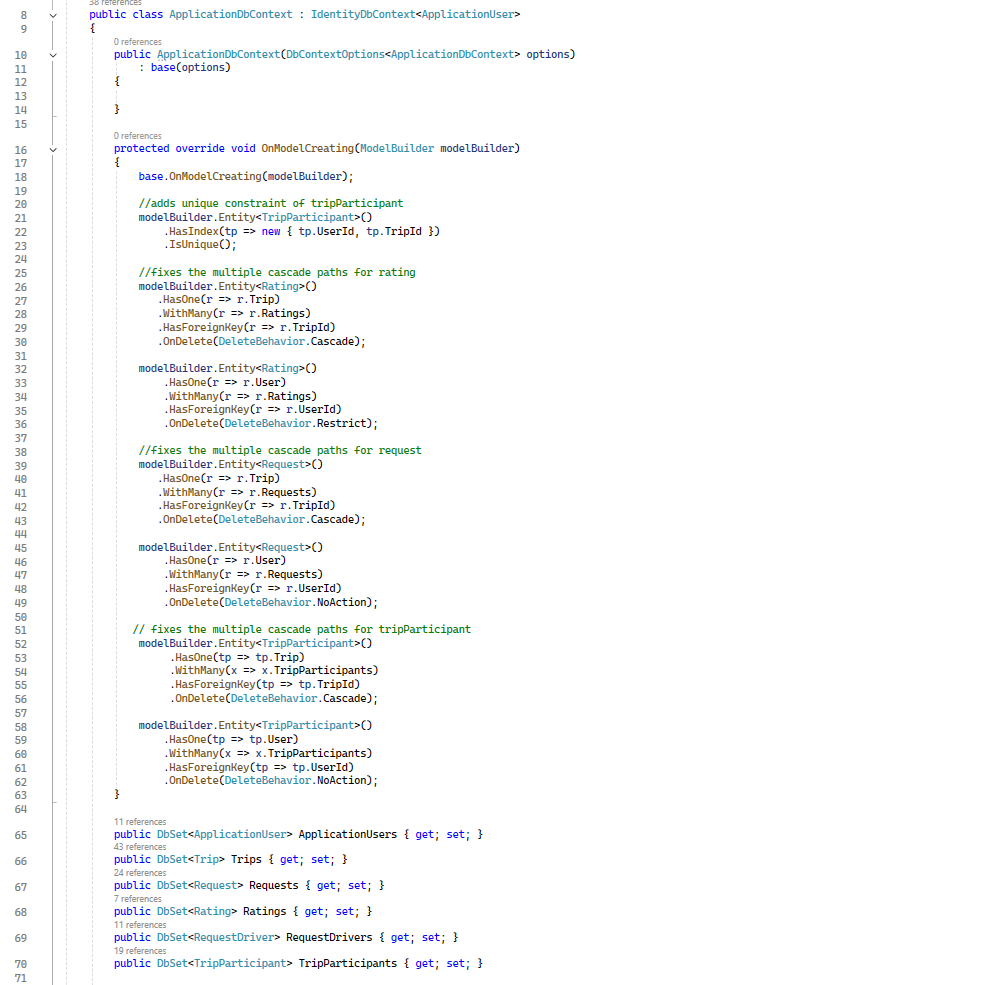
За успешно реализиране трябва да се инсталират “NuGet” пакети:

* Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore
* Microsoft.AspNetCore.Identity.EntityFrameworkCore
* Microsoft.AspNetCore.Identity.UI
* Microsoft.EntityFrameworkCore.SqlServer
* Microsoft.EntityFrameworkCore.Tools

### База данни

По условие дипломният проект се състои от два основни компонента като единия от тях е база данни. За тази цел е използван Microsoft SQL Server в комбинация с Entity Framework Core, в контекста на ASP.NET Core приложение. Това позволява създаване на модел на данните, управление на базата данни и извършване на операции като създаване, четене, актуализиране и изтриване на данни по лесен и удобен начин. Entity Framework Core предоставя мощен ORM (Object-Relational Mapping) инструмент, който улеснява взаимодействието с базата данни, като позволява манипулирането на данните с помощта на обекти и класове, вместо да се пишат SQL заявки директно.

* + 1. **Database Context**

****

Фигура 4 – Структура на ApplicationDbContext

За създаване на базата данни е направен класът „ApplicationDbContext“. Той наследява класа „IdentityDbContext“, това се прави, защото проекта предполага наличието на потребители и техните роли.

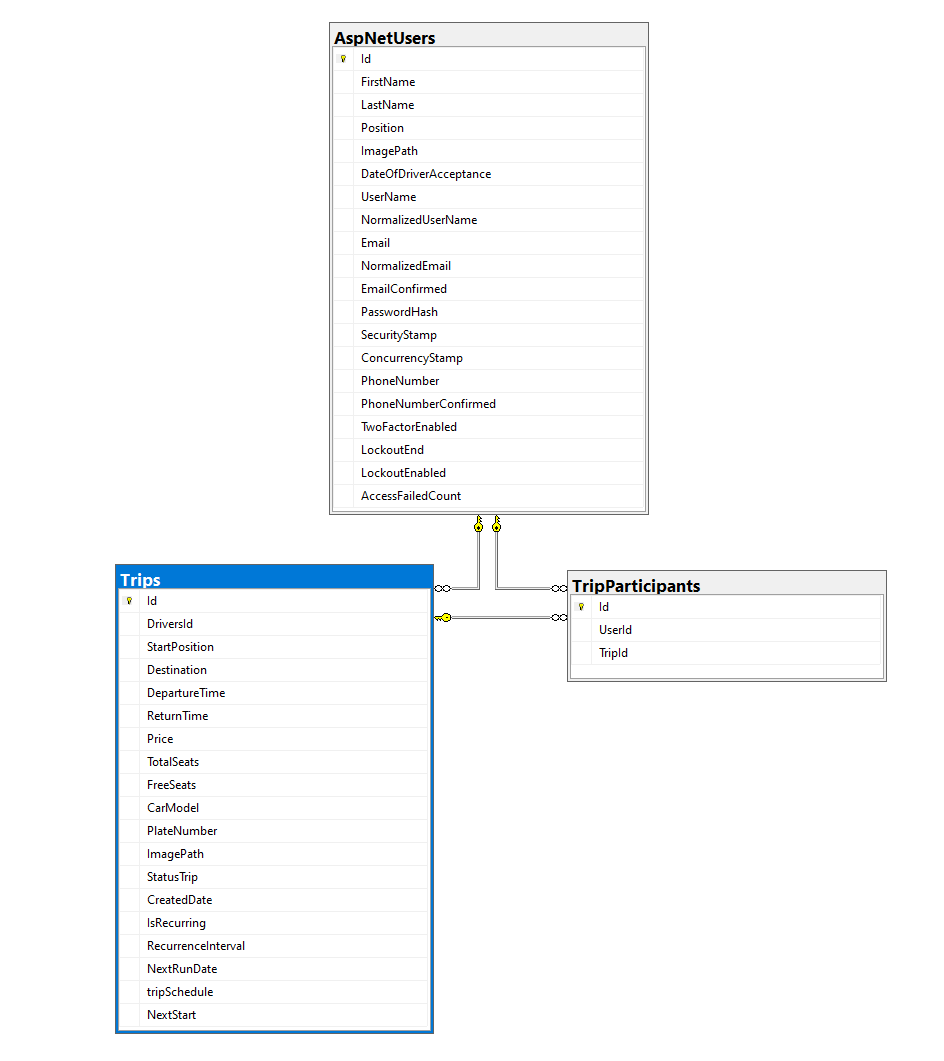
Базата от данни се състои общо от 13 таблици. 7 от тях са автоматично генерирани (точка 2.1.2), 2 от тях са информативни таблици (точка 2.1.3.5) и 6 създадени в процеса на разработка (точка 2.1.3).

Структурата на контекста е показа на Фиг. 4.

Редове 10-14 – това е конструкторът, който задава основните настройки на контекста чрез параметъра DbContextOptions.

Редове 16-63 – е презаписан метода OnModelCreating. Той се използва за дефиниране на връзките и ограниченията между таблиците в базата данни. С „HasOne“ и „WithMany“ методите се дефинират връзките между „TripParticipant” и „Trip“ и „ApplicationUser“. Специфичното тук е, че „TripParticipant” e междинна таблица използвана за свързване на уникални записи между един пътник и едно пътуване (фиг. 5).

Редове 65-70 – това е колекцията от класове, който са създадени в базата данни като таблици.



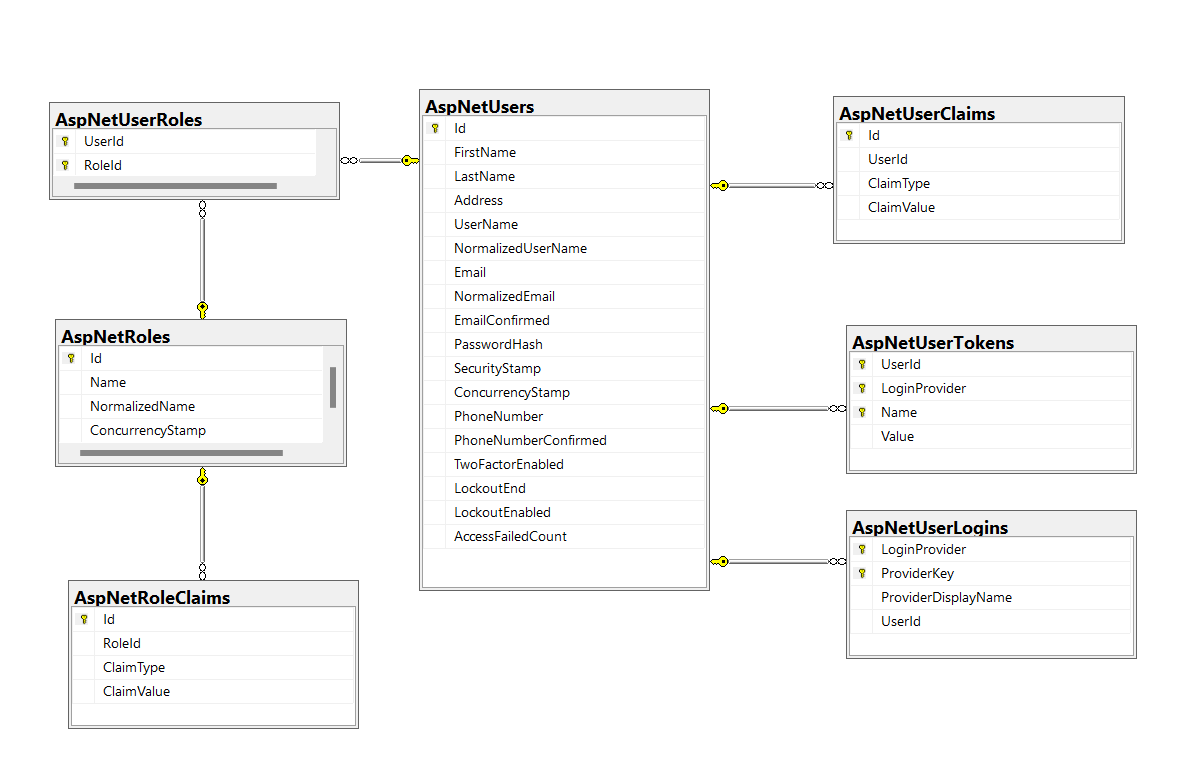
Фигура 5 – Специфичната връзка между таблиците „ApplicationsUser“ и „Trip“

* + 1. **Автоматично генерирани таблици**

Седемте автоматично генерирани таблици (фиг. 6):

* AspNetUsers – Съдържа информация за потребителите в системата.
* AspNetRoles – Съхранява ролите на потребителите.
* AspNetUserRoles – Установява връзката между потребителите и техните роли.
* AspNetRoleClaims – Съдържа допълнителна информация за ролите.
* AspNetUserClaims – Съдържа допълнителна информация за потребителите.
* AspNetUserLogins – Съхранява информация за външните идентификатори на потребителя.
* AspNetUserTokens – Съхранява токени за потвърждаване на идентичността на потребителите.

В проекта се използват само “AspNetUsers”, “AspNetRoles”, “AspNetUserRoles”.

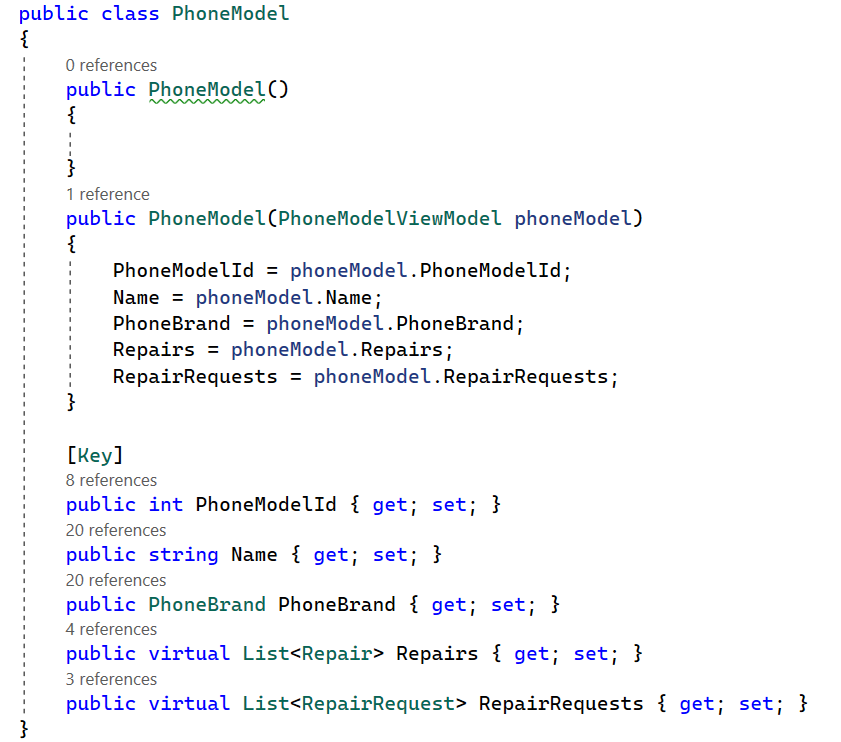


Фигура 6 – Структура на автоматично генерираните таблици.

* + 1. **Ръчно добавени таблици**
       1. **PhoneModels**

Описват се с:

* Название (Name)
* Марка (PhoneBrand)



Фигура 7 – Моделът на PhoneModel.

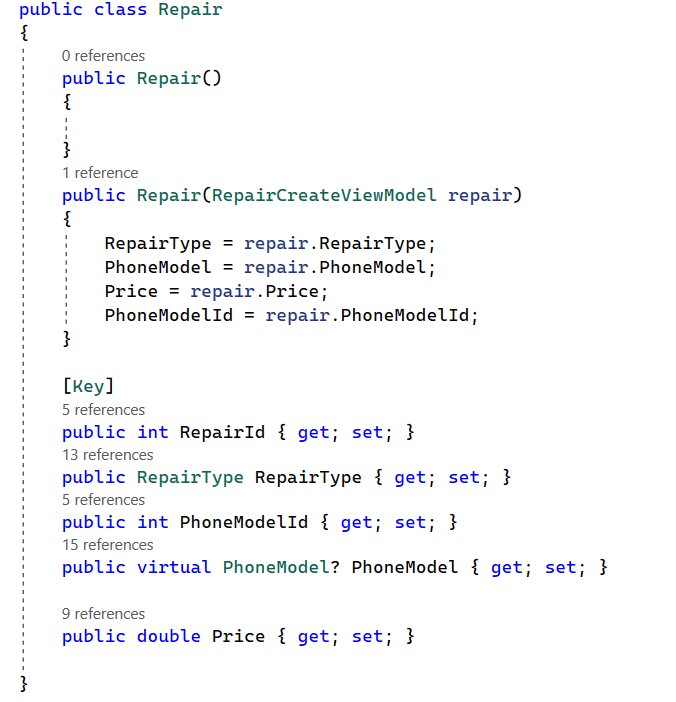
Има два конструктора – единият е празен, а другият приема като параметър “PhoneModelViewModel”, за да се преобразува от View Model до DBM (DataBaseModel).

Полетата “Repairs” и „RepairRequest“ осигуряват връзка между таблиците „PhoneModel“ – “Repair”, “PhoneModel” – “RepairRequest” като връзката и с двете е едно към много.

* + - 1. **Repairs**

Характеризират се с:

* Вид (RepairType)
* Марка и модел, за които се предлага ремонта (PhoneModel)
* Цена (Price)



Фигура 8 – Моделът на Repair.­

Има два конструктора – единият е празен, а другият приема като параметър “RepairCreateViewModel”, за да се преобразува от View Model до DBM (DataBaseModel).

Полетата “PhoneModelId” и „PhoneModel“ осигуряват връзка между таблиците „PhoneModel“ – “Repair” като връзката е едно към много.

* + - 1. **RepairRequests**

Те се характеризират с:

* Пореден номер (RepairRequestId)
* Тип (RepairRequestType)
* Клиент (ClientUser)
* Служител (WorkerUser)
* Дата (Date)
* Вид ремонт (RepairType)
* Марка и модел на телефона (PhoneModel)
* Допълнително описание (Description)
* Статус (Status)
* Оценка (Rating)
* Цена (Price)



Фигура 9 – Моделът на RepairRequest.

Има два конструктора – единият е празен, а другият приема като параметър “RepairRequestCreateViewModel”, за да се преобразува от ViewModel до DBM (DataBaseModel).

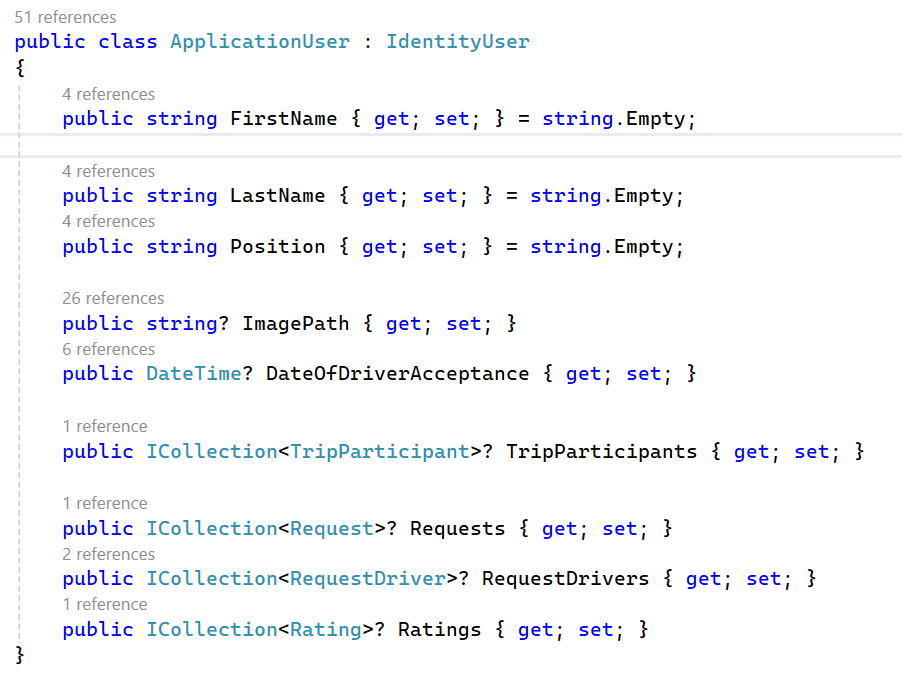
Полетата “ClientUserId”, “ClientUser” и „WorkerUserId“, “WorkerUser” осигуряват двойната връзка (обяснена в точка 2.1.1) между таблиците „User“ – “RepairRequest”.

Полетата „PhoneModelId“, “PhoneModel” осъществяват връзката между таблиците „PhoneModel“ – “RepairRequest” като връзката е едно към много.

* + - 1. **User**

Всеки регистриран потребител има:

* Потребителско име
* Парола
* E-mail
* Телефонен номер
* Собствено име (FirstName)
* Фамилно име (LastName)
* Роля



Фигура 10 – Моделът на User.

Полетата „ImagePath“ (запазва път към снимка) и „DateOfDriverAcceptance“ (запазва датата на ставане на шофьор) са полета единствено за потребители, които са промотирани към роля „Driver“ – Шофьор. Полето „Position“ е използвано единствено с цел мое улеснение при изграждане на програмата.

Полетата “TripParticipants”, „Requests“, „RequestDrivers“ и „Ratings“ осигуряват връзките много към много и една към много (обяснени в точка 2.1.1) между таблиците.

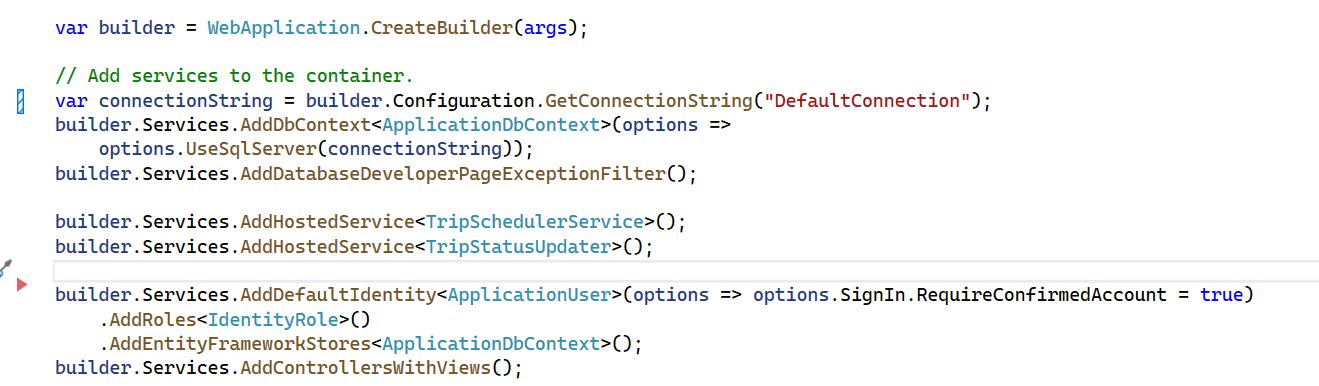
В моделът показан на Фиг. 10 виждаме, че има по-малко полета от изисканите по условие. Това е така, защото останалите са автоматично генерирани при наследяването на класа „IdentityUser“.

* + - 1. **Enum tables**

В базата данни ръчно са направени таблици с данни за всички “Enums” ползвани в проекта. Това осигурява справката на данните, погледнати само през базата.

### Потребителски интерфейс и бизнес логика

След изграждането на моделите за базата данни, трябва да се направи връзка между кода и базата. Това се постига чрез “ConnectionString” намиращ се в “appsettings.json” файла.



Фигура 11 – Осъществяване на връзка с базата данни

Редове 17-18 – Тук се конфигурира идентичността на потребителите в приложението. Включва настройките за потребителите (User), ролите (IdentityRole) и връзката с базата данни чрез „ApplicationDbContext“.

За инициализация на базата данни се използват команди предоставени от EntityFramework Core. Те се пишат в Package Manager Console. Използваните команди:

* add-migration [name] – добавя миграция (реализация на базата данни спрямо кода в приложението).
* remove-migration – изтрива последно добавената миграция.
* update-database – обновява базата данни спрямо последно добавената миграция, която не е извършена.

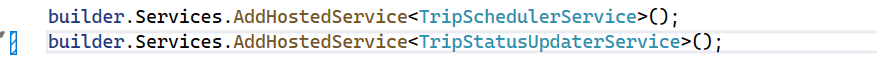
Следващите стъпки са да се изградят изгледи, контролери, „services“, модели за изгледите, които да допринесат за изпълнението на зададените задачи и реализацията на проекта.

* + 1. **Role Seeder**

По условие сайтът трябва да има един администратор, който да е първоначално зададен. За да се постигне това е направен Role Seeder класът. В него има метод, който да снабди базата с нужните данни. Този метод се вика в Program.cs, който е отговорен за стартирането на проекта. Ако в базата няма никакви записи, се създават всички роли (Admin, User, Worker) и се добавя по един потребител от всеки вид.

* + 1. **Services**

В проекта са използвани Услуги (Services). Те са отговорни за извършването на конкретни задачи в приложението като обработка на данни, управление на бизнес логика, връзка с външни услуги или взаимодействие с бази данни. Услугите се използват за разделяне на функционалността на приложението на по-малки, управляеми единици, което прави кода по-организиран. В тези Услуги се съдържат CRUD (Create, Read, Update, Delete) операциите на всеки модел.



Фигура 12 – Регистриране на зависимостите в Услугите.

Scoped услугите в ASP.NET Core са услуги, които се инстанцират в рамките на един и същ контекст на заявка (HTTP request context). Това означава, че за всеки HTTP request се създава нова инстанция на scoped услугата, която е достъпна в рамките на целия жизнен цикъл на този request.

Когато един компонент възложи scoped услуга чрез внедряване на зависимости (dependency injection), тази услуга ще бъде инстанцирана точно веднъж за всеки HTTP request. Това гарантира, че всеки потребител получава своята инстанция на услугата и че данните между различните потребители се изолират (фиг. 12).

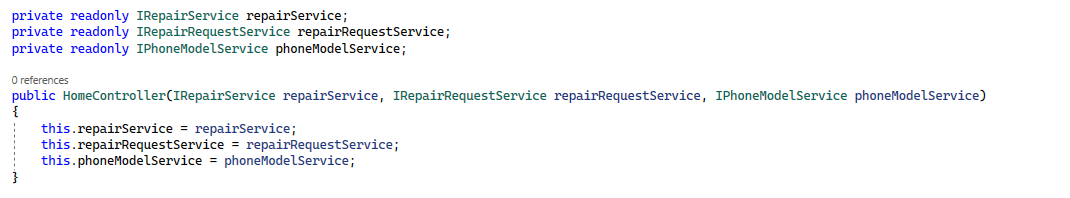
Scoped услугите са особено полезни, когато е нужно да се споделят данни между различните компоненти в рамките на един и същ HTTP request, но не и между различните заявки. Те се използват широко за инстанциране на компоненти, които управляват данни за текущия потребител, като например потребителската сесия или за достъп до базата данни в рамките на един и същ HTTP request.

* + 1. **Контролери**

В проекта има 5 контролера, които заедно с Услугите, отговарят за бизнес логиката на приложението. През тях минават всички заявки от и към базата данни, позволяват преминаването през различни изгледи. Достъпът до методите на контролерите се определя от authorization attribute (атрибут за авторизация). Всички CRUD операции свързани с базата данни са постигнати посредством Услуги за съответния модел. Повечето преобразувания от DBO до View Model се извършват чрез LINQ.

* HomeController

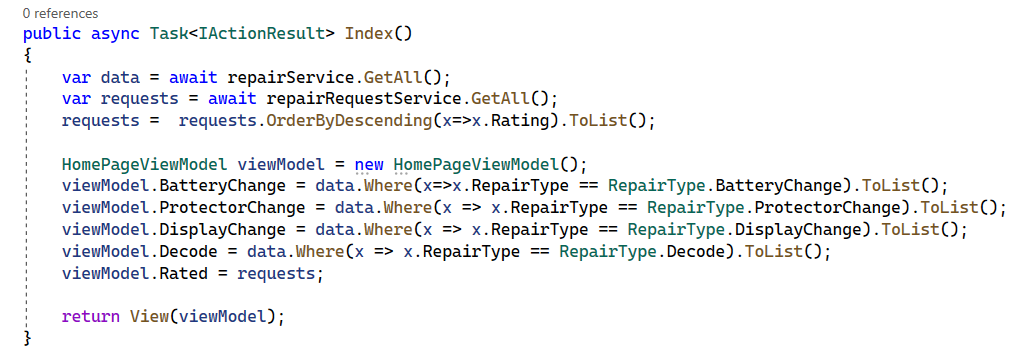
Конструкторът (фиг. 13) създава Услугите, които ще се използват в методите.



Фигура 13 – HomeController конструктор.

Index:

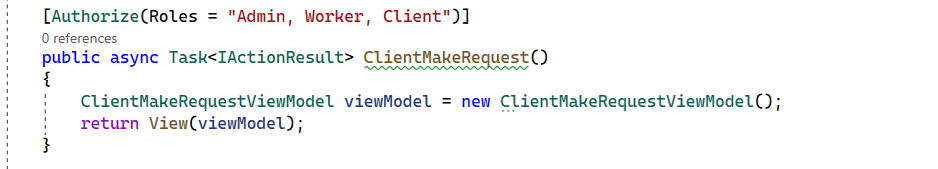
* Да вземе всички поправки, заявки за поправка, да ги подреди по рейтинг.
* Да създаде View Model и да му даде нужните данни.
* Да препрати към съответния изглед (Начало) като праща View Model.



Фигура 14 – Метод Index.

ClientMakeRequest:

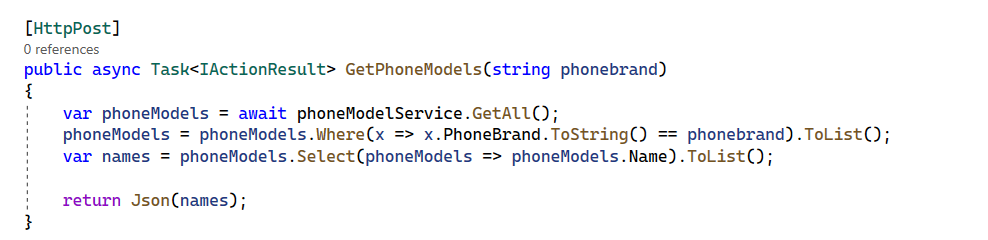
* Тук достъп имат само регистрирани потребители
* Да създаде View Model и да го прати към съответния изглед.



Фигура 15 – Метод ClientMakeRequest.

GetPhoneModels:

* Взима всички модели телефони и ги филтрира по подадения критерии.
* Връща данните в Json формат.



Фигура 16 – Метод GetPhoneModels.

GetRepairTypes:

* Да вземе всички модели телефони спрямо определено име.
* Да вземе всички поправки.
* Да подбере данните по даден критерии и да ги върне като Json.



Фигура 17 – Метод GetRepairTypes.

Privacy:

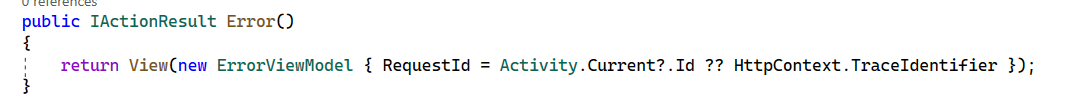
* Да върне съответния изглед.



Фигура 18 – Метод Privacy.

Error:

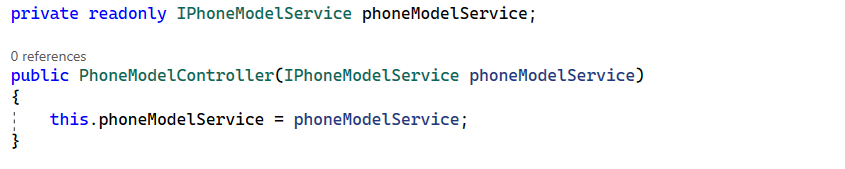
* Да върне изглед съответната грешка.



Фигура 19 – Метод Error.

* PhoneModelController

Конструкторът създава Услугите, които ще се използват в методите.



Фигура 20 – PhoneModelController конструктор.

Index:

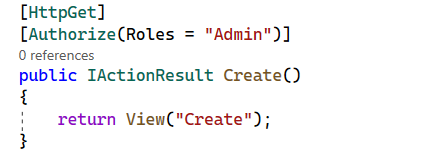
* Да вземе всички модели телефони.
* Да направи View Model и да го снабди с нужната информация.
* Да прати към съответния изглед като подава View Model като параметър.



Фигура 21 – Метод Index.

Create (get):

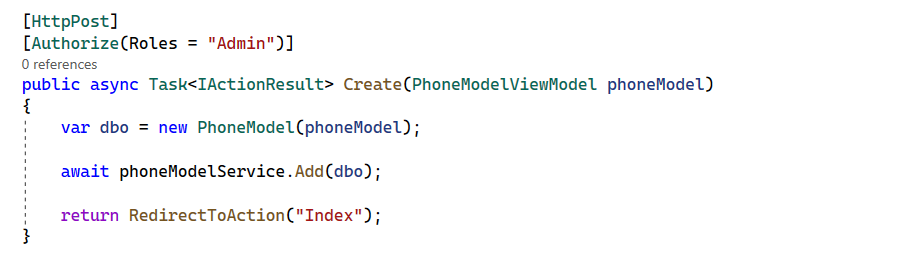
* Той е достъпен само за администратора и препраща към съответния изглед.



Фигура 22 – Метод Create (get).

Create (post):

* Той е достъпен само за администратора.
* Приема входни данни (View Model).
* Преобразува данните до DBO.
* Създава запис в базата данни.
* Препраща към Индекс метода.



Фигура 23 – Метод Create (post).

Edit (get):

* Той е достъпен само за администратора.
* Получава параметър (id) и по него намира съответния запис в базата данни.
* Ако съществува такъв запис, го връща преобразуван като View Model, ако не - връща изглед с грешка.



Фигура 24 – Метод Edit (get).

Edit (post):

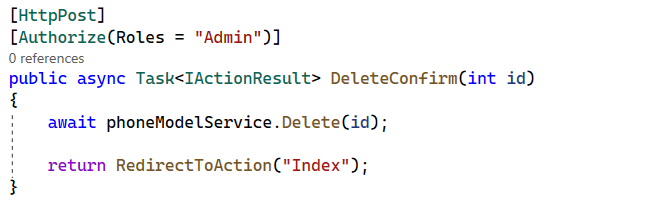
* Той е достъпен само за администратора.
* Получава View Model и спрямо този модел обновява модела в базата данни, съответстващ на него.
* Препраща към Индекс метода.



Фигура 25 – Метод Edit (post).

DeleteConfirm (post):

* Получава параметър (id) и се изтрива съответния запис в базата данни.
* Препраща към Индекс метода.



Фигура 27 – Метод DeleteConfirm (post).

* RepairController

Конструкторът създава Услугите, които ще се използват в методите.

Контролерът съдържа същите CRUD операции като PhoneModelController.

* RepairRequestController

Конструкторът създава Услугите, които ще се използват в методите.

Контролерът съдържа същите CRUD операции като PhoneModelController.



Фигура 28 – RepairRequestController конструктор.

Index:

* Той е достъпен само за администратора и служителите.
* Взема всички заявки за ремонти и ги преобразува във View Model.
* Праща към съответния изглед, подава View Model като параметър.



Фигура 29 – Метод Index.

Search:

* Това е методът, който обслужва търсачката в изгледа *Заявки.*
* Той е достъпен само за администратора и служителите.
* Възможно е да получи параметри по които да се извърши търсенето.
* Вземат се всички заявки за поправка и се филтрират по тези параметри.
* Филтрираните данни се преобразуват в View Model и се връщат на изгледа.



Фигура 30 – Метод Search.

SortByDate:

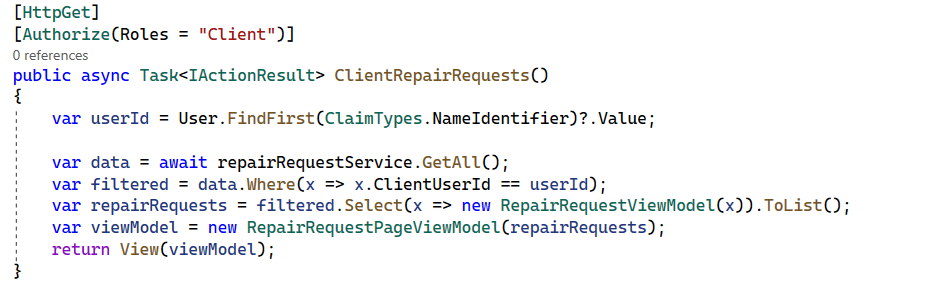
* Този метод обслужва бутонът „Дата“, намиращ се в изгледа *Заявки.*
* Той е достъпен само за администратора и служителите.
* Възможно е да получи параметри.
* Този метод съдържа в себе си метода Search.
* Сортира всички заявки за поправка по дата.
* Преобразува сортираните данни в View Model и ги връща на изгледа.



Фигура 31 – Метод SortByDate

ClientRepairRequests:

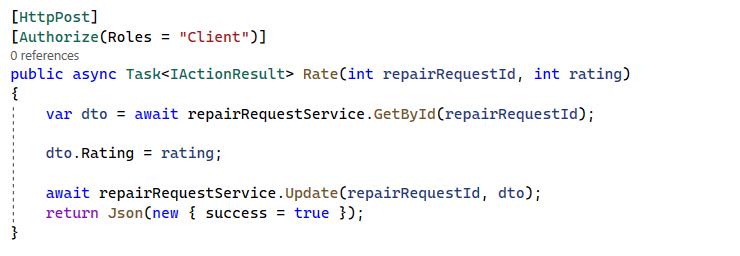
* Той е достъпен само за клиенти.
* Взима id на клиентa, който е логнат в момента.
* Взима всички заявки за поправка. Филтрира ги по взетото id. Преобразува ги във View Model и праща към съответния изглед.



Фигура 32 – Метод ClientRepairRequests.

Rate:

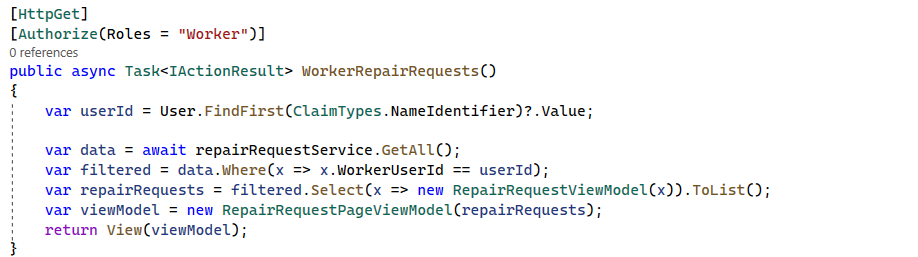
* Той е достъпен само за клиенти.
* Получава два параметъра (id, рейтинг).
* Спрямо полученото id намира отговарящата заявка на него.
* Обновява рейтинга на заявката.
* Връща Json с данни за успех.



Фигура 33 – Метод Rate.

WorkerRepairRequests:

* Този метод работи по същия начин както и ClientRepairRequests, само че за служители.



Фигура 32 – Метод WorkerRepairRequests.

GetRepairTypes:

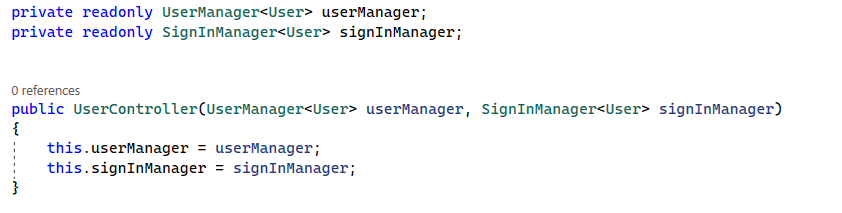
* Получава параметър View Model.
* Взимат се всички заявки за поправка и се филтрират спрямо определено id от получения View Model.
* Връща данните на изгледа в Json формат.



Фигура 33 – Метод GetRepairTypes.

* UserController

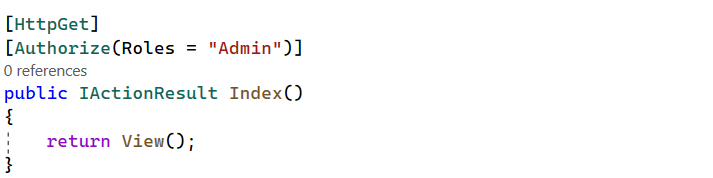
Конструкторът създава Услугите, които ще се използват в методите.



Фигура 34 – UserController конструктор.

Index (get):

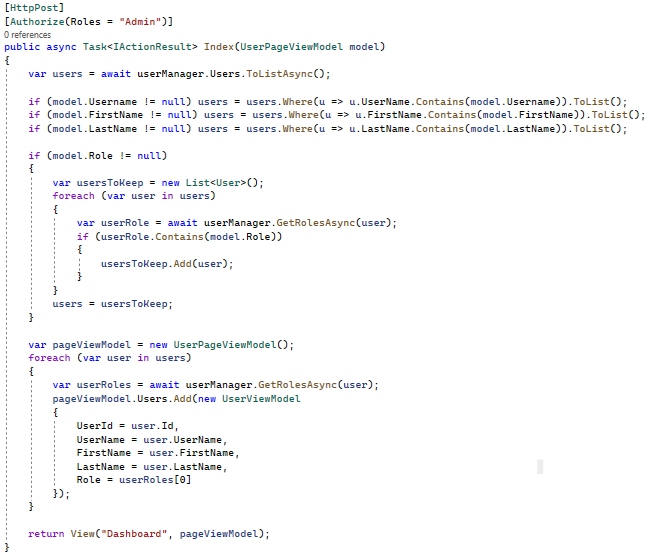
* Той е достъпен само за администратора.
* Връща към съответния изглед.



Фигура 35 – Метод Index (get).

Index (post):

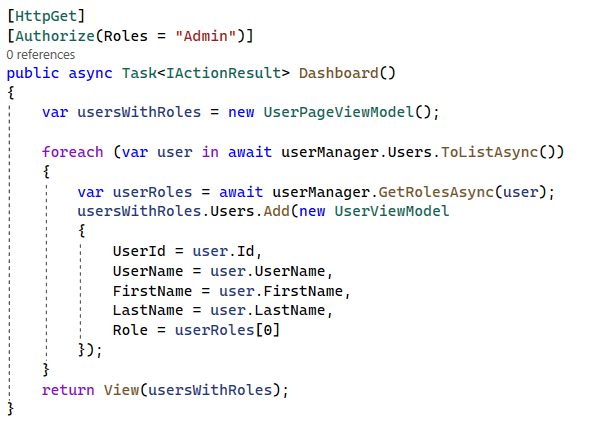
* Този метод е достъпен само за администратора.
* Той е предназначен за търсачката в административния панел.
* Приема View Model и по неговите данни филтрира всички потребители и ги връща към изгледа.



Фигура 36 – Метод Index (post).

Dashboard:

* Този метод е достъпен само за администратора.
* Взема всички потребители и ги връща на изгледа.



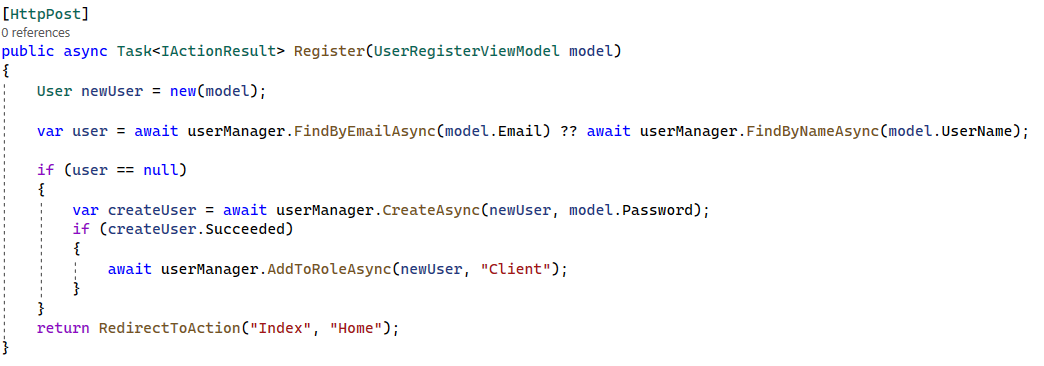
Фигура 37 – Метод Dashboard.

Register (get):

* Праща към съответния изглед.

Register (post):

* Получава View Model и го преобразува до DBO.
* Проверява дали съществува потребител с такъв e-mail.
* Ако не съществува такъв потребител го създава и препраща към Index метода в Home контролера.



Фигура 38 – Метод Register (post).

Login (get):

* Праща към съответния изглед.

Login (post):

* Опитва да влезе в профила на потребителя.
* Ако е успешно го логва и праща на началната страница, ако не – показва грешките.



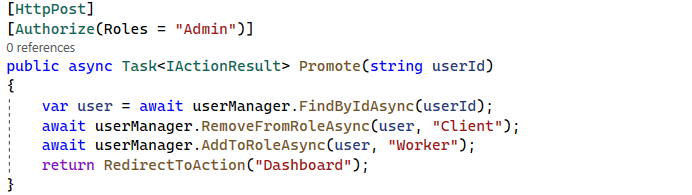
Фигура 39 – Метод Login (post).

Logout:

* Излиза от акаунта на потребителя и го праща на началната страница.

Promote:

* Повишава потребителя със съответното id.



Фигура 40 – Метод Promote.

Demote:

* Понижава потребителя със съответното id.



Фигура 41 – Метод Demote.

* + 1. **Enums**

В папката Enums се намират всички данни, които са били зададени в условието и не могат да се променят при използване на приложението.

* PhoneBrand – записани са 9 популярни марки телефони
* RepairRequestStatus:
* PendingConfirmation (изчакваща одобрение)
* OngoingRepair (в процес на поправка)
* Completed (завършена)
* RepairRequestType:
* Standard (14 дни)
* Fast (7 дни)
* Express (3дни)
* RepairType:
* DisplayChange (смяна на дисплей)
* BatteryChange (смяна на батерия)
* Decode (декодиране)
* ProtectorChange (смяна на протектор)
  + 1. **Модели за изгледи**

View моделите помагат за по-добър подбор на данните, които се предоставят на всеки изглед. Те също така допринасят за защита на данните като не позволяват директният достъп до моделите в базата данни. Използват се като модели в съответния изглед. Те са полезен инструмент при по-сложни функционалности на програмата.

В проекта има 5 папки с модели за изгледите:

* HomeModels:
* ClientMakeRequestViewModel
* HomePageViewModel
* PhoneModelModels:
* PhoneModelPageViewModel
* PhoneModelViewModel
* RepairModels
* RepairCreateViewModel
* RepairPageViewModel
* RepairViewModel
* RepairRequestModels
* RepairRequestCreateViewModel
* RepairRequestPageViewModel
* RepairRequestViewModel
* UserModels
* UserLoginViewModel
* UserPageViewModel
* UserRegisterViewModel
* UserViewModel
  + 1. **Изгледи**

Изгледите са частта, която потребителя вижда. За техният дизайн е използвано предимно Bootstrap 5. За определени случаи е написан и CSS код. За икони е използван FontAwesome.

* + - 1. **Home**

*Направи заявка*

Тук клиентът може да види нагледно през какъв процес минава правенето на една заявка. Този панел му позволява да провери колко скъпо ще му излезе поправката на телефона. Това става чрез следването на 4 стъпки (Избирането на телефон, ремонт, тип ремонт, получаване на информация за поръчката). При натискането на бутоните, избирането на опции в падащите менюта, селектирането на радио бутони, страницата се обновява динамично и няма презарежда при всяко кликване. Това е постигнато чрез JavaScript, jQuery, AJAX.

*Начало*

Началната страница се разделя на 5 секции (Приветствия, Предлагани услуги, Покана за регистрация, За нас, Оценки на клиентите). Страницата се вижда от всички. Посетителите виждат бутон, в секцията която ги кани да се регистрират, той ги води към *Регистрация.*

* + - 1. **PhoneModel**

*Модели телефони*

Клиентите и служителите виждат таблица с всички телефони, за които могат да се правят ремонти. Администраторите разполагат с бутони за създаване, редактиране и изтриване на такива.

*Добавяне на модел на телефон*

Изглед предназначен за създаването на модел на телефон. Има полета за попълване на информация и бутон за добавяне.

*Редактиране на модел на телефон*

Същият изглед като *Добавяне на модел на телефон,* с разликата, че е предназначен за редакция.

* + - 1. **Repair**

*Ремонти*

Клиентите и служителите виждат таблица с всички ремонти. Администраторите разполагат с бутони за създаване, редактиране и изтриване на такива.

*Добавяне на нов ремонт*

Изглед предназначен за създаването на ремонт. Има полета за попълване на информация и бутон за добавяне.

*Редактиране на ремонт*

Същият изглед като *Добавяне на нов ремонт,* с разликата, че е предназначен за редакция.

* + - 1. **RepairRequest**

*Заявки*

Този изглед е достъпен само за служителите и администратора. Всички заявки за ремонти са представени в таблица. Над нея има меню за търсене. Администраторът може да редактира и изтрива всички заявки, докато служителите – само тези, които са възложени на тях. И двата вида потребители могат да създават нови заявки.

*Добавяне на нова заявка*

Изглед предназначен за създаването на нова заявка. Има полета за попълване на информация и бутон за добавяне.

*Редактиране на заявка*

Същият изглед като *Добавяне на нова заявка,* с разликата, че е предназначен за редакция.

*Клиентски заявки /Моите заявки/*

Тук клиентът вижда всички свои заявки представени в таблица. Ако заявката е завършена, има бутон за оценка.

*Заявки на работника /Моите заявки/*

Тук служителят вижда всички свои заявки представени в таблица. Има бутони за създаване, редакция и изтриване на заявка.

* + - 1. **Shared**

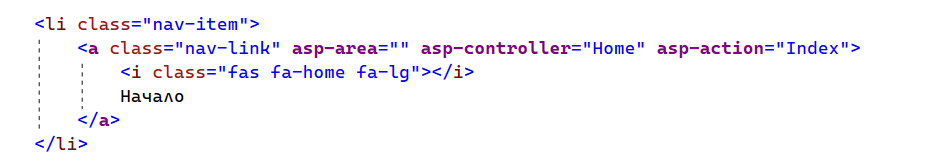
Тук се съдържат всички изгледи, които са споделени с всички останали.

„Layout“ - съдържа в себе си навигационното меню, което се вижда при всяка страница на сайта освен при регистрацията и влизането (там е изключен). Тук е направена връзката с Bootstrap:

<link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />

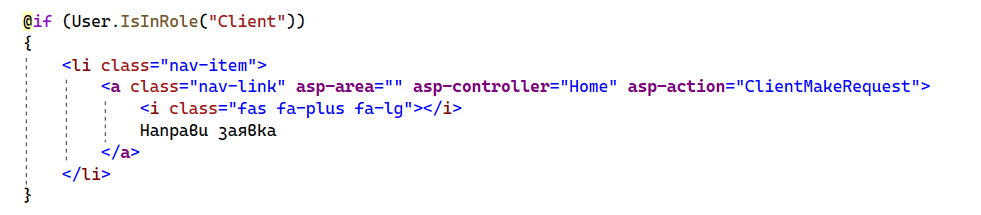
В зависимост като какъв се разглежда сайта, навигационното меню има различни панели към които може да води. Навигационното меню води към:

*Начало*

**

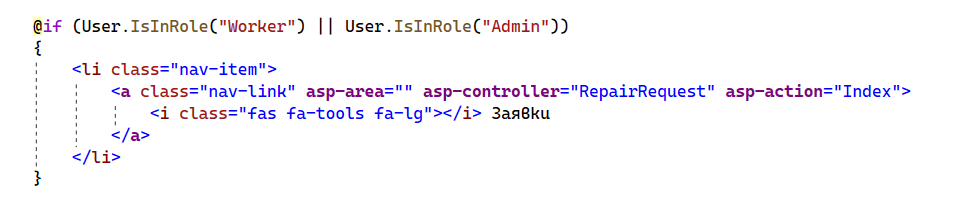
Фигура 42 – Бутон „Начало“.

*Направи заявка –* ако си клиент

**

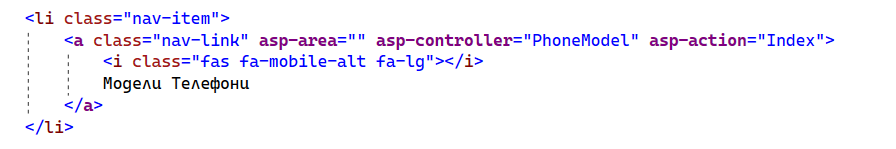
Фигура 43 – Бутон „Направи заявка“.

*Заявки –* ако си служител или администратор

**

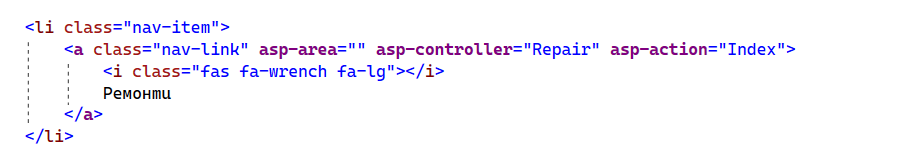
Фигура 44 – Бутон „Заявки“.

*Модели Телефони*

**

Фигура 45 – Бутон „Модели телефони“.

*Ремонти*

**

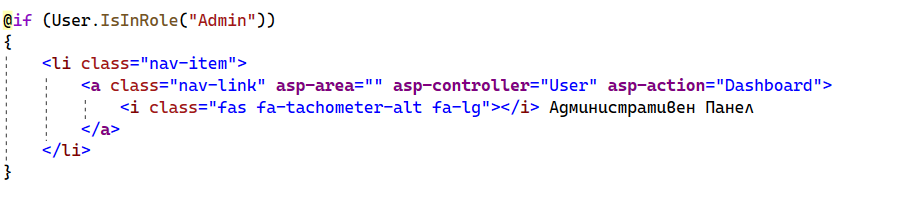
Фигура 46 – Бутон „Ремонти“.

*Моите заявки –* ако си служител или клиент

**

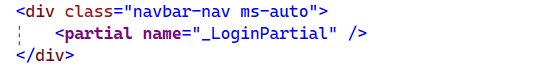
Фигура 47 – Бутон „Моите заявки“

*Административен панел –* ако си администратор

**

Фигура 48 – Бутон „Административен панел“

*Регистрация / Вход*

**

Фигура 49 – Код за генериране на изгледа с опциите за регистрация, вход.

**

Фигура 50 – Изгледът с бутоните за влизане, излизане и регистрация

* + - 1. **User**

*Регистрация*

Панел за регистриране на нов потребител със съответните полета за попълване на информация. Бутони за регистрация и вход. Има логото на сайта.

*Вход*

Панел за вход на потребител със съответните полета за попълване. Бутони за вход и регистрация. Има логото на сайта.

*Административен панел*

Панел показващ на администратора всички потребители на сайта в таблица. Има търсачка. Клиентите могат да бъдат повишени до служители, а служителите – понижени до клиенти.

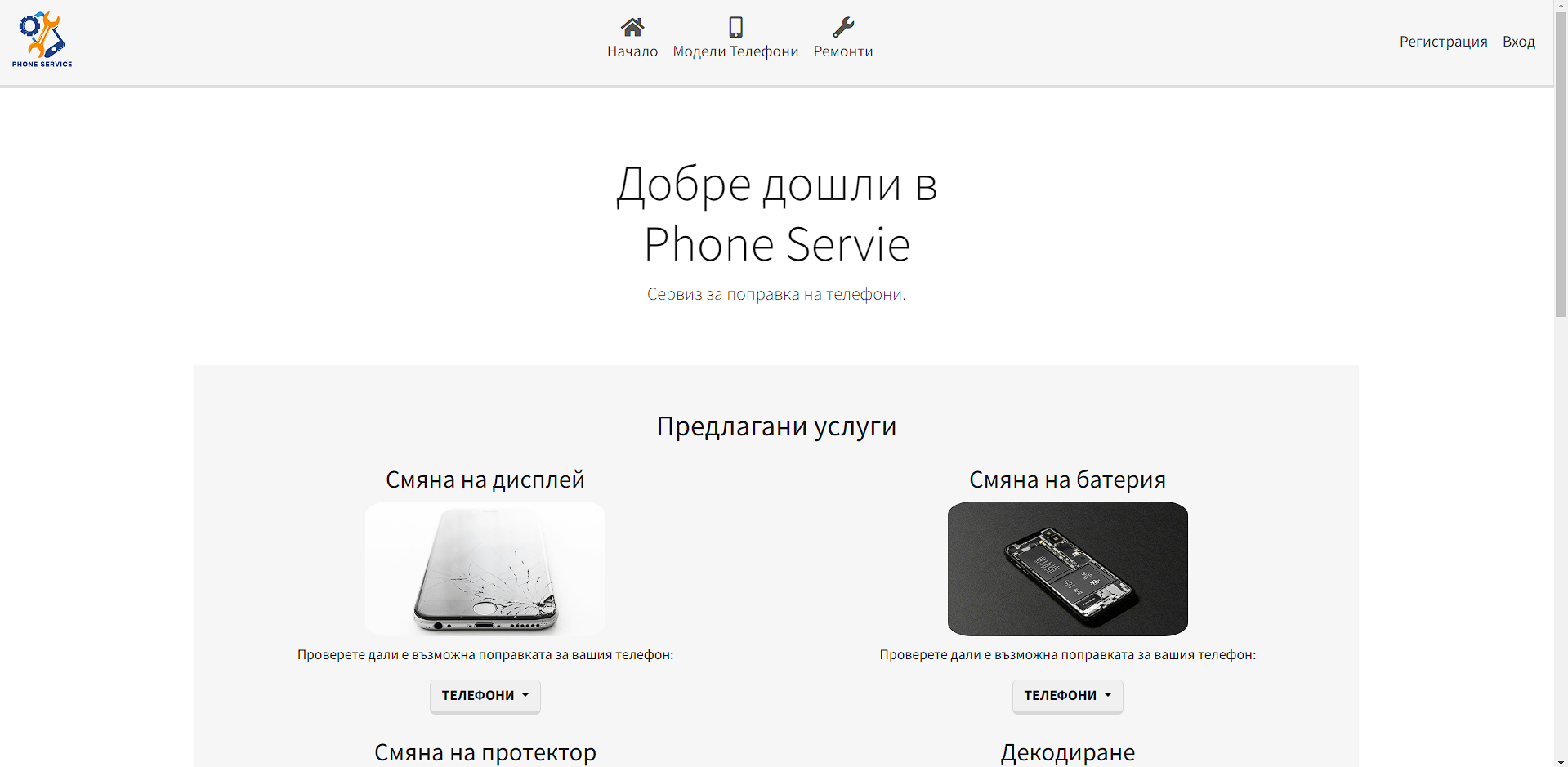
# Заключение

Разработено е уеб-приложение за сервиз за телефони. Архитектурата му е изградена на база на MVC модела. За реализиране на уеб приложението са използвани средства като HTML, CSS, JavaScript, C#, EntityFramework Core, ASP.NET Core, Microsoft SQL Server, Bootstrap, FontAwesome. Разработената система е гъвкава, сигурна и лесна за използване от всякакъв вид потребител. Това допринася за подобряване на процесите в сервиза и за увеличаване на удовлетворението на клиентите. Като предложение за доразвиване на проекта, може да се добавят повече роли, а клиентите на сайта да имат повече възможности. Един такъв проект би бил реално приложим в управлението на сервиз за телефони, тъй като ще улесни работата на служителите и достъпа до информация на клиентите.

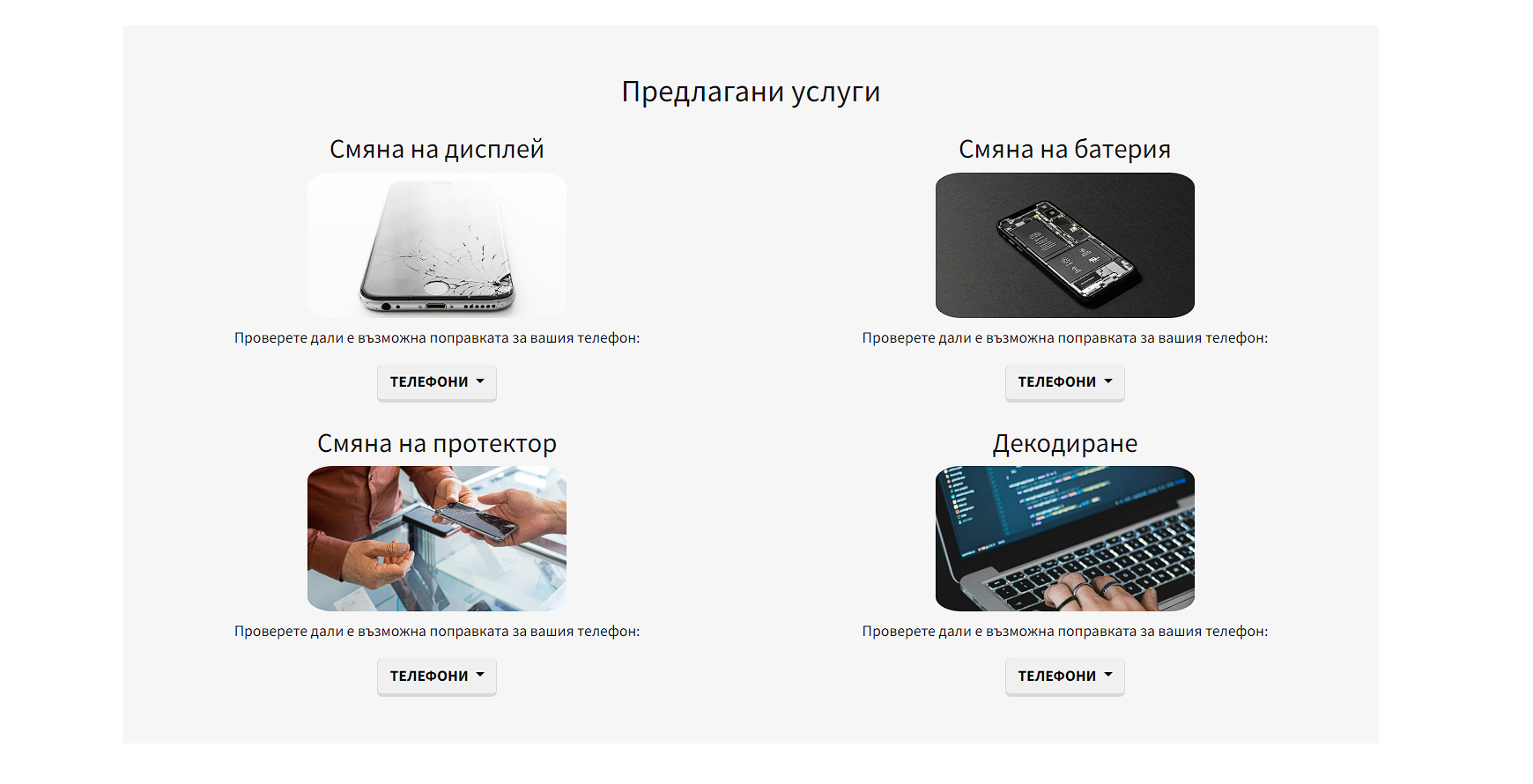
# Използвани източници

1. <https://chat.openai.com/>
2. <https://www.w3schools.com/>
3. <https://getbootstrap.com/>
4. Светлин Наков, „Принципи на програмирането със C#“
5. Joseph Albahari, “C# 10 in a Nutshell”

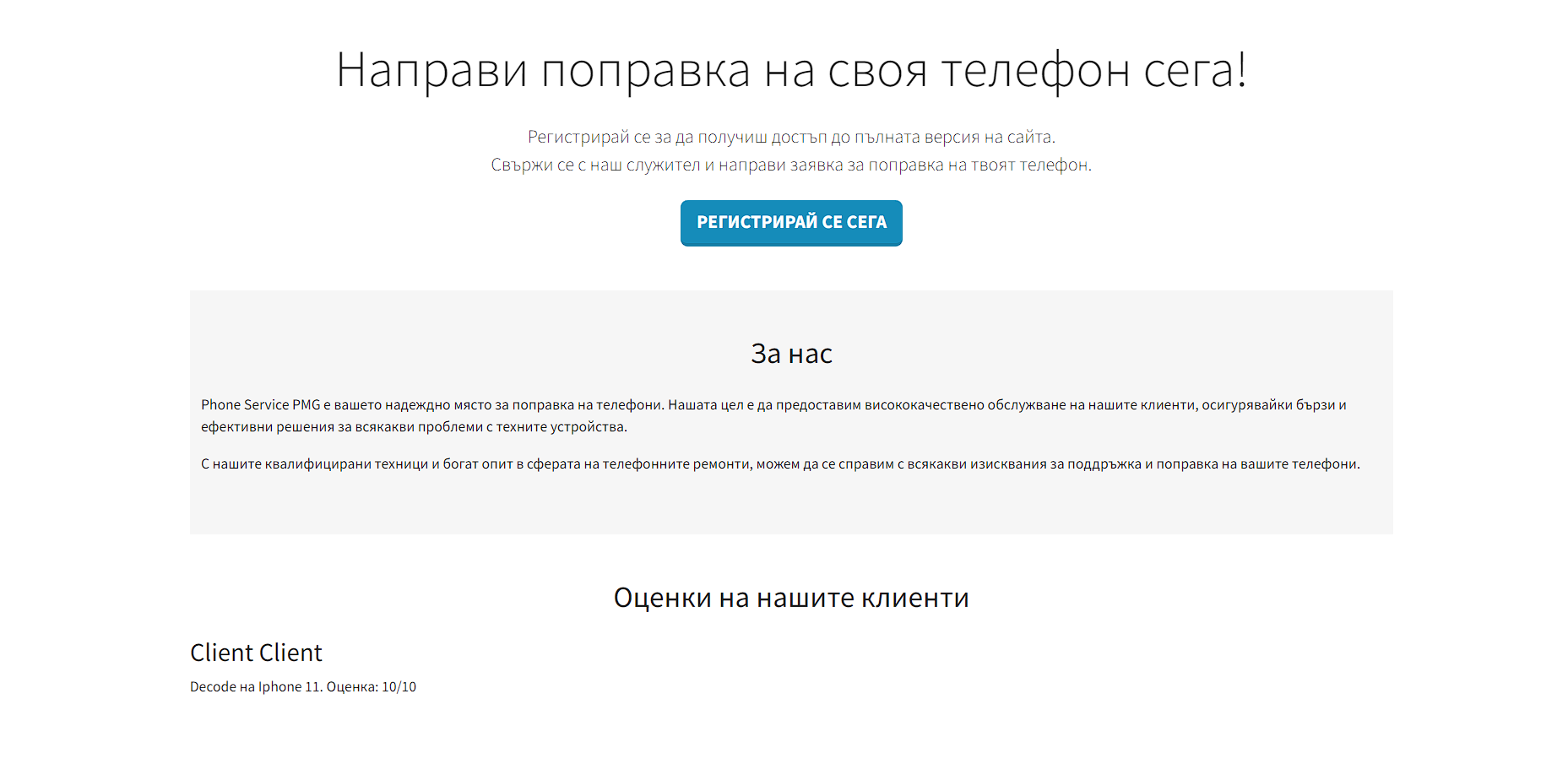
# Приложения

****

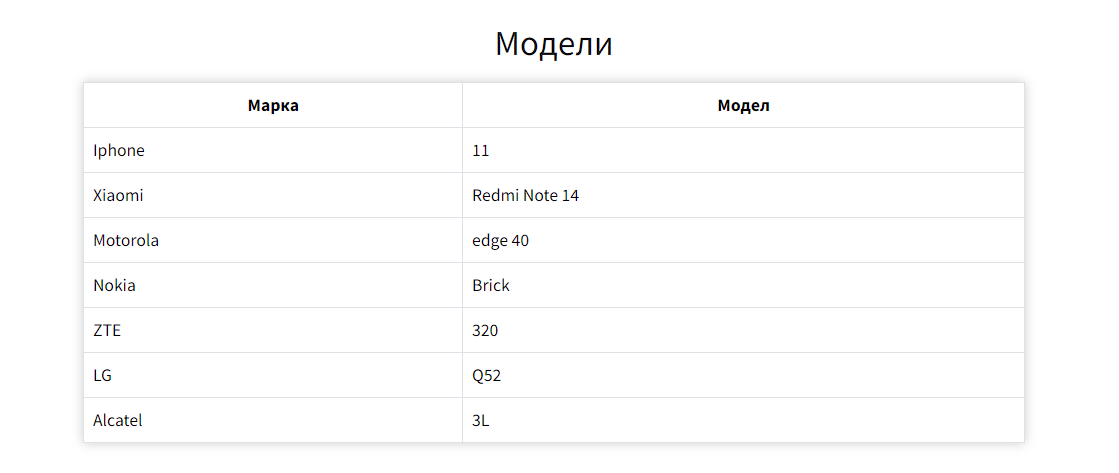
Фигура 51 – Начална страница част 1.

****

Фигура 52 – Начална страница част 2.

****

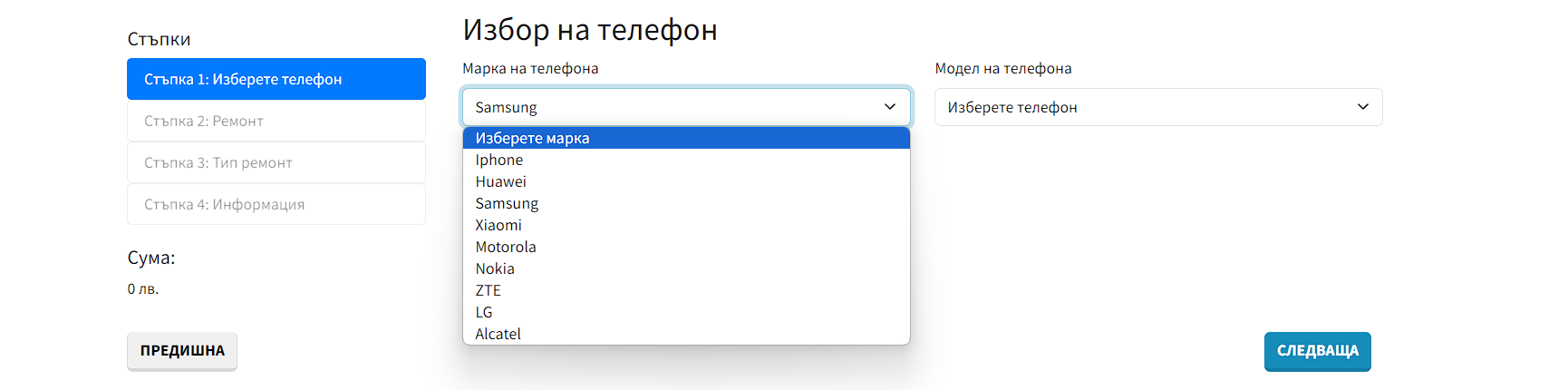
Фигура 53 – Начална страница част 3.

****

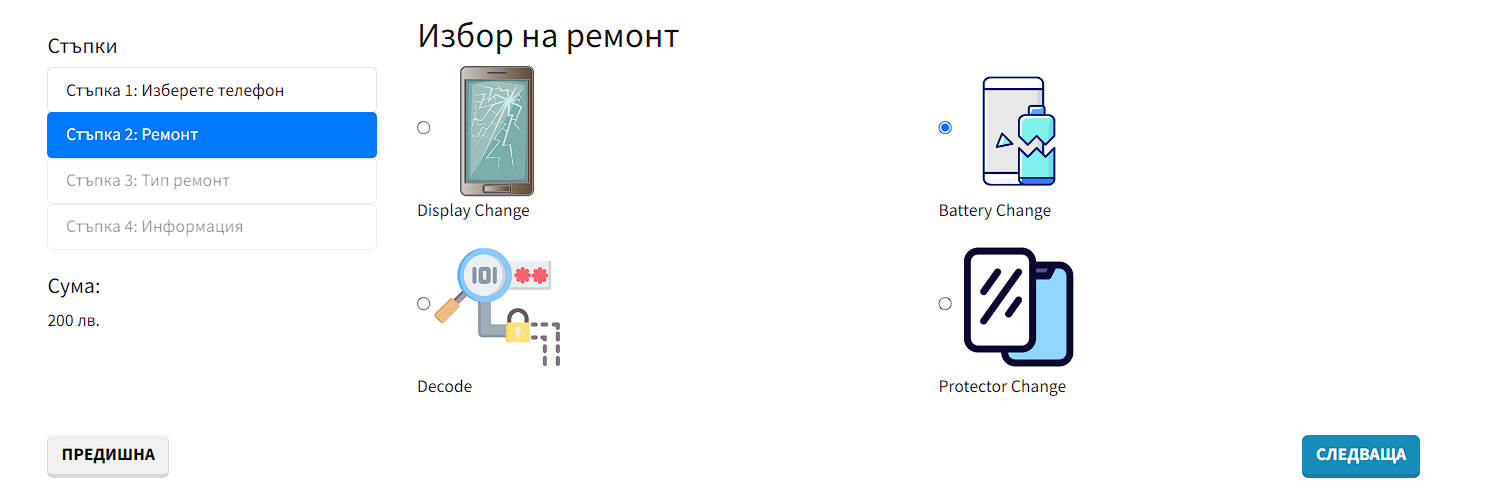
Фигура 54 – Таблица с телефони погледната през служител и потребител.

****

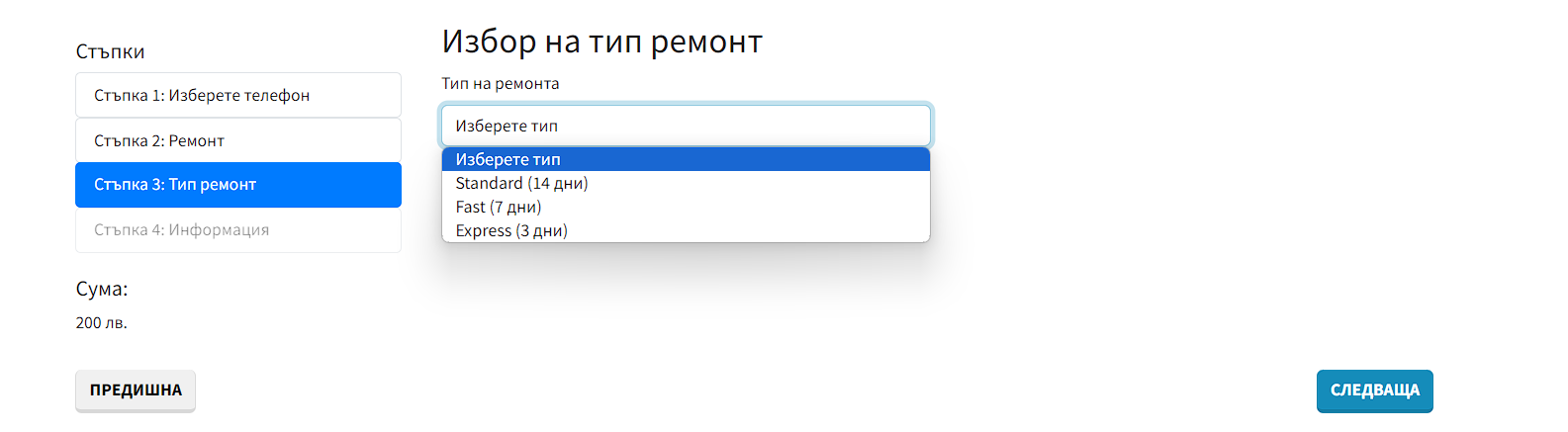
Фигура 55 – Таблица с ремонти погледната през служител и потребител.

****

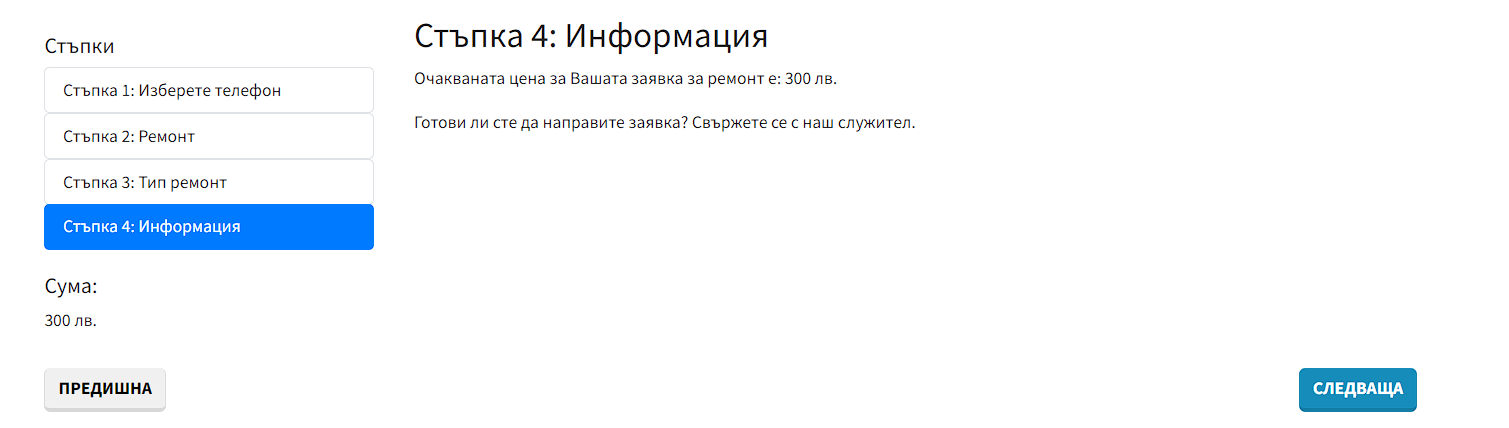
Фигура 56 – Създаване на демо заявка от потребител част 1.

****

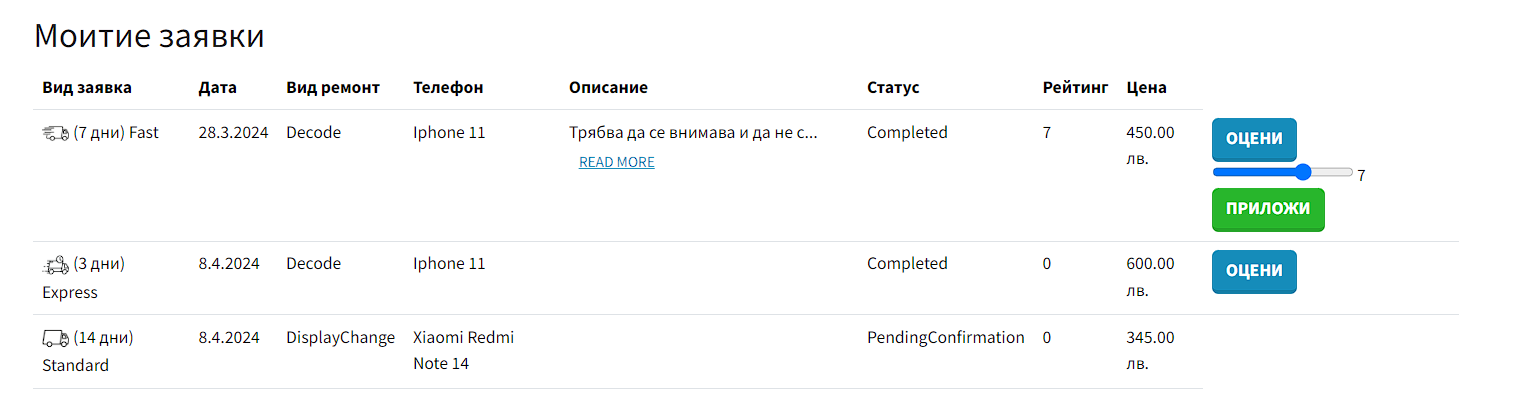
Фигура 57 – Създаване на демо заявка от потребител част 2.

****

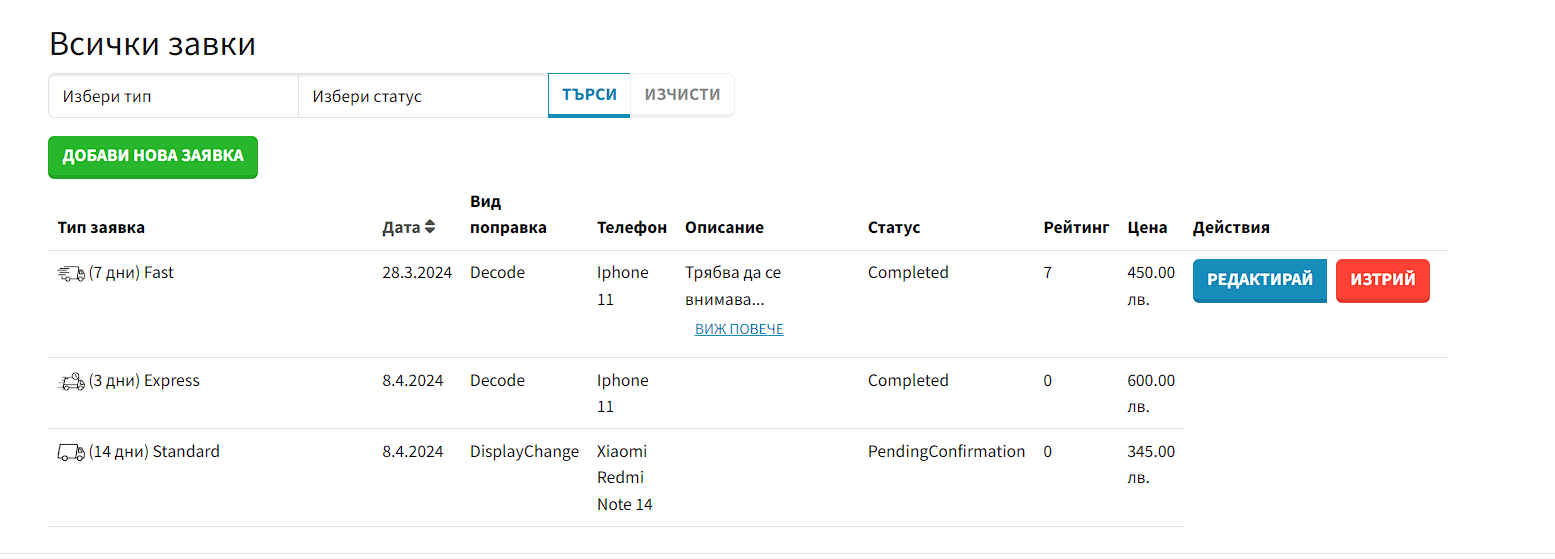
Фигура 58 – Създаване на демо заявка от потребител част 3.

****

Фигура 59 – Създаване на демо заявка от потребител част 4.

****

Фигура 60 – Таблица със заявки погледнати през потребителя.

****

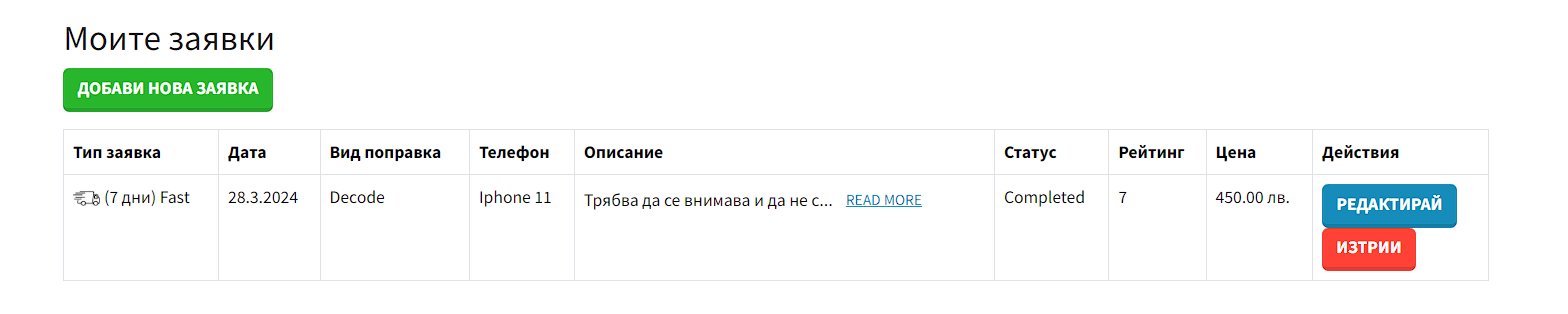
Фигура 61 – Таблица с всички заявки погледната през служител.

****

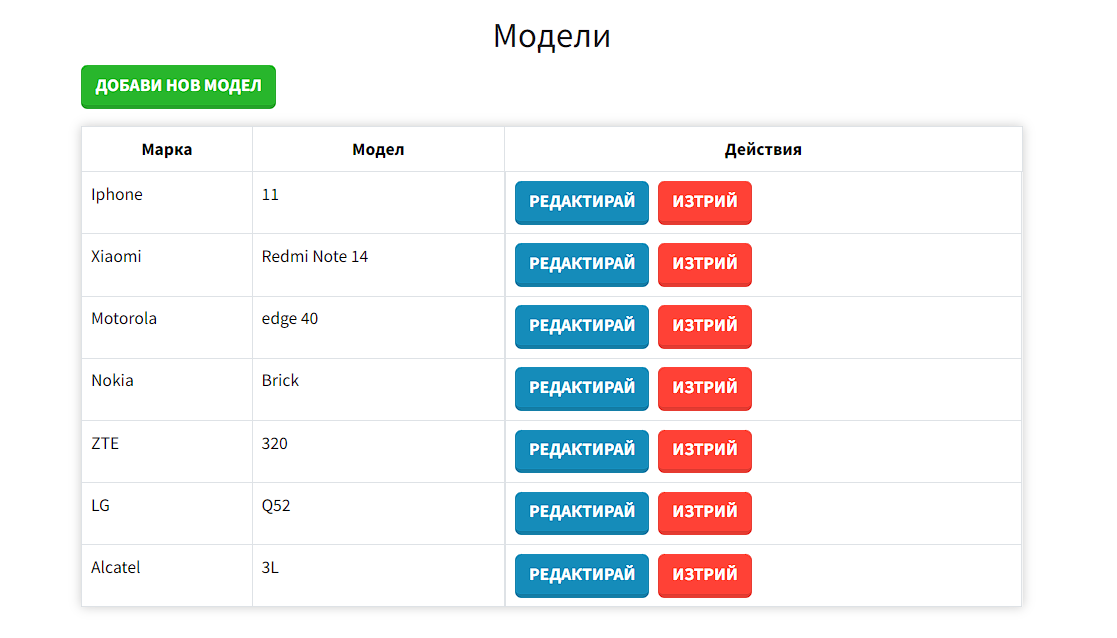
Фигура 62 – Форма за добавяне на заявка.

****

Фигура 63 – Форма за редакция на заявка.

****

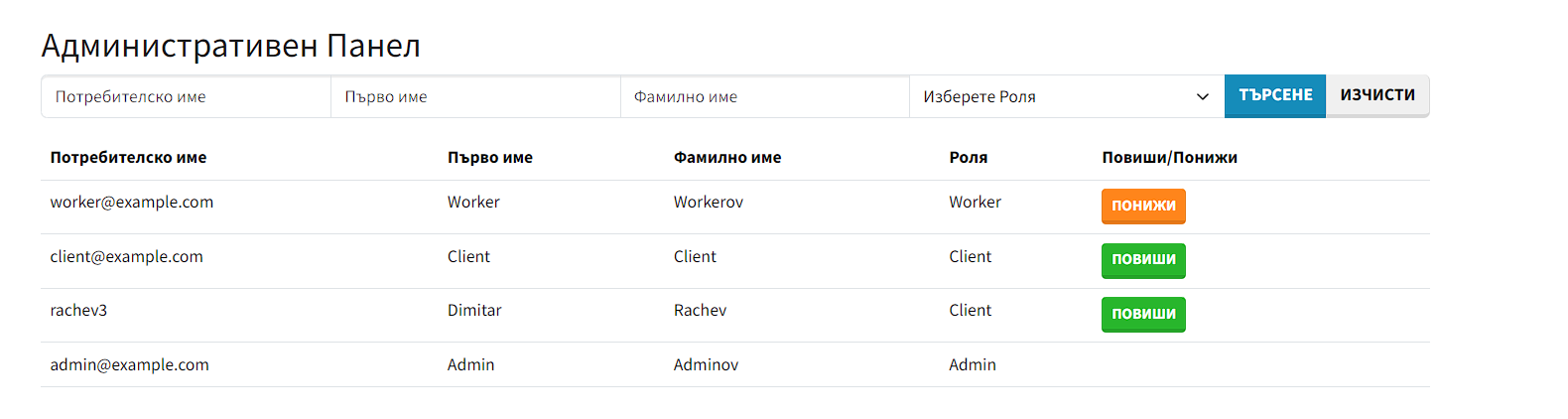
Фигура 64 – Таблица с заявките на работник.

****

Фигура 65 – Таблица с телефони погледната през администратор.

****

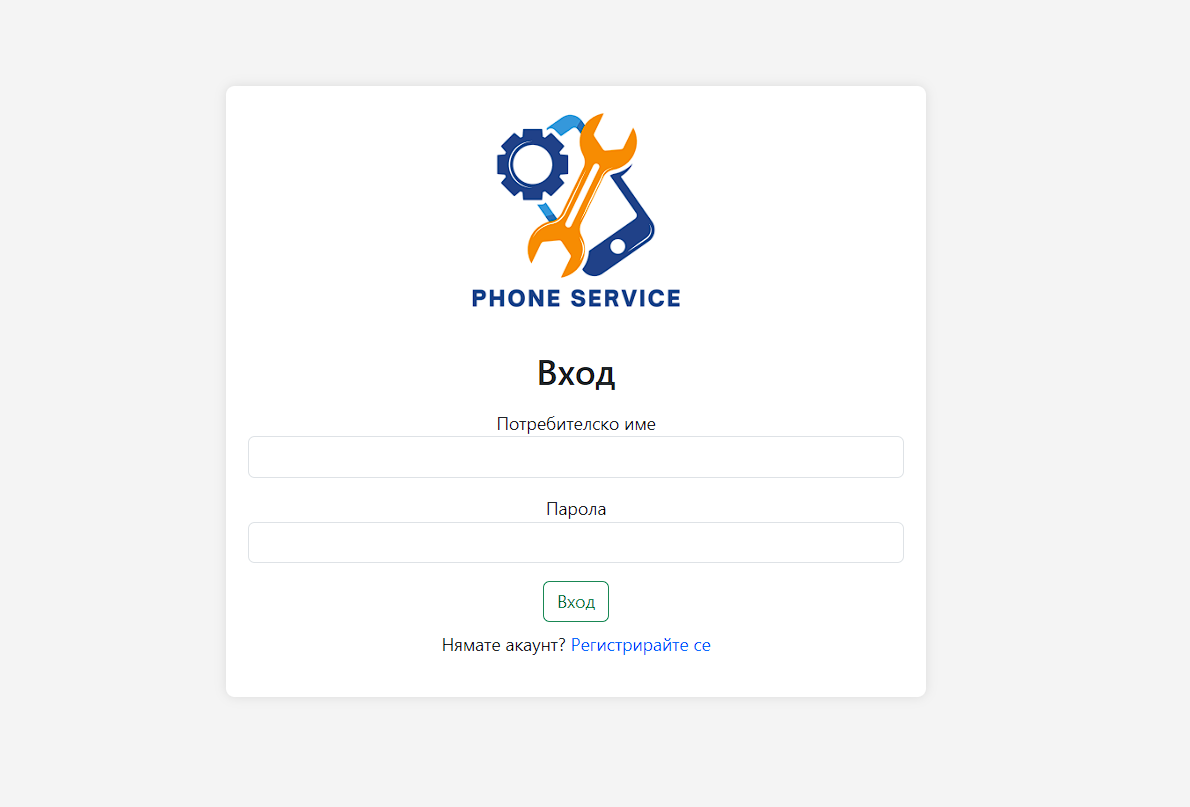
Фигура 66 – Таблица с ремонти погледната през администратор.

****

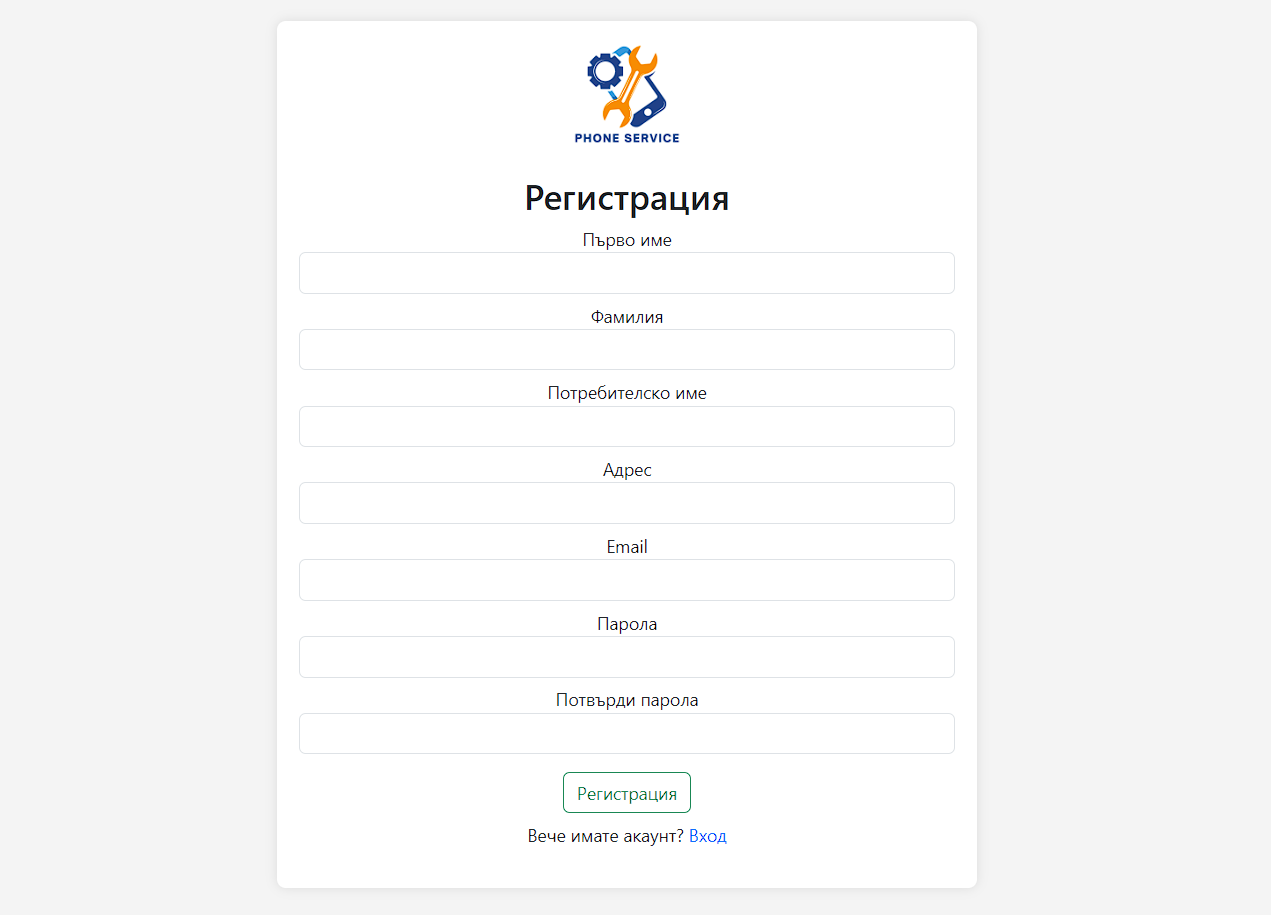
Фигура 67 – Административен панел.

****

Фигура 68 – Как изглежда валидирането.

****

Фигура 69 – Панел за вход на потребителя.

****

Фигура 70 – Панел за регистрация на потребителя.