**ДИПЛОМЕН ПРОЕКТ**

**ЗА ДЪРЖАВЕН ЗРЕЛОСТЕН ИЗПИТ**

по професия код 481030 „Приложен програмист“

специалност код 4810301 Приложно програмиране“

ТЕМА: „Електронна система за наемане на битова техника“

Автор:

Васил Любенов Младенов, клас XII В

Ръководител:

Даниела Пендашева

БургасСЪДЪРЖАНИЕ

[1 Увод 3](#_Toc163628641)

[2 Цели и обхват на софтуерното приложение 4](#_Toc163628642)

[3 Анализ на решението 5](#_Toc163628643)

[3.1 Потребителски изисквания и работен процес 5](#_Toc163628644)

[3.2 Примерен потребителски интерфейс 6](#_Toc163628645)

[3.3 Диаграми на анализа 8](#_Toc163628646)

[3.3.1 Диаграма на случаи за употреба 8](#_Toc163628647)

[3.3.2 Диаграма на дейността 9](#_Toc163628648)

[3.3.3 Диаграма на последователността 10](#_Toc163628649)

[3.4 Модел на съдържанието / данните 11](#_Toc163628650)

[3.4.1 Диаграма за отношенията между моделите 11](#_Toc163628651)

[3.4.2 Нормализация 12](#_Toc163628652)

[3.4.3 Пояснения за таблиците 17](#_Toc163628653)

[4 Дизайн 19](#_Toc163628654)

[4.1 Реализация на архитектурата на приложението 20](#_Toc163628655)

[4.2 Описание на слоевете, предназначението им, библиотеки и методи включени в съответния слой. 21](#_Toc163628656)

[4.2.1 Слой на бизнес логиката 21](#_Toc163628657)

[4.2.2 Слой на достъпа до база данни 23](#_Toc163628658)

[4.2.3 Уеб приложение 24](#_Toc163628659)

[4.3 Организация и код на заявките към база от данни 26](#_Toc163628660)

[4.3.1 Contracts 26](#_Toc163628661)

[4.3.2 IdentityModel 30](#_Toc163628662)

[4.3.3 Models 31](#_Toc163628663)

[4.3.4 Configurations 32](#_Toc163628664)

[4.4 Наличие на потребителски интерфейс (конзолен, графичен, уеб) 34](#_Toc163628665)

[5 Ефективност и бързодействие на решението 35](#_Toc163628666)

[6 Тестване 36](#_Toc163628667)

[7 Заключение и възможно бъдещо развитие 38](#_Toc163628668)

[8 Използвани литературни източници и Уеб сайтове 39](#_Toc163628669)

[9 Приложения 40](#_Toc163628670)

[10 Критерии и показатели за оценяване 42](#_Toc163628671)

# Увод

Съвременният свят на технологиите и онлайн платформите променя начина, по който извършваме множество дейности, включително наемането на битова техника. Във времена, когато удобството и ефективността са от съществено значение за потребителите, електронната система за наемане на битова техника се налага като неизбежна и напълно логична следваща стъпка в еволюцията на този процес.

Първо и най-важно, такава система предлага безсравнимо удобство за клиентите. Вместо да трябва да отделят време за посещение на магазини или провеждане на дълги телефонни разговори, клиентите могат лесно и бързо да наемат необходимата техника от уюта на своите домове или офиси. Това освобождава техните графици и ги прави по-продуктивни.

Освен удобство, електронната система за наемане предоставя и гъвкавост на резервациите. Клиентите могат да правят резервации за техника в удобно за тях време, без да бъдат ограничени от работното време на магазините. Това им дава по-голяма свобода и контрол върху процеса на наемане.

Допълнително, системата предлага повишена прозрачност и информация. Клиентите могат лесно да преглеждат наличността на различни видове техника, цените за наемане и условията за ползване. Това им дава възможността да вземат по-добре информирани решения и да избегнат нежелани изненади.

За доставчиците на услуги, електронната система предлага значителни предимства. Тя позволява по-лесно и ефективно управление на инвентара и резервациите. Те могат да проследяват наличността на техниката, да управляват резервациите и да планират доставките си по-ефективно, което в крайна сметка подобрява тяхната бизнес ефективност.

В този документ предстои описание на поставените цели, предложените решения, реализации и анализи, чрез набор от UML диаграми, тестове и разсъждения.

# Цели и обхват на софтуерното приложение

Софтуерното приложение има за цел да предостави удобно и ефективно решение за наемане на битова техника през интернет. То ще обхване всички основни етапи на процеса на наемане - от избора и резервацията на техника до плащането и получаването на продукта.

Потребителите на системата се разделят на два основни типа

* Клиенти - търсят и наемат битова техника;
* Администратори – представляват доставчиците на услуги, които предлагат техниката за наемане.

Двата типа на потребители могат да се различат и по техните дейности

* Клиентите преглеждат наличната битова техника и цените, избират желаната от тях техника, период за наемане и плащат наема през онлайн платформа;
* Доставчиците на услуги имат възможността да добавят нови продукти, да управляват наличността и цените им и да обработват и потвърждават заявките за наем.

С тези потребители и дейности, проектът си поставя поредица от цели и подцели, които да улеснят използването на приложението

* Да се реализира уеб приложение на MVC с подходящите модели за извършването на работата и пазенето на данни;
* Да се създаде сигурен бизнес слой;
* Да се създадат подходящите операции на работа с данните в моделите;
* Да се създаде система за регистрация на потребителите;
* Да се осигури интуитивен и лесен за навигация и наемане интерфейс.

Със съчетание на тези цели и подцели, софтуерното приложение ще предложи цялостно и задоволително решение за наемане на битова техника, което ще бъде полезно както за клиентите, така и за доставчиците на услуги.

# Анализ на решението

## Потребителски изисквания и работен процес

**Входни данни**

* Потребителска регистрация: Потребителите предоставят лична информация, адрес и контактни данни (Фигура 1);
* Списък с достъпната за наемане техника: Включва вид техника и период на наемане (Фигура 2).

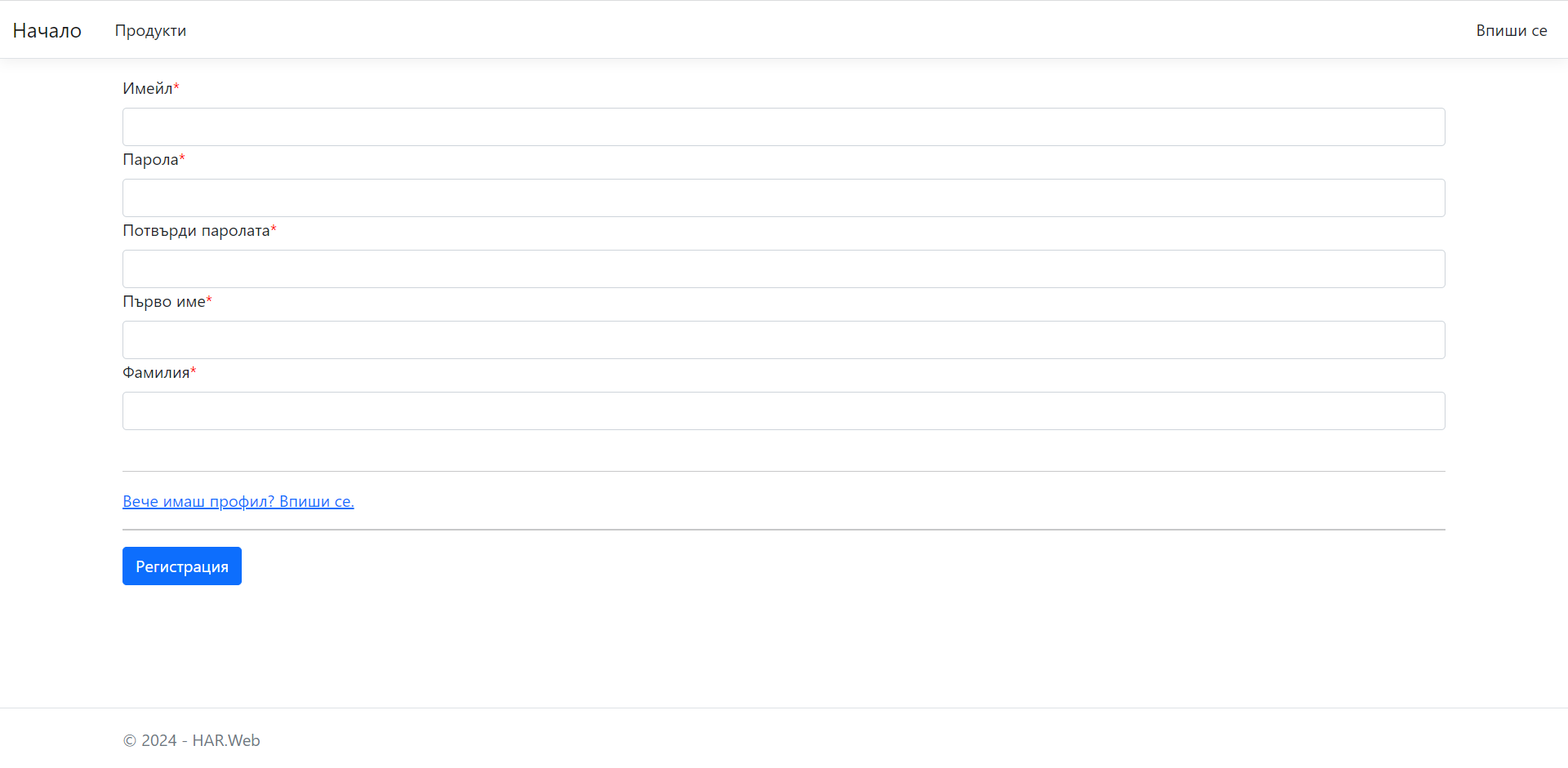
**Обработка и запазване в системата**

* Регистрация на потребителите в системата: Потребителите попълват формуляр с необходимата информация;
* Избор на техника и период на наемане: Потребителите избират желаната техника и периода на наемане;
* Проверка на наличността на избраната техника и изчисляване на цената за наемане: Системата проверява наличността на техниката за съответния период и изчислява цената за наемане;
* Потвърждение на поръчката: Потребителите потвърждават поръчката и внасят необходимите такси.

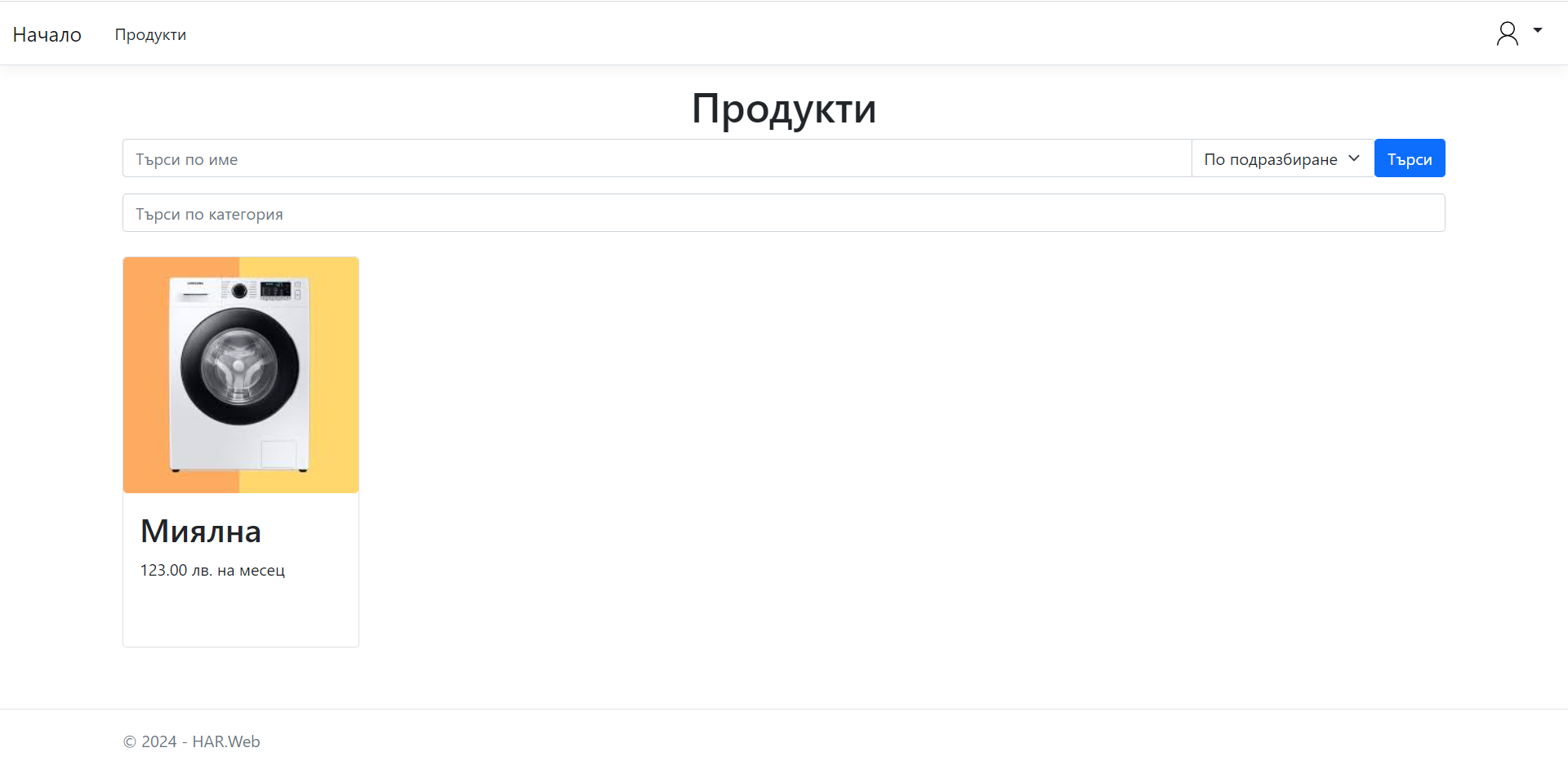
**Изходни данни**

* Страница с количката и добавените в нея продукти (Фигура 3);
* Потвърждение за успешно направената поръчка (Фигура 4);
* Подробности за наетата техника и период на наемане;
* История от предишните наеми на клиента.

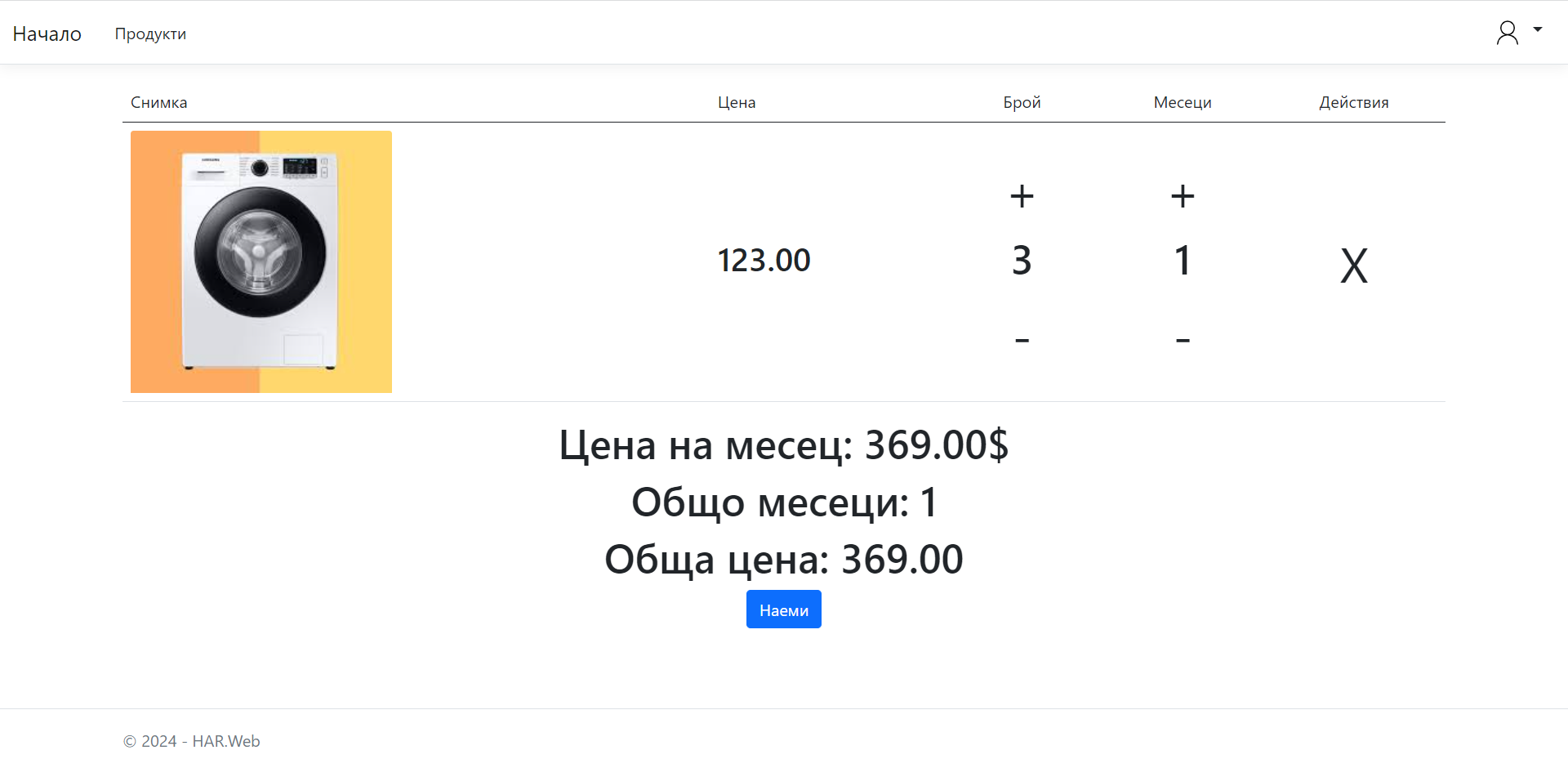
## Примерен потребителски интерфейс



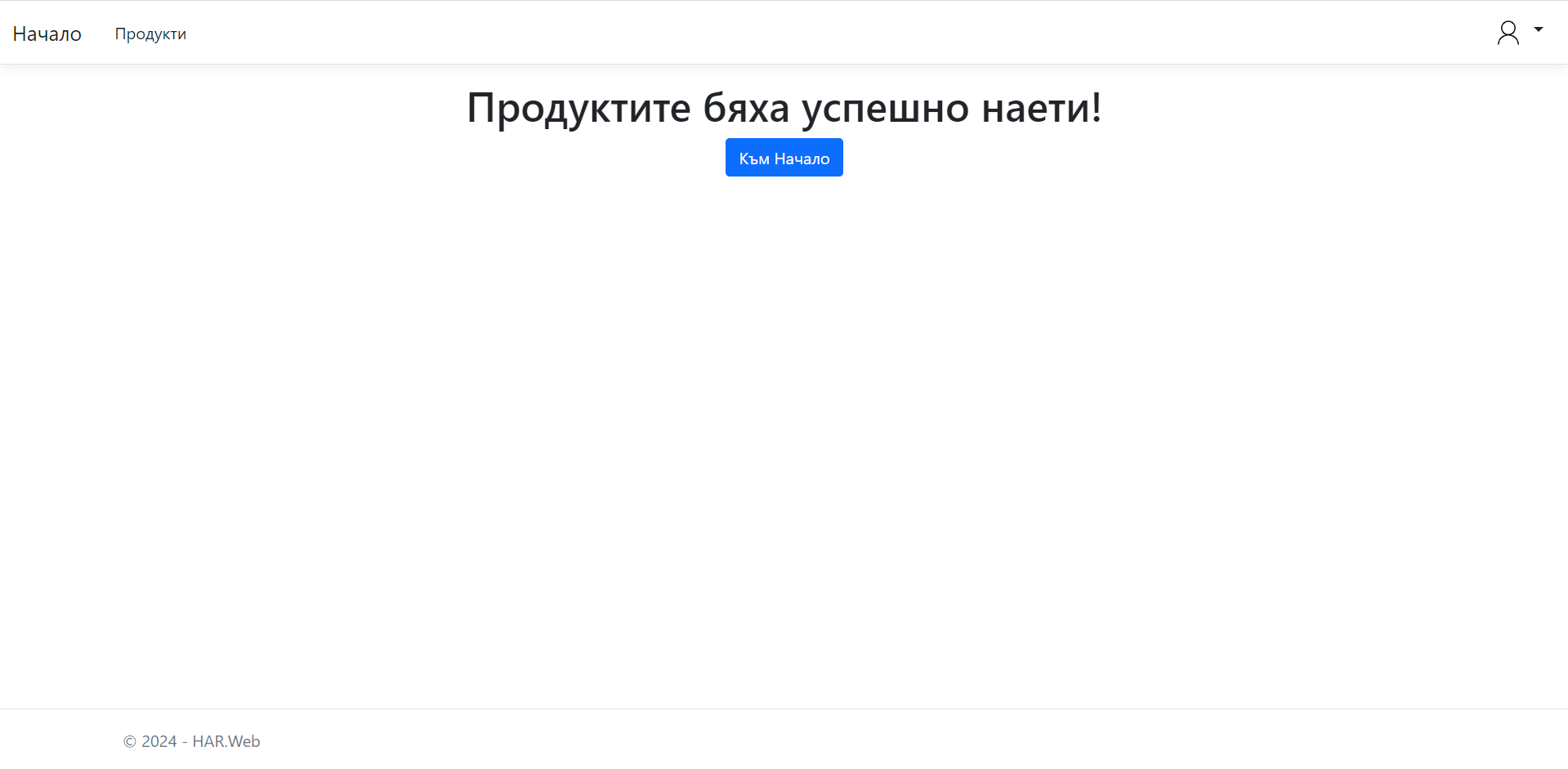
Фигура 1 Снимка на примерен интерфейс за регистриране на потребител



Фигура 2 Снимка на примерен интерфейс за изписване на наличните продукти



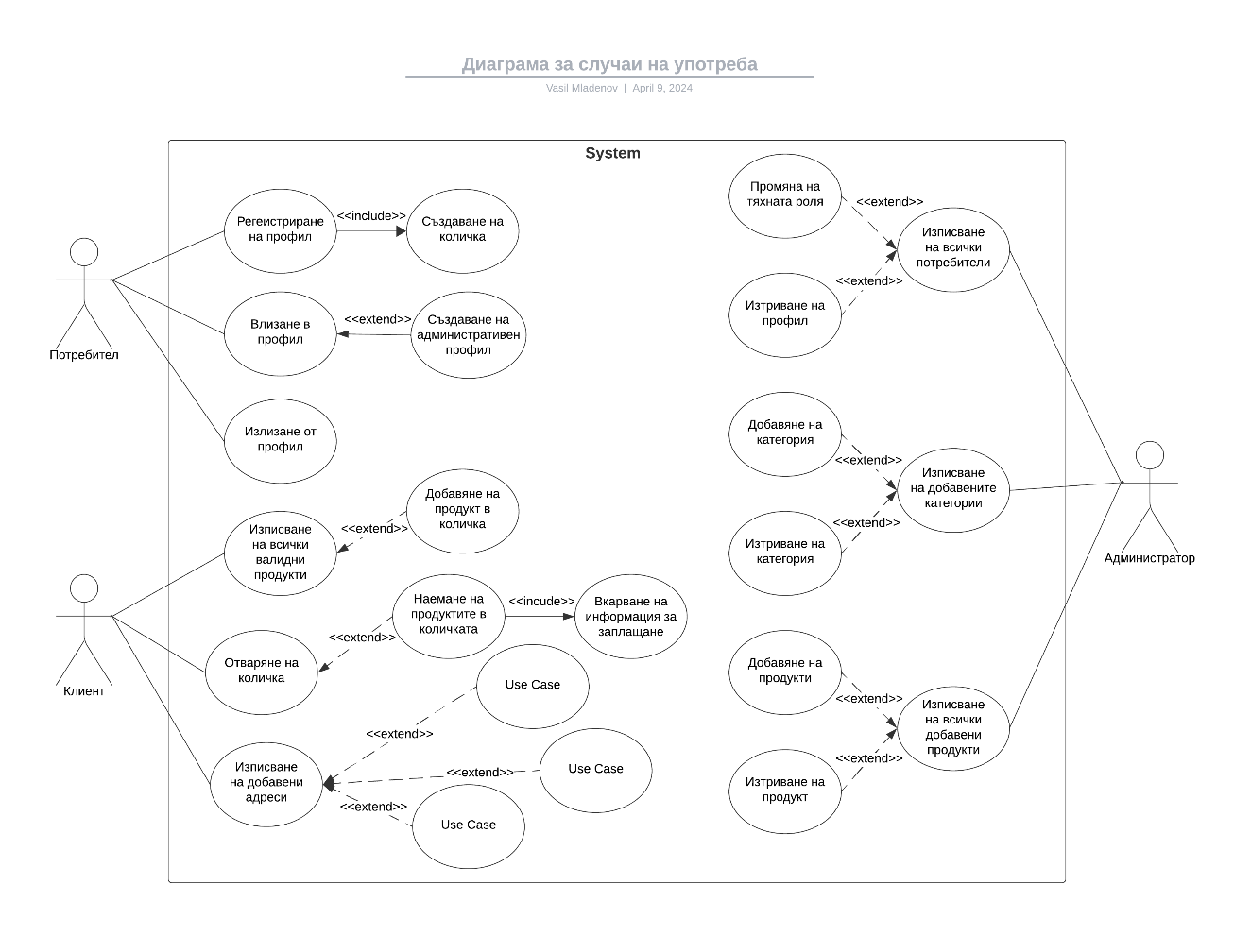
Фигура 3 Снимка за примерен интерфейс за количката на потребителя



Фигура 4 Снимка на примерен интерфейс за успешно наети продукти

## Диаграми на анализа

### Диаграма на случаи за употреба

Фигура 5 Диаграма на случаите за употреба (Use Case)

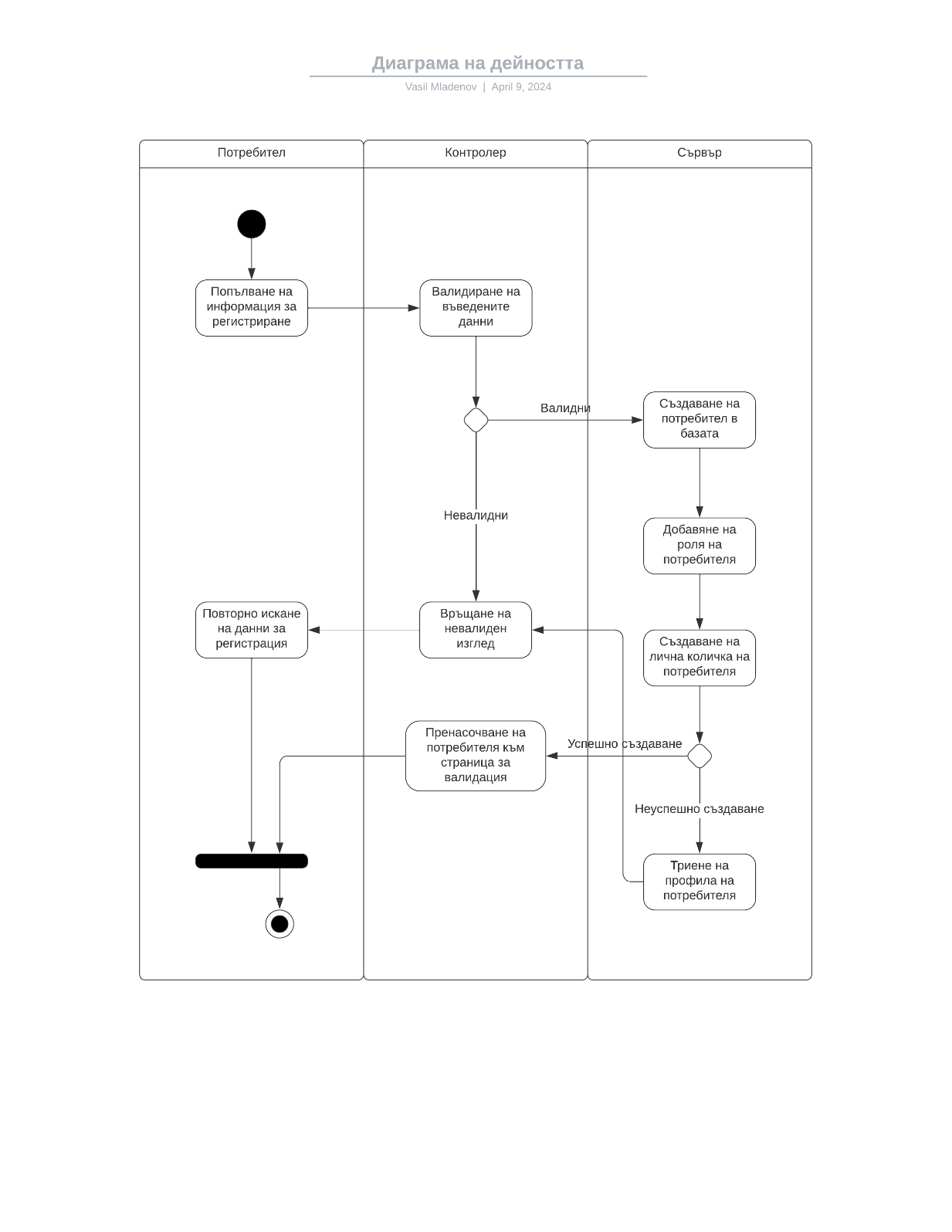
Работният процес споделя прилики и в двете роли (Клиент, Администратор).

Потребителите представляват всички лица, които използват системата, тоест включва клиентите и администраторите. Те могат да се регистрират и валидират.

Клиентите могат да виждат всички адреси, които са добавили в профилите си, предназначени за използване при наемането на продуктите. Клиентите също така могат да достъпват и личната си информация и да я обновяват. Другата възможност на клиентите е да избират от наличните продукти, добавени в системата от администраторите, и да ги добавят в количката си, от там те могат да ги наемат.

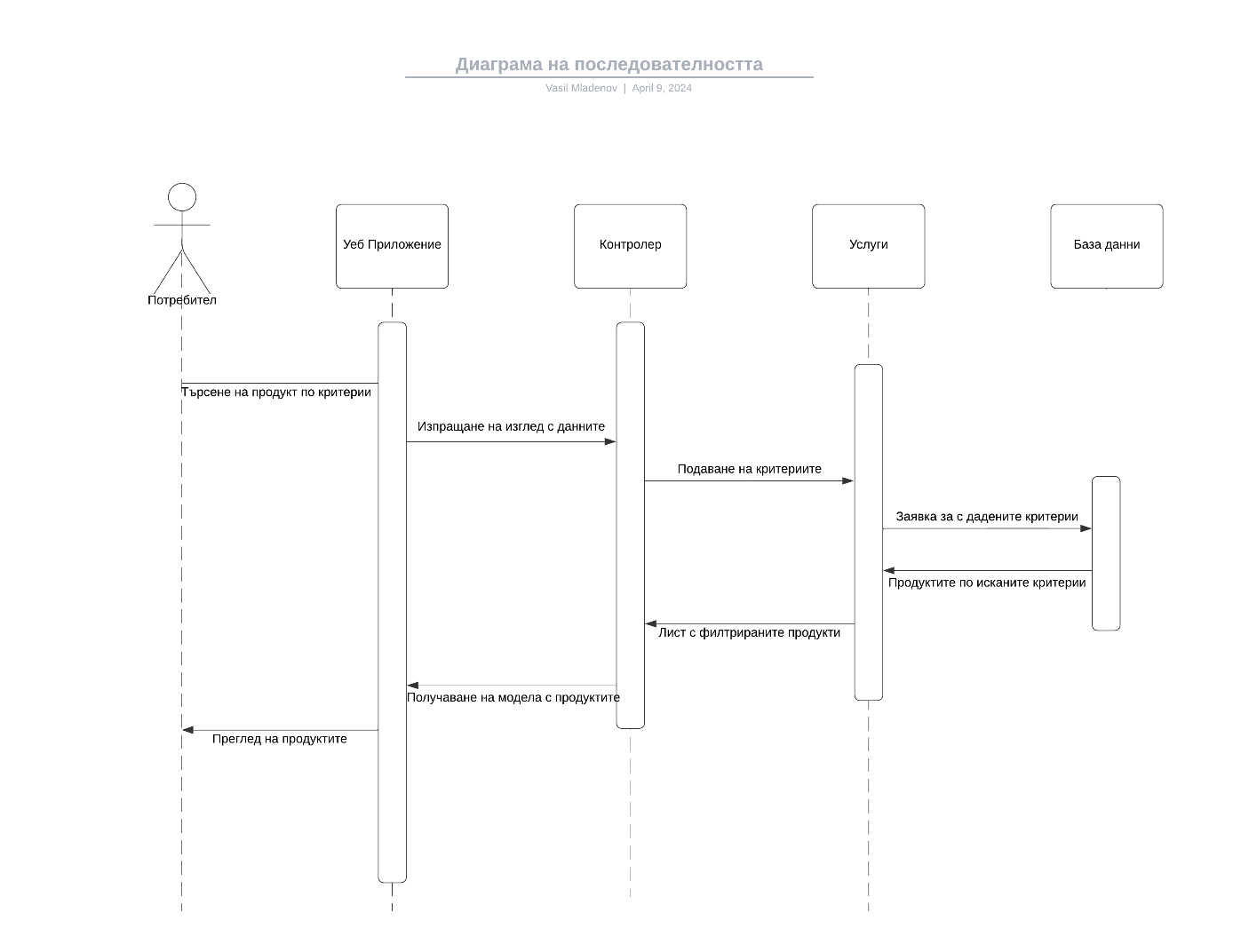
Администраторите притежават най-много функционалности, тъй като те имат възможността да оперират с продуктите, наемите, категориите и всички създадени профили. Те могат да променят данните въведени във всички обекти от базата, което улеснява боравенето с данните.

### Диаграма на дейността

Фигура 6 Диаграма на дейността (Activity)

Диаграмата на дейността (Фигура 6), описва процеса на една от по-сложните операции в системата – регистрирането на потребители. Регистрирането се случва когато потребителя въведе данните си, контролера проверява тяхната валидност и ако са валидни ги изпраща към сървъра, който добавя потребителя в базата данни и му създава количка. Ако количката е създадена успешно се връщаме към контролера и той препраща потребителя към страницата за валидиране на профил. В случай че някое от действията е неуспешно, контролера връща потребителя към страницата за регистриране и изписва проблемите.

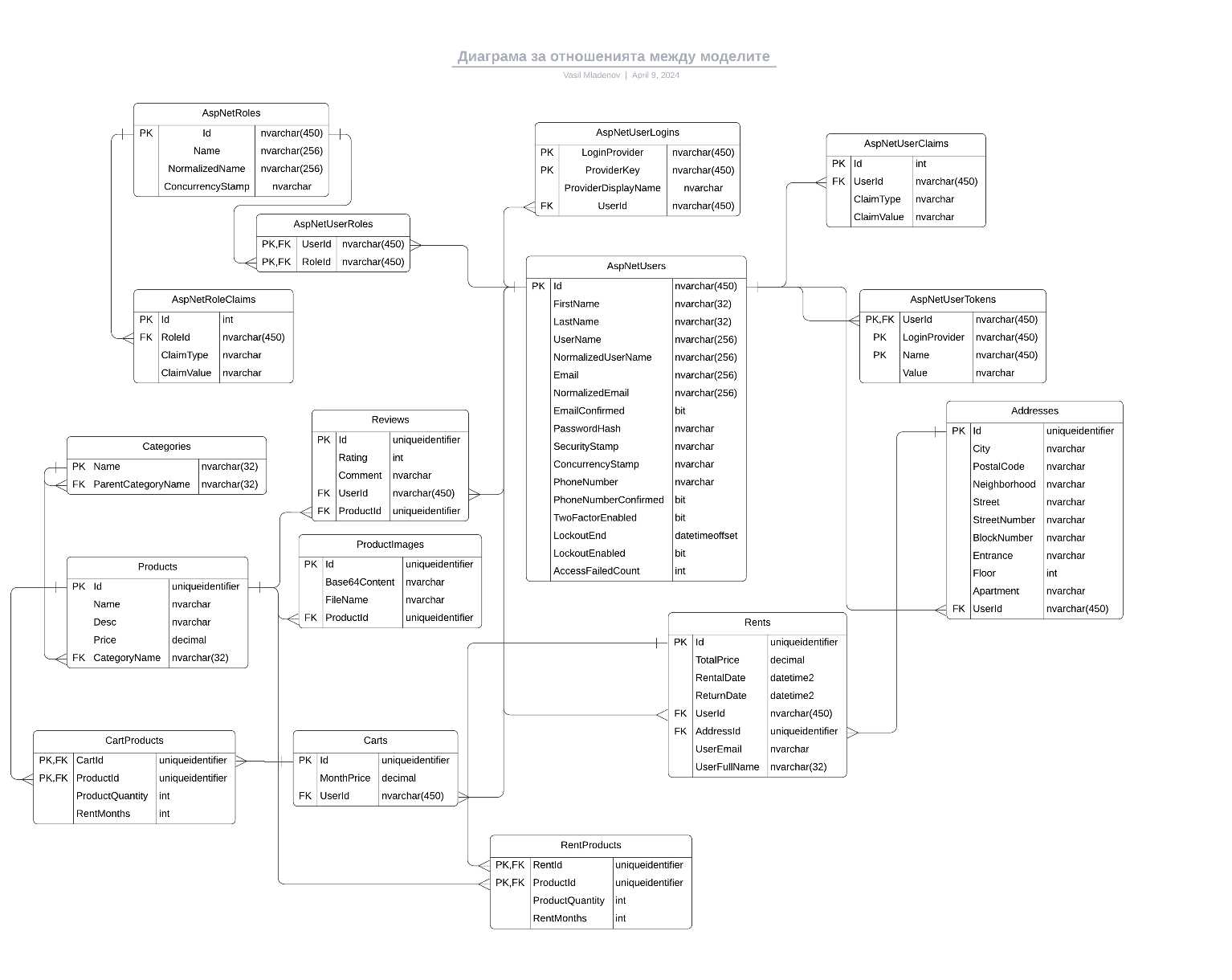
### Диаграма на последователността

Фигура 7 Диаграма на последователността (Sequence)

Диаграмата на последователността (Фигура 7), показва най-често изпълняваната операция в системата, изкарването на всички налични за наемане продукти. Начинът, по който протича тази операция е следния: потребителят въвежда критериите, по които би искал да филтрира продуктите, тези критерии се изпращат към контролера, който достъпва услугите и подава критериите. Услугата изпраща заявка към базата с дадените критерии и тя връща листа с продуктите. От там услугата връща на контролера листа с продуктите, а той ги изписва на уеб приложението.

## Модел на съдържанието / данните

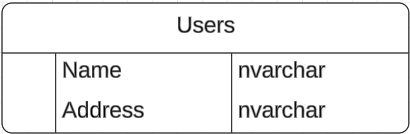
### Диаграма за отношенията между моделите

Фигура 8 Диаграма за отношенията между моделите (Entity Relationship)

### Нормализация

Базата данни представлява съвкупност от таблици, свързани релационно помежду си, чрез множество кардинални връзки. Нормализацията е начин на подредба на релационни бази от данни, където се цели запазването на целостта данните и намаляването на тяхното повтаряне – един запис не трябва да се повтаря и съответна трябва да бъдат построени отношения между таблиците, за да се осъществи това. Макар че съществуват повече от 3 нормални форми, проекта е нормализиран до трета, защото тя е стандарта и въвеждането на повече форми значително би усложнило процесът по извличане на данни. За осъществяването на всяка нормална форма е нужно условията не предишната нормална форма да бъдат изпълнени. Така се получава инкрементален процес на зависимости при всяка нормална форма.

Нека започнем от първа нормална форма. Нейните изисквания са, че таблицата, която разглеждаме трябва задължително да има първичен ключ и стойностите във всеки ред да са атомични ( напълно единични, съществуването на списъци или няколко записа наведнъж е строго забранено ). Всеки ред трябва да бъде уникален, наличието на първичен ключ осигурява това. Нека вземем една много намалена и примитивна версия на таблицата Users за пример. Първоначално, тя би изглеждала така:

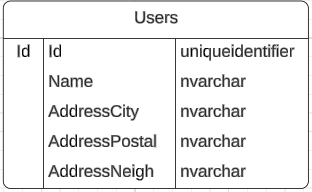


Фигура 9 Диаграма на ненормализирана таблица

|  |  |
| --- | --- |
| Name | Address |
| Vasil | Burgas, 8000, Vazdrajdane… |
| Peter | Sofia, 1000, Lazur |
| Kalin | Plovdiv, 4000, Vazov |

Таблица 1

В тази извадка от данни няма колона, която уникално да идентифицира потребителя, защото имената може да се повтарят, а адресите – да се сменят. Колоната за адреси също съдържа списък с данни, по точно – града, пощенски код и жилищен комплекс, което нарушава правилото на първа нормална форма за атомичност. За да се оправи това, може да създадем нова колона с име Id – първичен ключ, който да е уникален за всеки запис. Съответно може да се раздели адреса на три колони, които по – специфично указват града и държавата. С тези промени таблицата би изглеждала така:



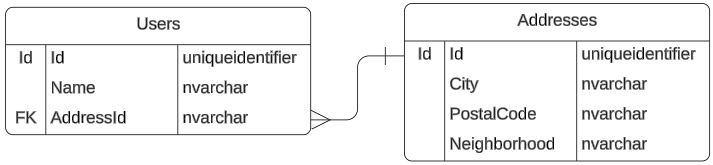
Фигура 10 Диаграма на таблица в първа нормална форма

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Id | Name | AddressCity | AddressPostal | AddressNeigh |
| 1 | Vasil | Burgas | 8000 | Vazrajdane |
| 2 | Peter | Sofia | 1000 | Lazur |
| 3 | Kalin | Plovdiv | 4000 | Vazov |

Таблица 2

Втората нормална форма изисква първоначална първа нормална форма. Това условие е изпълнено. Втората форма гласи, че всеки атрибут на таблица трябва да е функционално зависим от първичния ключ.

Таблицата преминава във втора нормална форма, когато анализираме полетата и стигнем до заключението, че не всички колони зависят напълно от първичния ключ Id. AddressPostal и AddressNeigh не зависят единствено от Id, а зависят и от текущия адрес. Този проблем се оправя, когато създадем отделна таблица за адресите, ето така:



Фигура 11 Диаграма на таблица във втора нормална форма

|  |  |  |
| --- | --- | --- |
| Id | Name | AddressId |
| 1 | Vasil | 1 |
| 2 | Peter | 2 |
| 3 | Kalin | 3 |

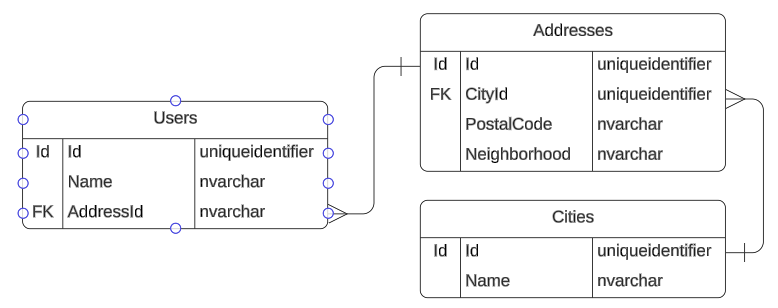
Таблица 3

|  |  |  |  |
| --- | --- | --- | --- |
| Id | City | PostalCode | Neighborhood |
| 1 | Burgas | 8000 | Vazrajdane |
| 2 | Sofia | 1000 | Lazur |
| 3 | Plovdiv | 4000 | Vazov |

Таблица 4

Втората нормална форма е постигната. Изискванията за трета нормална форма са, че трябва информацията да е нормализирана във втора норма и колоните на таблицата да са устроени по такъв начин, че промяната на запис в една колона не би се отразил на верността на информацията в друга колона, или така наречената транзитивна зависимост. Нека приемем, че има колона А, която има референтна връзка към колона Б, но колона Б, от своя страна, има референтна връзка към колона В. Следователно, колона А реферира, или е свързана с колона В транзитивно.

Сегашното състояние на данните отразява факта, че ако променим града, то тогава полетата с информацията за пощенския код и комплекса няма да са валидни, защото просто са грешна информация за тази друга държава. За да запазим целостта на данните, трябва да разделим таблицата с адреси на други таблица, които се отнасят към градовете:



Фигура 12 Диаграма на таблица в трета нормална форма

|  |  |  |
| --- | --- | --- |
| Id | Name | AddressId |
| 1 | Vasil | 1 |
| 2 | Peter | 2 |
| 3 | Kalin | 3 |

Таблица 5

|  |  |  |  |
| --- | --- | --- | --- |
| Id | CityId | PostalCode | Neighborhood |
| 1 | 1 | 8000 | Vazrajdane |
| 2 | 2 | 1000 | Lazur |
| 3 | 3 | 4000 | Vazov |

Таблица 6

|  |  |
| --- | --- |
| Id | Name |
| 1 | Burgas |
| 2 | Sofia |
| 3 | Plovdiv |

Таблица 7

С тези промени таблиците вече са нормализирани в трета нормална форма и може да видим кой е от Бургас като целостта на тези данни е съхранена, чрез механизма за ограничения в базата данни. Аналогично, за останалите таблици в базата, стигаме до заключението, че тя е нормализирана в трета нормална форма.

### Пояснения за таблиците

Структурата на таблиците в базата данни е от решаващо значение за функционирането на системата за наемане на битова техника. Ще разгледаме внимателно всяка от тези таблиците и ще обясним защо са устроени по определен начин.

На първо място, таблиците, генерирани от рамката Identity, като AspNetUsers, са от изключително значение за потвърждаването на самоличността на потребителите и тяхното упълномощаване. Въпреки че проектът не взаимодейства директно с тези таблици, техното им присъствие е от съществено значение за гарантирането на работоспособността на системата.

Таблицата Addresses съхранява информация за адресите, до които ще се доставят наетите продукти. Тази таблица е ключова за осигуряването на точност и прецизност при изпращането на наетите артикули до потребителите. Без нея би било трудно да се осигури коректната доставка на продуктите на клиентите.

Categories таблицата съдържа всички категории, които могат да се приложат на продуктите. Важно е да се подчертае, че категориите могат да имат родителска категория, което позволява по-голяма гъвкавост при организацията на продуктите в системата. Това улеснява навигацията на потребителите и прави процеса на търсене по-ефективен.

Products таблицата съхранява информация за всички налични продукти, които могат да бъдат наети от потребителите. В нея се съхраняват основните характеристики на всякакъв вид битова техника. Има възможност за свързване с таблицата ProductImages, където се съхраняват снимките на продуктите, което подобрява визуалното представяне на продуктите и улеснява избора на клиентите.

Някои от таблиците имат колони с чужди ключове, които могат да бъдат празни. Например, таблицата Rents съхранява информация за наемите на продуктите и може да съдържа чужд ключ, свързан със запис в таблицата за потребителите. Това е направено с цел да се запазят данните за наемите, дори ако потребителят изтрие своя профил от системата, като се гарантира цялостността на данните.

Всички тези таблиците са стратегически устроени, за да осигурят ефективно и надеждно функциониране на системата за наемане на битова техника. Техните правилна структура и взаимосвързаност са от ключово значение за оптималното функциониране и изпълнение на приложението.

# Дизайн

Проектът е изграден върху платформата .NET Core, с езикът C#. Този избор осигурява гъвкавост и мощност, благодарение на обширния набор от инструменти и възможности на .NET Core. В проекта се използва версия 8 на .NET Core, както и на останалите библиотеки, разработени от Microsoft.

Софтуерът е структуриран в три основни слоя - слой на бизнес логиката, слой за достъп до базата данни и презентационен слой, който представлява уеб приложението, изградено с помощта на ASP.NET Core, използващо архитектурния модела MVC (Model-View-Controller). Система е разделен на 5 проекта:

* слой на бизнес логиката;
* слой за достъп до базата данни;
* проект за тестване на бизнесовия слой;
* уеб приложение (презентационен слой);
* споделен проект, който улеснява правилното писане на код.

За постигане на целите си, приложението използва набор от важни зависимости, сред които са:

* Entity Framework Core 8: Рамка, която улеснява достъпа до базата данни и работата с нея. Проектът е разработен с помощта на подхода "Code First", където първо се пишат моделите на данните, а после се генерират съответните таблици и връзки в базата данни;
* EF Core 8 InMemoryDatabase: Тази зависимост позволява създаването на фалшива база от данни, която съществува само в паметта на приложението. Това е изключително полезно при тестване на различни функции и е предпочитано пред фалшифицирането на данни с други рамки като Moq;
* AutoMapper: Библиотека, която улеснява преобразуването на типовете между моделите на данните и изгледите, които се представят на потребителите;
* Identity: Рамка, която се интегрира в ASP.NET Core MVC и използва EF Core 8, за да реализира потвърждаването на самоличността на потребителите, управлението на техните роли и упълномощаването им;
* xUnit: Рамка, която се използва за единично тестване на различните компоненти на системата.

Приложението също така използва шаблони за дизайн като Инжектиране на зависимости (Dependency Injection) и Обръщане на контрол (Inversion of Control), които подобряват поддръжката и разширяемостта на системата, както и улесняват тестването. Тези шаблони се използват главно в уеб приложението, където е необходимо да се инжектират класове от бизнес слоя като услуги.

## Реализация на архитектурата на приложението

A screenshot of a computer screen

Description automatically generatedФигура 13 Диаграма на общата файлова архитектура на приложението с папки и имеви пространства

## Описание на слоевете, предназначението им, библиотеки и методи включени в съответния слой.

### A computer screen shot of a computer Description automatically generatedСлой на бизнес логиката

Фигура 14 Клас диаграма на услугите в визнесовия слой

Бизнесовия слой съдържа основните бизнес функционалности на приложението. Разделен е на няколко “namespace”-а или папки:

* Contracts – в тази папка се намират всички интерфейси на услугите. Това включва дефинирането на интерфейсите за всички функционалности, които приложението предлага. Интерфейсите служат като договор между слоя на бизнес логиката и останалите компоненти на приложението, като контролерите и слоя за достъп до базата данни;
* Implementations – в тази папка се намират всички класове, които имплементират интерфейсите от папката Contracts. Тук се съдържа конкретната имплементация на бизнес логиката за различните модели от базата данни. Тези класове включват методи, които реализират конкретната логика за всяка отделна функционалност на приложението;
* ViewModels – тук се намират всички класове, които изпълняват ролята на модели за изглед. Тези модели съдържат информация, която може да бъде въведена от контролерите и използвана в изгледите за представяне на данните на потребителите. Моделите за изглед обикновено са предназначени за входни данни от потребителя или за представяне на резултатите от бизнес логиката към потребителя.

Слоя за бизнес логиката играе решаваща роля в приложението, като осигурява цялостната функционалност и взаимодействие между различните компоненти на системата. Структурата му, разделена на интерфейси и техните конкретни имплементации, помага за постигане на гъвкавост, разширяемост и управление на взаимозависимостите в приложението.

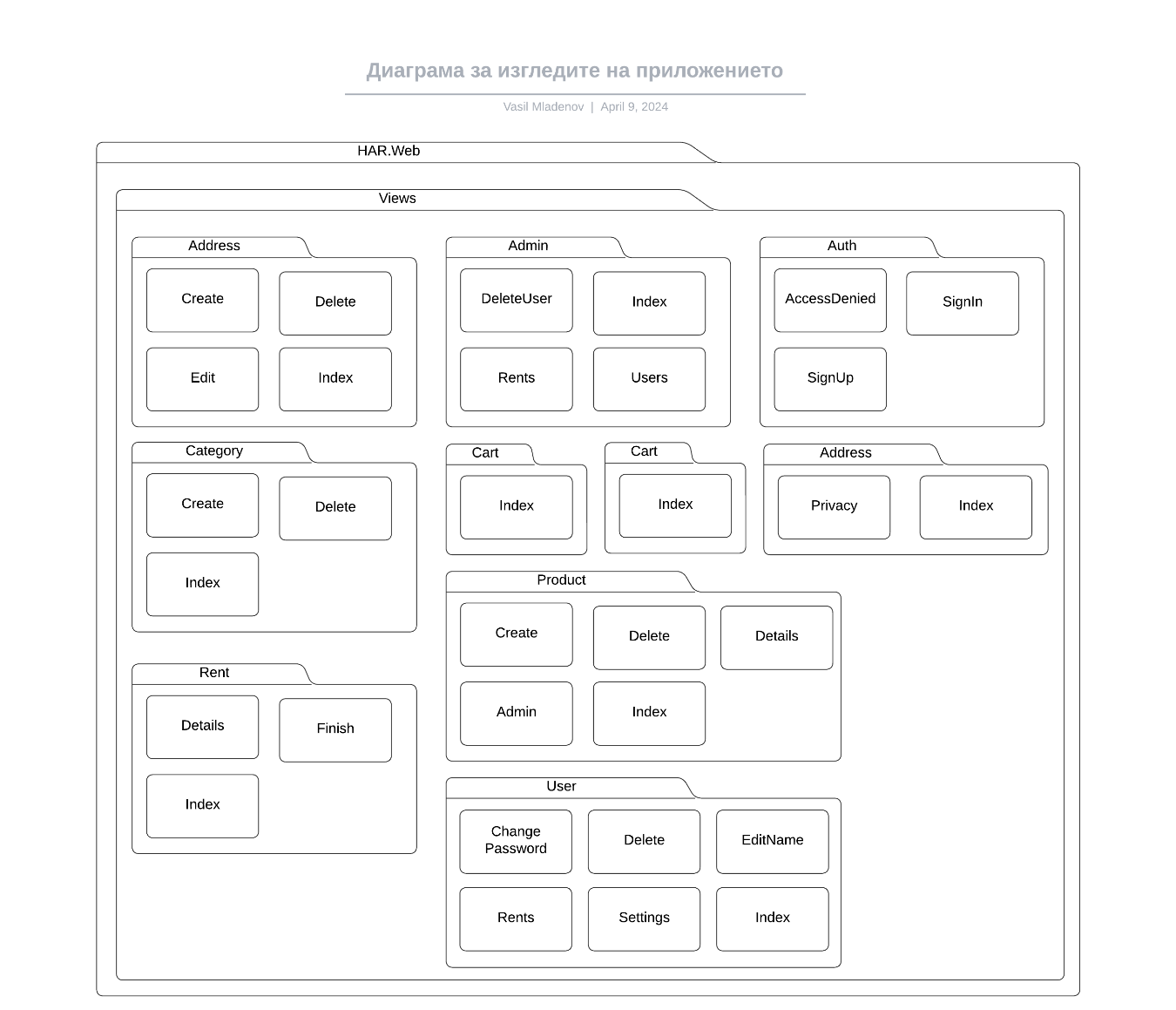
### Слой на достъпа до база данни

A screenshot of a computer

Description automatically generatedВ слоя за достъп до базата се пазят самите модели, с които работи приложението, както и класът за контекста на базата данни, с които работи рамката Entity Framework Core.

Фигура 15 Клас диаграма на моделите в базата данни

### Уеб приложение

Тук е самото уеб приложение, което използва MVC (Модел-Изглед-Контрольор) модела на работа. В него се съдържат главно изгледите и контролерите, които използват логиката от бизнес слоя на приложението. Моделите, по които са създадени изгледите са различни от моделите, с които работи базата. Тук работата се случва с модели на изгледа.

Фигура 16 Диаграма на изгледите и техните папки в уеб приложението

На диаграмата по-отгоре може да видим всички изгледи, които се добавени в уеб приложението. Те се намират и в папки, което улеснява търсенето им по файлове.

A computer screen shot of a computer network

Description automatically generated with medium confidence

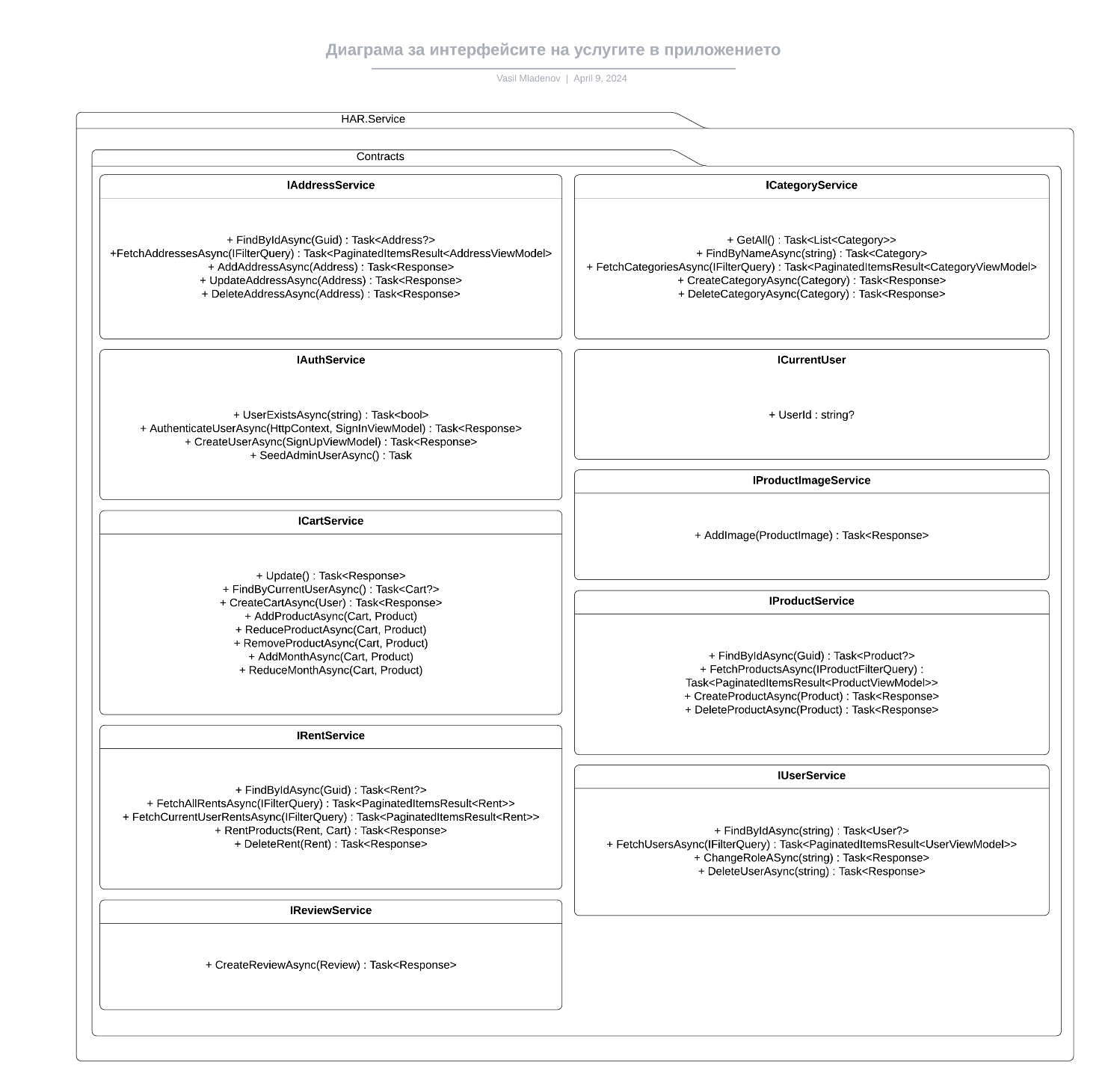
Фигура 17 Клас диаграма на контролерите в уеб апликацията

## Организация и код на заявките към база от данни

Операциите за достъпване с базата данни се намират изцяло в слоя за бизнес логика който извършва връзка до слоя за достъп с базата данни.

Операциите са намират в няколко имеви пространства или папки.

### Contracts

В това имево пространство се намират всички интерфейси, които биват наследени от услугите намиращи се в папката Implementations. По-долната диаграма показва структурата на тези интерфейси.

Фигура 18 Диаграма на файловата архитектура на интерфейсите на услугите

* IAddressService – този интерфейс представлява контракт за работа с адресите на потребителите. Предоставя възможност на потребителите да управляват личните си адреси, като им позволява да ги създават, редактират, изтриват и преглеждат.
  + Task<Address?> FindByIdAsync(Guid id)
  + Task<Response> AddAddressAsync(Address address)
  + Task<Response> UpdateAddressAsync(Address address)
  + Task<Response> DeleteAddressAsync(Address address)
* IAuthService – този интерфейс извършва валидация и регистрация на потребителите в системата. Освен това, включва метод за проверка на съществуващ административен профил и създаване на такъв, ако такъв не съществува.
  + Task<bool> UserExistsAsync(string email)
  + Task<Response> CreateUserAsync(SignUpViewModel model)
  + Task SeedAdminUserAsync()
* ICartService – този интерфейс контролира всички продукти в персоналните колички на потребителите, както и съхранението и управлението на самия колички. Предоставя възможност за добавяне, премахване и управление на продуктите в количката.
  + Task<Response> Update();
  + Task<Cart?> FindByCurrentUserAsync();
  + Task<Response> CreateCartAsync(User user);
  + Task<Response> AddProductAsync(Cart cart, Product product)
  + Task<Response> ReduceProductAsync(Cart cart, Product product)
  + Task<Response> RemoveProductAsync(Cart cart, Product product)
  + Task<Response> AddMonthAsync(Cart cart, Product product)
  + Task<Response> ReduceMonthAsync(Cart cart, Product product)
* ICategoryService – този интерфейс представлява контракт за работа с категориите на продуктите. Само администраторите имат достъп до него и могат да създават, редактират и изтриват категории. Предоставя възможност за управление на категориите и тяхното свързване с продуктите.
  + Task<List<Category>> GetAll()
  + Task<Category?> FindByNameAsync(string name)
  + Task<Response> CreateCategoryAsync(Category category)
  + Task<Response> DeleteCategoryAsync(Category category)
* ICurrentUser - този интерфейс пази информация за текущо влезналия в системата потребител, като запазва само неговия идентификационен номер. Използва се за идентификация на потребителя и осигуряване на персонализирани услуги.
  + public string? UserId { get; }
* IProductImageService – този интерфейс предоставя функционалности за работа със снимките на продуктите в базата данни. Включва методи за добавяне, изтриване и преглеждане на снимки, както и връзката им с конкретни продукти.
  + Task<Response> AddImage(ProductImage productImage)
* IProductService – този интерфейс представлява услуга за работа с продуктите. Включва методи за добавяне, редактиране, изтриване и преглеждане на продукти в системата.
  + Task<Product?> FindByIdAsync(Guid id)
  + Task<Response> CreateProductAsync(Product product)
  + Task<Response> DeleteProductAsync(Product product)
* IRentService – този интерфейс представлява контракт за работа с наемането на продукти и тяхното плащане. Включва методи за създаване, обновяване и изтриване на наеми, както и за преглеждане на наети продукти.
  + Task<Rent?> FindByIdAsync(Guid rentId)
  + Task<Response> RentProducts(Rent rent, Cart cart)
  + Тask<Response> DeleteRent(Rent rent)
* IReviewService – този интерфейс представлява контракт за работа с отзивите на продуктите. Включва методи за създаване, редактиране, изтриване и преглеждане на отзиви от потребителите.
* IUserService – този интерфейс допълва вградените методи на рамката Identity и работи с ролите на потребителите. Включва функционалности за управление на потребителите, тяхното регистриране, влизане в системата и ролята им.
  + Task<User?> FindByIdAsync(string userId)
  + Task<Response> ChangeRoleAsync(string userId)
  + Task<Response> DeleteUserAsync(string userId)

Тези интерфейси представляват ключови компоненти в архитектурата на приложението. Те се градят един върху друг, като предоставят ясно дефинирани абстракции за различните аспекти на функционалността на системата. На първо място, IAuthService и IUserService предоставят възможност за управление на потребителите и техните роли в системата. След това, ICurrentUser осигурява идентификация на текущия потребител, който е от съществено значение за персонализирането на услугите.

Следващите по важност са ICategoryService, IProductService, и IProductImageService, които предоставят възможности за управление на категориите на продуктите, самия продукт и свързаните с тях изображения. Тези интерфейси са основни за функционирането на системата за наемане на продукти, като осигуряват правилното организиране и визуализация на наличните артикули.

След тях, ICartService и IRentService предоставят възможности за управление на количките на потребителите и процеса на наемане на продукти. Те са от решаващо значение за потребителския опит и успешното изпълнение на заявките за наем на продукти.

Накрая, IAddressService и IReviewService осигуряват възможности за управление на адресите на потребителите и отзивите за продуктите. Те допълват функционалността на системата, като предоставят допълнителни удобства и функционалности за потребителите.

Всички тези интерфейси заедно формират основата на приложението, като осигуряват гъвкавост, лесна поддръжка и разширяемост на функционалността му. Използването на добре дефинирани интерфейси помага за разделението на отговорностите в приложението и подобрява общата му архитектура.

### IdentityModel

Фигура 19 Диаграма на моделите от рамката Identity

Това имево пространство / папка описва всички модели, които се генерират автоматично от рамката Identity, която ние получаваме от ASP.NET Core.

Генерираните модели биват следните

* AspNetUsers
* AspNetRoles
* AspNetUserRoles
* AspNetUserLogin
* AspNetUserToken
* AspNetUserClaim
* AspNetRoleClaim

### A screenshot of a computer Description automatically generatedModels

Фигура 20 Клас диаграма на моделите в базата данни

Това имево пространство / папка съдържа основните модели, по които е създадена базата данни, и с които работи приложението. Списък с модели в тази папка:

* Address
* Cart
* CartProduct
* Category
* Product
* ProductImage
* RentProduct
* Rent
* Review
* User – наследява IdentityDbUser, и му добавя име и фамилия

### Configurations и Data

В папката за слоя за достъп за базата има още две папки. Едната е Configurations, а другата е Data. В папката Configurations седят следните файлове:

* AddressConfiguration
* CartConfiguration
* CartProductConfiguration
* CategoryConfiguration
* ProductConfiguration
* ProductImageConfiguration
* RentConfiguration
* RentProductConfiguration
* ReviewConfiguration

Във всички тези файлове се пояснява на базата как трябва да реагира при изтриване на техните зависими полета. При някои, чуждия ключ се задава празен докато при друго се трие обектът. Тези поведения се изразяват чрез вградения в Entity Framework Core, Fluent API.

В папката Data седи „сърцето“ на слоя за достъп до базата, а именно контекстът, който управлява връзката с базата данни и мапирането на моделите към таблицата в базата данни.

Контекстът на данни, наречен DataContext, наследява от IdentityDbContext<User>, който е специализиран за потребителския модел на ASP.NET Core Identity. Това позволява лесното интегриране на системата за управление на потребителите в приложението.

В контекста на данни са дефинирани няколко DbSet свойства, които представят отделни таблици в базата данни. Те са свързани с моделите на данните, които се използват в приложението. Някои от тези модели включват Address, Cart, Category, Product, Review и други, които са необходими за функционирането на системата за наемане на продукти.

В метода OnModelCreating се извиква методът ApplyConfigurationsFromAssembly, който прилага конфигурациите за моделите от сборката, в която се намира контекстът на данни. Това позволява дефинирането на допълнителни конфигурации за моделите, като например ограничения за външни ключове и други.

Този контекст на данни представлява основата на връзката между приложението и базата данни, като предоставя възможност за извършване на операции като заявки към данните, добавяне, редактиране и изтриване на записи и други.

## Наличие на потребителски интерфейс (конзолен, графичен, уеб)

Приложението е уеб апликация, която е вдигната благодарение на рамката ASP.NET Core MVC. За стил използва библиотеките на Bootstrap, с което се спомага работата по изгледите без нуждата от опит в дизайн.

Приложението е базирано на ASP.NET Core MVC и Bootstrap, представлява предизвикателство, което изисква внимателно съчетание на функционалност и дизайн, за да се осигури удобство и ефективност за потребителите.

Потребителският интерфейс е ключов компонент във всяка такава система. Началната страница трябва да привлече вниманието на потребителите с представяне на наличната битова техника за наем и актуални промоции. Също така, трябва да предостави лесен начин за търсене на желаните продукти. Страницата за продукти трябва да представи списък с всички налични продукти с информация за всяко една цена, наличност и кратко описание. При избор на продукт, страницата за детайли трябва да осигури подробна информация, включително снимки, технически спецификации и условия за наем.

Интерактивната количка е важна функционалност, която позволява на потребителите да добавят, премахват или редактират продуктите в количката си преди да завършат поръчката. Страницата за завършване на поръчката предоставя форма за въвеждане на информация за доставка и плащане, която потребителите трябва да попълнят преди да потвърдят поръчката си.

Относно функционалностите за потребителя, системата трябва да предостави възможност за търсене и филтриране на продуктите, регистрация и вход в системата, преглед на предходни поръчки и управление на профила на потребителя. Тези функции трябва да бъдат представени по начин, който да е интуитивен и лесен за използване.

Административният панел трябва да осигури удобен интерфейс за управление на продуктите, поръчките и потребителите. Администраторът трябва да може да добавя, редактира и изтрива продукти, както и да преглежда и обработва поръчки. Управлението на потребителите също трябва да бъде възможно, включително триенето на акаунти или промяна на данните им.

Базата данни играе ключова роля за съхранение на информацията за потребителите, продуктите, поръчките и други важни данни. Тя трябва да бъде добре проектирана и оптимизирана, за да осигури бърз и надежден достъп до информацията.

Тези елементи съчетават технология и дизайн, за да създадат иновативно и удобно за използване приложение за наемане на битова техника, което отговаря на нуждите на своите потребители.

# Ефективност и бързодействие на решението

Проектът не работи с алгоритми от висока сложност. По–голямата част от логиката на приложението е свързана с обработката на данните на потребителите, чрез базата данни. Това се случва, през многобройните хранилищата, които използват методите на LINQ (Language Integrated Queries) за да обслужи заявките на потребителите. Поради тази причина, равносметката за бързодействието и ефективността може да се направи, като се разгледа сложността на алгоритмите, използвани от LINQ.

Повечето методи са със сложност O(n), защото просто обхождат колекцията. Примери за методи с така сложност са Select(), Single(), First(). Приложението употребява тези методи в имплементацията на методи за четене от базата данни, като GetDetails(), GetAllRecords() В тези случаи сложността на методите е O(n), но има и други методи, като OrderBy(), които имат логаритмична сложност (O(n logn)). Този метод се вика от хранилищата при търсенето и страницирането (Paging) на записи. (Banimahd, 2023)

Методът Where() е по – особен, защото няма точна сложност. Приложението го използва многократно в методи, като GetWhere(), GetDetails(), IsUserInRole(). Истината е, че неговата сложност зависи от метода, който кара изброимата колекция да се изпълни, което най – често е .ToList(). Това придава на изброените методи сложност O(n).

Обобщено, сложността на методите, извикани от приложението е променлива, спрямо обстоятелствата, в които са повикани. Най – чести обаче са сложностите O(n logn) и O(n).

# Тестване

Проектът е тестван, чрез рамката xUnit. Има един отделен проект за тестване: HAR.Tests, който тества слоя за бизнес логика. Конвенция за имена: ИмаНаМетод\_ТрябваДаИзвършиДаденоДействие().

A screenshot of a computer

Description automatically generated

Фигура 21 Клас диаграма на тестовете на бизнес слоя

Слоят за тестване на бизнес логиката играе важна роля в уверяването, че функционалностите на приложението работят правилно и отговарят на изискванията на потребителите. Този проект, се състои от четири файла, които тестват основните услуги на приложението - AddressService, CartService, CategoryService и ReviewService.

* AddressServiceTests – тестовете в този файл целят да проверят функционалността за създаване, изтриване, преглеждане и обновяване на адресите на потребителите. Това е от изключителна важност, тъй като адресите са ключова информация за доставката на наетите продукти. Тестването на тези функционалности гарантира, че адресите се обработват правилно и потребителите могат да получат своите продукти на правилния адрес
* CartServiceTests – в този файл се тестват функционалностите, свързани с управлението на количката на потребителите. Тук се проверява дали потребителите могат успешно да добавят, премахнат, увеличат или намалят броя наети продукти в своята количка. Тъй като наличието на продукти в количката е от съществено значение за завършване на поръчката, този слой от тествания е от ключово значение за правилното функциониране на приложението
* CategoryServiceTests – в този файл се тестват функционалностите, свързани с управлението на количката на потребителите. Тук се проверява дали потребителите могат успешно да добавят, премахнат, увеличат или намалят броя наети продукти в своята количка. Тъй като наличието на продукти в количката е от съществено значение за завършване на поръчката, този слой от тествания е от ключово значение за правилното функциониране на приложението
* ReviewServiceTests – в този файл се тестват функционалностите, свързани с управлението на количката на потребителите. Тук се проверява дали потребителите могат успешно да добавят, премахнат, увеличат или намалят броя наети продукти в своята количка. Тъй като наличието на продукти в количката е от съществено значение за завършване на поръчката, този слой от тествания е от ключово значение за правилното функциониране на приложението

В заключение, тестовете в тези файлове играят ключова роля в уверяването, че бизнес логиката на приложението работи правилно и отговаря на очакванията на потребителите, като осигурява коректно изпълнение на важните функционалности като управление на адресите, количките, категориите и отзивите на продуктите.

# Заключение и възможно бъдещо развитие

Проектът е разработен до такава степен, че да изпълнява всички поставени цели. Архитектурата на приложението е подпомогната от добрите практики, спазени при създаването моделите, изгледите и всички класове и интерфейси. Създадена е среда за управление на продуктите от страна на администраторите.

Приложението предоставя възможността на всички роли да следят, наблюдават и участват в процеса по намиране на желания продукт, по–активно и осъзнато.

Клиентите имат възможност да маркират критерии на продуктите, което значително подпомага бързото и лесно намиране на продуктите, с които биха искали да се сдобият.

Бъдеща доразработка на проекта предвижда включването и въвеждането на модерен изглед и елегантен външен вид на приложението, както и набор от нови функционалности и параметри, по които клиентите по подробно биха могли да търсят и сортират подходящи продукти.

Друга част от проекта, която се предвижда като развитие е възможност за сътрудничество с куриерски фирми.

# Използвани литературни източници и Уеб сайтове

Banimahd, M. (25 1 2023 r.). *Understanding Time Complexity of C# List and LINQ Methods.* Изтеглено на 15 4 2023 r. от https://masoudx.medium.com/understanding-time-complexity-of-c-list-and-linq-methods-6b5e36d66243

Bootstrap. (н.д.). *Bootstrap*. Извлечено от Bootstrap: https://getbootstrap.com/docs/4.1/getting-started/introduction/

*Database Normalization: A Step-By-Step-Guide With Examples*. (26 9 2022 r.). Извлечено от Database Star: https://www.databasestar.com/database-normalization/

Lucidchart. (н.д.). *UML Class Diagram Tutorial*. Извлечено от Lucidchart: https://www.lucidchart.com/pages/uml-class-diagram

Microsoft. (н.д.). *ASP.NET Core Documentation*. Извлечено от Learn Microsoft: https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-8.0

Microsoft. (н.д.). *C# Documentation*. Извлечено от Microsoft: https://learn.microsoft.com/en-us/dotnet/csharp/

Microsoft. (н.д.). *Entity Framework Core Documentation*. Извлечено от Microsoft: https://learn.microsoft.com/en-us/ef/core/

Microsoft. (н.д.). *SQL Server Technical Documentation*. Извлечено от Microsoft: https://learn.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver16

Mozilla. (н.д.). *HTML: HyperText Markup Language*. Извлечено от Mozilla Developer: https://developer.mozilla.org/en-US/docs/Web/HTML

# Приложения

Фигура 1 Снимка на примерен интерфейс за регистриране на потребител 6

Фигура 2 Снимка на примерен интерфейс за изписване на наличните продукти 6

Фигура 3 Снимка за примерен интерфейс за количката на потребителя 7

Фигура 4 Снимка на примерен интерфейс за успешно наети продукти 7

Фигура 5 Диаграма на случаите за употреба (Use Case) 8

Фигура 6 Диаграма на дейността (Activity) 9

Фигура 7 Диаграма на последователността (Sequence) 10

Фигура 8 Диаграма за отношенията между моделите (Entity Relationship) 11

Фигура 9 Диаграма на ненормализирана таблица 12

Фигура 10 Диаграма на таблица в първа нормална форма 13

Фигура 11 Диаграма на таблица във втора нормална форма 14

Фигура 12 Диаграма на таблица в трета нормална форма 15

Фигура 13 Диаграма на общата файлова архитектура на приложението с папки и имеви пространства 20

Фигура 14 Клас диаграма на услугите в визнесовия слой 21

Фигура 15 Клас диаграма на моделите в базата данни 23

Фигура 16 Диаграма на изгледите и техните папки в уеб приложението 24

Фигура 17 Клас диаграма на контролерите в уеб апликацията 25

Фигура 18 Диаграма на файловата архитектура на интерфейсите на услугите 26

Фигура 19 Диаграма на моделите от рамката Identity 30

Фигура 20 Клас диаграма на моделите в базата данни 31

Фигура 21 Клас диаграма на тестовете на бизнес слоя 36

Таблица 1 12

Таблица 2 13

Таблица 3 14

Таблица 4 14

Таблица 5 16

Таблица 6 16

Таблица 7 16

# Критерии и показатели за оценяване

|  |  |  |
| --- | --- | --- |
| Критерии и показатели за оценяване | Максимален брой точки за показателите | Максимален брой точки за критерия |
| 1. Съответствие с изискванията за съдържание и структура на дипломния проект |  | 20 |
| 1. 1. логическа последователност и структура на изложението, балансиране на отделните части | 4 |  |
| 1.2. задълбоченост и пълнота при формулиране на обекта, предмета, целта и задачите в разработването на темата | 7 |  |
| 1.3. използване на подходящи изследователски методи | 4 |  |
| 1.4. стил и оформяне на дипломната работа (терминология, стил на писане, текстообработка и оформяне на фигури и таблици) | 5 |  |
| 2. Съответствие между поставените цели на дипломния проект и получените резултати |  | 20 |
| 2.1. изводите следват пряко от изложението, формулирани са ясно, решават поставените в началото на изследването цели и задачи и водят до убедителна защита на поставената теза | 10 |  |
| 2.2. оригиналност, значимост и актуалност на темата | 6 |  |
| 2.3. задълбоченост и обоснованост на предложенията и насоките | 4 |  |
| 3. Представяне на дипломния проект |  | 20 |
| 3.1. представянето на разработката по темата e ясно и точно | 5 |  |
| 3.2. онагледяване на експозето с:  а) презентация;  б) графични материали;  в) практически резултати;  г) компютърна мултимедийна симулация и анимация | 10 |  |
| 3.3. умения за презентиране | 5 |  |
| 4. Отговори на зададените въпроси от рецензента и/или членовете на комисията за защита на дипломен проект |  | 30 |
| 4.1. разбира същността на зададените въпроси и отговаря пълно, точно и убедително | 10 |  |
| 4.2. логически построени и точни отговори на зададените въпроси | 10 |  |
| 4.3. съдържателни и обосновани отговори на въпросите | 10 |  |
| 5. Използване на професионалната терминология, добър и ясен стил, обща езикова грамотност |  | 10 |
| 5.1. Правилно използване на професионалната терминология | 5 |  |
| 5.2. Ясен изказ и обща езикова грамотност | 5 |  |
| Общ брой точки: | Максимален бр. точки 100 | Максимален бр. точки 100 |