МИНИСТЕРСТВО НА ОБРАЗОВАНИЕТО И НАУКАТА

**ПРОФЕСИОНАЛНА ГИМНАЗИЯ “ГЕН. ВЛАДИМИР ЗАИМОВ” гр. СОПОТ**

4330 гр. Сопот, ул. ”Иван Вазов” №1, тел./факс: /03134/ 83-31, 83-32, e-mail: [pgzaimov@yahoo.com](mailto:pgzaimov@yahoo.com)

**ДИПЛОМЕН ПРОЕКТ**

**Тема:** **Разработка на уеб платформа за изготвяне и следене на хранителен режим**

*Ученик: Надежда Любенова Антонова*

***Професия:*** *код 481020 „Системен програмист“*

***Специалност:*** *код 4810201 „Системно програмиране“*

***Консултант:*** *Живко Маджев*

Сопот, 2023 г.

**Съдържание**

**Увод (Въведение) 5**   
 **ПЪРВА ГЛАВА.** Проучване **6**

1.1. Предпоставка за създаване **6**

1.2. Анализ на съществуващи решения **7**

1.3. Цел на дипломната работа **9**

**ВТОРА ГЛАВА.** Проектиране **10**

2.1. Използвани технологии и подготовка на работната среда **10**

2.2. Изисквания към софтуерния продукт **19**

2.2.1. Софтуерни **19**

2.2.2. Интерфейсни **21**

2.3. Изготвяне на Use Cases Diagrams **23**

**ТРЕТА ГЛАВА.** Описание на реализирания проект **24**

3.1. Структура на базата данни **24**

3.1.1. CodeFirst подход **25**

3.1.2. Изпълнение на миграциите **26**

3.1.3. Дизайн на базата данни **28**

3.2. Описание на CRUD операциите **29**

3.2.3. Как работи шаблона на Contoller с GET и POST заявки **32**

3.3. Работа с View моделите **34**

3.3.1. Описание на сайта

**ЧЕТВЪРТА ГЛАВА.** Анализ на получените резултати

**Заключение**

Използвана литература

**Увод**

Плановете за хранителен режим са предназначени да осигурят енергиино ефективен здравословен комфорт и спестяване на време за човека.

Здравето на човека като състояние на пълно физическо, психично и социално благополучие, е резултат от комплексното въздействие на екологични, социалноикономически, медицински и психо-емоционални фактори. Върху човека въздействат много фактори, като някои от тях укрепват здравето, а други го увреждат (замърсена жизнена среда, шум, нервно-психично напрежение, неправилно хранене, намалена двигателна активност, злоупотреба с алкохол и никотин). Човекът трябва не само да се предпазва от неблагоприятните фактори на природната и социалната среда, но да познава добре и да се възползва оптимално от т.нар. здравословни фактори на околната среда. Здравето и болестта са две форми на съществуване на човека, но между тях няма резки граници

Диета е контролирано приемане на храни и течности с цел постигане на определени резултати, измервани в килограми, мускулна маса, мазнини и др. С други думи това е хранителен режим. Управлението на пропорцията между белтъчините, мазнините и въглехидратите, които се приемат с храната е от основно значение за постигане на максимално добри резултати при управление на даден хранителен режим или при определено коригиране на килограми, поради заболяване, практикуване на спорт или някаква друга необходимост.

Постигането и поддържането на здраве е продължаващ, непрекъснат процес, който се повлиява от еволюцията на знанията за здравословен начин на живот и практиките за грижата за здравето, личните усилия и медицинската помощ.

За правилно изпълнение на хранителния режим е нужен треньор, който изготвя хранителен план спрямо изискванията на клиента. Всеки треньор е специлиазиран в определена сфера. Така всеки от тях изготвя хранителен план свързан с неговата сфера, познавайки добре функционалността на плана. Също така са необходими редактори на храни и хранене, и регистрирани диетолози.

**ГЛАВА 1 - ПРОУЧВАНЕ**

**1.1. Предпоставка за създаване на продукта**

При създаването на сайта за хранителните режими първо се направи проучване на българското и чуждестранното интернет пространство за съществуващи сайтове на подобна тематика - как са организирани тези сайтове, какъв софтуер е използван за тяхната реализация, до колко тези сайтове са изчерпателни, по какъв начин е поднесена информацията. Необходимостта от такова проучване възникна, за да може да се очертае нужния модел за разработката и за да се проучи какво потребителя интуитивно очаква да види в подобен род сайтове. Дефинираха се следните критерии за оценяване на подобни Web –сайтове:

• Въведение - разглеждане на заглавната част на даден сайт, до колко тя е атрактивна и увлекателна, до колко може да задържи вниманието на потребителя.

• Определяне потребителите на дадения сайт – кой би използвал дадения сайт.

• Интерфейс на сайтът - как изглежда сайтът графично, до колко е атрактивен или претрупан, интуитивна ли е ориентацията по страниците.

• Изчерпателност и организация на информацията - каква информация съдържа дадения сайт и до колко тя е покрила засегнатата област и по какъв начин е структурирана.

• Навигация - до колко лесно и интуитивно става предвижването по страниците на сайта.

• Скорост на зареждане - с каква скорост се зарежда сайта, до колко са оптимизирани графичните ресурси.

• Софтуер - какъв софтуер е използван за изработването на сайта.

• Динамичност на сайта - има ли възможност потребителя да общува със сайта.

**1.2. Анализ на съществуващи решения и реализации**

След нарпавено проучване по темата, бе установено съществуването на много малко на брой уеб сайтове, предлагащи подробна и систематизирана информация по съответен хранителен план; информация за методиката на изготвянето и изпълнението на хранителния режим. Повечето страници са посветени на ограничен брой планове. Съществуват и такива, които са посветени на плановете за изчистване на излишни мазнини и стягане на тялото, и сваляне на килограми, чрез калориен дефицит. Тяхно предимство е, че информацията е достъпна от едно място и потребителят може лесно да сравни начина на изработване на плановете за отделните видове.

Важно е да се отбележи, че се наблюдава тенденция сайтовете да бъдат претрупани с много реклами, отвличащи вниманието на посетителя и не са изцяло ориентирани към проучването на дадения план. Голяма част от информацията за плановете на другите видове се откриват предимно в web сайтове за здраве, но тя не е изчерпателна .

Обща черта на повечето от сайтовете, посветени на изготвянето на план е, че те не предлагат никакъв начин за комуникация с посетителите. Много малко от тях са атрактивни и предлагащи достатъчен графичен материал за описанието на плановете. Сравнение с един от сайтовете:

**Диети**

[*https://smartdiet.bg*](https://smartdiet.bg)

• Въведение - Заглавната страница е без ефекти и предлага кратък увод относно съдържанието на сайта.

• Определяне потребителите на дадения сайт – Сайтът е достъпен

за всички българо- говорещи потребители.

• Интерфейс на сайта - Семпъл, няма вграден специфичен елемент, който да отличи плановете от такива на други сайтове. Липсва меню с препратки към съответно другите страници, даващи повече информация по темата.

• Изчерпателност и организация на информация - информацията

е сравнително пълна, има различни мнения на клиенти, както и информация за специфични основни нива, като метаболитно и психологическо. Има достатъчно снимков материал, който да допълва описанието на плановете.

• Навигация - Навигацията е затруднена. Това, което не е добре направено , че няма меню, от което да отидеш на дадена страница, а трябва от footer-a. Липсва карта на сайта, която да очертава, каква информация е включена.

• Скорост на зареждане - Снимковият материал е добре подбран и направен, но страниците, които съдържат видеа, се зареждат значително по-бавно

• Динамичност на сайта – Потребителя има възможност да даде своя коментар за сайта и да изпрати информация за съществуването на този сайт на свой приятел посредством HTML форма

В интернет пространството съществуват още много сайтове, съдържащи подобна информация за някой хранителен режим, но в тях се описват само малка част от плановете. В повечето случаи темата не е засегната достатъчно задълбочено и подробно. По-голяма част от тези сайтове не са проектирани, така че да улесняват потребителя– навигацията в тях е лоша, за читателя е трудно да открие нужната информация, придвижването в тях е затруднено и e почти невъзможно за потребителя да проследи откъде е минал и как да се върне на мястото, от което е тръгнал.

В сравнение с разгледаните български сайтове, съдържащи информация за хранителните режими, наличните сайтове описващи плановете на други страни предлагат значително по – оскъдни сведения, информацията в тях не е систематизирана и достъпът до нея е затруднен поради лошата навигация.

Вземайки предвид всички тези негативни страни на съществуващите сайтове за хранителните режими и планове, ние се постарахме да създадем по-богат, с добра навигация и илюстриран с много изображения сайт за изготвнето на хранителните планове.

**1.3. Цел на дипломната работа**

Целта на този проект е да се разработи уеб платформа за изготвяне и следене на хранителен режим, където потребителите могат да се свързват с даден треньор от платформата, с когото желаят да работят**.** След регистрация и одобрение на заявката от треньора, потребителите могат да преглеждат изготвения за тях индивидуален режим, както и да следят своя прогрес.

След избора на треньор и одобрението му, ще се направи разговор с клиента, на който се снема пълен антропометричен, здравен и хранителен статус. Също така хранителният режим трябва да е съобразен с неговата възраст, тегло, физическо и здравословно състояние, както и с начина му на живот. На базата на информацията, която получи, се създава специално за клиента персонализиран хранителен режим, който ще отговаря на нуждите на организма му и ще е спрямо неговите вкусови предпочитания. Персонално изработеният хранителен режим включва най-необходими покупки, разнообразни варианти на всяко хранене (най-често между 4 и 6) и междинни хранения (над 5), базирани на предварително попълнен въпросник, в който потребителите уточняват предпочитани храни, както и такива които искат да бъдат избегнати на всяка цена.

Суплементарен план (хранителни добавки), ако имат желание да използват такива.

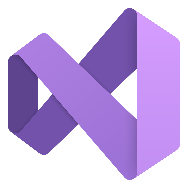
Хранителният режим може да бъде допълнително съобразен с предпочитани брой хранения дневно, работотно време и други индивидуални фактори. Потребителите ще имат възможността да отбелязват всяко хранене.

Химичен състав на храните – кратко описание на всяка една храна, колко калории съдържа. Консултации за всяка седмица от режимите, за да се дискутира всичко необходимо и да се направят нужните промени по режимите, ако такива са необходими – на базата на данните, които се предоставят и това как се чувстват.

**ГЛАВА 2 - ПРОЕКТИРАНЕ НА СТРУКТУРАТА НА САЙТА**

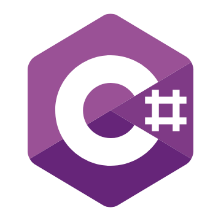
**2.1. Използвани технологии и подготовка на работната среда**

Настоящата система е разработена на база на едни от последнитетехнологии, предлагани от Microsoft.

 **2.1.1. Microsoft Visual Studio** е мощна [интегрирана среда за разработка](https://bg.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B8%D1%80%D0%B0%D0%BD%D0%B0_%D1%81%D1%80%D0%B5%D0%B4%D0%B0_%D0%B7%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0) на софтуерни приложения за [Windows](https://bg.wikipedia.org/wiki/Windows) и за платформата [.NET Framework](https://bg.wikipedia.org/wiki/.NET_Framework). Използва се за разработка на конзолни и [графични потребителски интерфейс](https://bg.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D0%BD_%D0%BF%D0%BE%D1%82%D1%80%D0%B5%D0%B1%D0%B8%D1%82%D0%B5%D0%BB%D1%81%D0%BA%D0%B8_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81) приложения, както и Windows Forms или WPF приложения, [уеб сайтове](https://bg.wikipedia.org/wiki/%D0%A3%D0%B5%D0%B1%D1%81%D0%B0%D0%B9%D1%82), [уеб приложения](https://bg.wikipedia.org/wiki/%D0%A3%D0%B5%D0%B1_%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5) и [уеб услуги](https://bg.wikipedia.org/wiki/%D0%A3%D0%B5%D0%B1_%D1%83%D1%81%D0%BB%D1%83%D0%B3%D0%B0) на всички поддържани платформи от Microsoft Windows, [Windows Mobile](https://bg.wikipedia.org/wiki/Windows_Mobile), Windows CE, .NET Framework, .NET Compact Framework и Microsoft Silverlight.

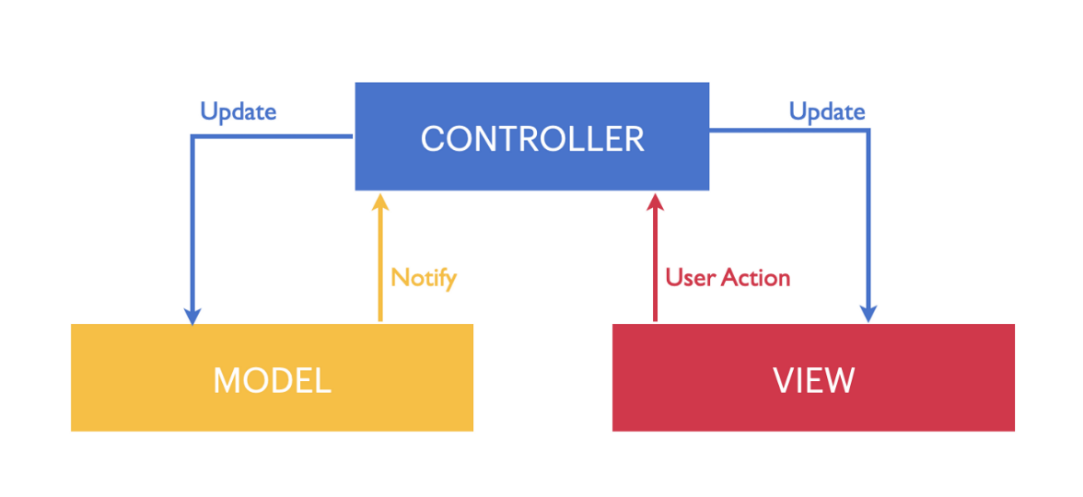
Visual Studio предоставя мощна интегрирана среда за писане на код, компилиране, изпълнение, дебъгване (както за високо така и за машинно ниво), тестване на приложения, дизайн на потребителски интерфейс (форми, диалози, уеб страници, визуални контроли и други), моделиране на данни, моделиране на класове, изпълнение на тестове, пакетиране на приложения и стотици други функции.

Visual Studio поддържа различни [езици за програмиране](https://bg.wikipedia.org/wiki/%D0%95%D0%B7%D0%B8%D1%86%D0%B8_%D0%B7%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%B8%D1%80%D0%B0%D0%BD%D0%B5" \o "Езици за програмиране) (например: [C#](https://bg.wikipedia.org/wiki/C_Sharp), VB.NET, [C](https://bg.wikipedia.org/wiki/C_(%D0%B5%D0%B7%D0%B8%D0%BA_%D0%B7%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%B8%D1%80%D0%B0%D0%BD%D0%B5))/[C++](https://bg.wikipedia.org/wiki/C%2B%2B), [F#](https://bg.wikipedia.org/w/index.php?title=F_Sharp&action=edit&redlink=1), [XML](https://bg.wikipedia.org/wiki/XML)/[XSLT](https://bg.wikipedia.org/wiki/XSLT), [HTML](https://bg.wikipedia.org/wiki/HTML)/[XHTML](https://bg.wikipedia.org/wiki/XHTML), [JavaScript](https://bg.wikipedia.org/wiki/JavaScript), [CSS](https://bg.wikipedia.org/wiki/CSS) и други) и различни технологии за разработка на софтуер (Win32, COM, [ASP.NET](https://bg.wikipedia.org/wiki/ASP.NET), ADO.NET Entity Framework, Windows Forms, WPF, Silverlight и още десетки други Windows и .NET технологии).

** 2.1.2. C#** е обектно ориентиран език за програмиране, разработен от Microsoft като част от софтуерната платформа .NET. Стремежът още при създаването на C# езика е бил да се създаде прост, модерен, обектно ориентиран език с общо предназначение. Основа за C# са C++, Java и донякъде езици като Delphi, VB.NET и C. Той е проектиран да балансира мощност (C++) с възможност за бързо разработване (Visual Basic и Java).

Те представляват съвкупност от дефиниции на класове, които съдържат в себе си методи, а в методите е разположена програмната логика – инструкциите, които компютърът изпълнява. Програмите на C# представляват един или няколко файла с разширение .cs, в които се съдържат дефиниции на класове и други типове. Тези файлове се компилират от компилатора на C# до изпълним код и в резултат се получават асемблита – файлове със същото име, но с различно разширение (.exe или .dll).

**2.1.3. Трислойна архитектура MVC - „Model-view-controller“.**



***Фиг.1-MVC модел***

**Модел-Изглед-Контролер** (Model-View-Controller или MVC) е архитектурен [шаблон за дизайн](https://www.wikiwand.com/bg/%D0%A8%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%B7%D0%B0_%D0%B4%D0%B8%D0%B7%D0%B0%D0%B9%D0%BD" \o "Шаблон за дизайн) в програмирането, основан на разделянето на бизнес логиката от графичния интерфейс и данните в дадено приложение.

* **Модел** – ядрото на приложението, предопределено от областта, за която се разработва; обикновено това са данните от реалния свят, които се моделират и над които се работи – въвеждане, промяна, показване и т.н. Трябва да се прави разлика между реалния обкръжаващ свят и въображаемият абстрактен моделен свят, който е продукт на разума, който се възприема като твърдения, формули, математическа символика, схеми и други помощни средства. Например в банково приложение това са класовете, описващи клиентите, техните сметки, транзакциите, които са осъществили и т.н., както и класовете за извършване на операции над тези обекти (engines) – например клас Transfer с методи като createInterBankTransfer(), createInnerBankTransfer(), getCash() и т.н.
* **Изглед** (англ. View) – е изходящият поток от информация (*това, което приложението изпраща като отговор до дисплея, респективно – до потребителя, в следствие на неговата заявка*). Възможни са няколко различни изгледа на една и съща информация, като например различни диаграми за мениджмънт на даден ресурс или различни таблици. Например изгледът може да се състои от [PHP](https://bg.wikipedia.org/wiki/PHP) шаблонни класове, [JSP](https://bg.wikipedia.org/w/index.php?title=JSP&action=edit&redlink=1) страници, [ASP](https://bg.wikipedia.org/wiki/ASP) страници, [JFrame](https://bg.wikipedia.org/w/index.php?title=JFrame&action=edit&redlink=1" \o "JFrame (страницата не съществува)) наследници в [Swing](https://bg.wikipedia.org/w/index.php?title=Swing&action=edit&redlink=1) приложение. Зависи от това какъв [графичен интерфейс](https://bg.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D0%BD_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81" \o "Графичен интерфейс) се прави и каква платформа се използва
* **Контролер** – тази част от сорс кода (клас или библиотека), която взима данните от модела или извиква допълнителни методи върху модела, предварително обработва данните, и чак след това ги дава на изгледа. Например може да бъде създаден един малък [обект](https://bg.wikipedia.org/wiki/%D0%9E%D0%B1%D0%B5%D0%BA%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%B8%D1%80%D0%B0%D0%BD%D0%B5)" \o "Обект (програмиране)), в който да бъдат сложени данните за транзакцията – като в контролера бъдат взети данните за транзакцията от модела, бъдат преведени датите от [UNIX](https://bg.wikipedia.org/wiki/UNIX) формат в четим от потребителя формат, бъде преобразувана валутата от долари в евро например, бъде закръглено до втория знак вместо да се виждат данните както са в модела (и в базата) до 10-ия. Също така когато се прави уеб графичен интерфейс това би довело до много лесна модификация на [HTML](https://bg.wikipedia.org/wiki/HTML) кода дори от човек, който не е програмист – той ще гледа на шаблона просто като на обикновена HTML страница.

В [софтуерното инженерство](https://bg.wikipedia.org/wiki/%D0%A1%D0%BE%D1%84%D1%82%D1%83%D0%B5%D1%80%D0%BD%D0%BE%D1%82%D0%BE_%D0%B8%D0%BD%D0%B6%D0%B5%D0%BD%D0%B5%D1%80%D1%81%D1%82%D0%B2%D0%BE" \o "Софтуерното инженерство) **многослойната архитектура** (наричана още N-слойна архитектура) е [архитектура](https://bg.wikipedia.org/wiki/%D0%A1%D0%BE%D1%84%D1%82%D1%83%D0%B5%D1%80%D0%BD%D0%B0_%D0%B0%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0" \o "Софтуерна архитектура) от тип [клиент-сървър](https://bg.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B8%D0%B5%D0%BD%D1%82-%D1%81%D1%8A%D1%80%D0%B2%D1%8A%D1%80" \o "Клиент-сървър), в която [интерфейсът](https://bg.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%B8%D1%80%D0%B0%D0%BD%D0%B5)" \o "Интерфейс (програмиране)), обработката на приложения и съхранението и обработката на [данни](https://bg.wikipedia.org/wiki/%D0%94%D0%B0%D0%BD%D0%BD%D0%B8" \o "Данни) са логически разделени на отделни модули. Най-разпространената форма на многослойна архитектура е трислойната архитектура.

Многослойната архитектура осигурява модел, по който разработчиците могат да създават гъвкави приложения, които могат да бъдат използвани многократно. При разделянето на едно приложение на слоеве, разработчиците имат възможността да добавят или променят отделен слой, вместо да преработват цялото приложение.

Всички съвременни бизнес приложения имат достъп до различен вид данни като част от тяхната основна функционалност. След като [релационните бази данни](https://bg.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BB%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B8_%D0%B1%D0%B0%D0%B7%D0%B8_%D0%B4%D0%B0%D0%BD%D0%BD%D0%B8" \o "Релационни бази данни) добиха широка популярност, IT индустрията се преориентира от еднослоен модел (mainframe) към модела **[клиент-сървър](https://bg.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B8%D0%B5%D0%BD%D1%82-%D1%81%D1%8A%D1%80%D0%B2%D1%8A%D1%80" \o "Клиент-сървър)**. При него са налице **клиент**, който включва презентационна логика и голямата част от бизнес логиката на приложението, както и **сървър** включващ в себе си хранилище за данни (data storage) и част от бизнес логиката под формата на готови и съхранени заявки към базата данни.

Трислойната архитектура обикновено се състои от презентационен слой ([потребителски интерфейс](https://bg.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D1%80%D0%B5%D0%B1%D0%B8%D1%82%D0%B5%D0%BB%D1%81%D0%BA%D0%B8_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81" \o "Потребителски интерфейс)) на най-високо ниво в приложението. Той служи за прякото взаимодействие с потребителя и изпращането на заявки към бизнес слоя. Не трябва да има директна връзка между този слой и слоят за [бази данни](https://bg.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B8_%D0%B4%D0%B0%D0%BD%D0%BD%D0%B8" \o "Бази данни). Бизнес слоят служи за обработка на данните и работните процеси. Той комуникира както с презентационния слой, така и с базите данни. Слоят за данни комуникира само с бизнес логиката и служи за съхранение данни и тяхното използване. Често отделните модули се разработват и на различни платформи. Разработена е от Джон Дж. Донован, в неговата компания OEC (Open Environment Corporation), основана в [Кеймбридж](https://bg.wikipedia.org/wiki/%D0%9A%D0%B5%D0%B9%D0%BC%D0%B1%D1%80%D0%B8%D0%B4%D0%B6_(%D0%9C%D0%B0%D1%81%D0%B0%D1%87%D1%83%D0%B7%D0%B5%D1%82%D1%81)" \o "Кеймбридж (Масачузетс)), Масачузетс.

Освен стандартните предимства на модулния [софтуер](https://bg.wikipedia.org/wiki/%D0%A1%D0%BE%D1%84%D1%82%D1%83%D0%B5%D1%80" \o "Софтуер), трислойната архитектура е проектирана да позволява да бъде променен или заменен независимо, който и да е от трите слоя, за да отговаря на промени в изискванията или технологиите. Например, промяна на [операционната система](https://bg.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0" \o "Операционна система) на презентационния слой би засегнала само кода за [потребителският интерфейс](https://bg.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D1%80%D0%B5%D0%B1%D0%B8%D1%82%D0%B5%D0%BB%D1%81%D0%BA%D0%B8_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81" \o "Потребителски интерфейс).

В повечето случай, [потребителският интерфейс](https://bg.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D1%80%D0%B5%D0%B1%D0%B8%D1%82%D0%B5%D0%BB%D1%81%D0%BA%D0%B8_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81" \o "Потребителски интерфейс) работи на десктоп система и използва стандартен графичен интерфейс, функционалната логика за обработка на данните, която може да се състои от един или повече отделни модула, работещи на десктоп система или сървърно приложение, и РСУБД (релационните системи за управление на бази данни) работещи на сървър [база данни](https://bg.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D0%B8" \o "База данни) или [мейнфрейм](https://bg.wikipedia.org/wiki/%D0%9C%D0%B5%D0%B9%D0%BD%D1%84%D1%80%D0%B5%D0%B9%D0%BC" \o "Мейнфрейм), който съдържа логиката за съхранението и обработката на базите данни. Междинният слой може самият да бъде многослоен. В такъв случай цялостната архитектура се нарича N-слойна архитектура.

Трислойната архитектура се състои от следните три слоя:

В областта на уеб разработката трислойната архитектура често се използва в уеб сайтове, най-често т.нар. „електронни магазини“, които се състоят от 3 слоя:

1. Front-end слой (Клиентски слой) – този слой, е съдържанието, което се възпроизвежда от [уеб браузърът](https://bg.wikipedia.org/wiki/%D0%A3%D0%B5%D0%B1_%D0%B1%D1%80%D0%B0%D1%83%D0%B7%D1%8A%D1%80" \o "Уеб браузър) и достига до крайния потребител. То може да бъде генерирано статично или динамично.
2. Среден слой – той реализира логиката на самото уеб приложение. Това се осъществява със [сървър](https://bg.wikipedia.org/wiki/%D0%A1%D1%8A%D1%80%D0%B2%D1%8A%D1%80" \o "Сървър), който обработва и генерира динамично съдържание. Този сървър може да е с платформа един от следните езици – [Ruby on Rails](https://bg.wikipedia.org/wiki/Ruby_on_Rails), Java EE, [ASP.NET](https://bg.wikipedia.org/wiki/ASP.NET), [PHP](https://bg.wikipedia.org/wiki/PHP), ColdFusion, [Perl](https://bg.wikipedia.org/wiki/Perl), Node.js.
3. Back-end слой – той представлява [база-данни](https://bg.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D0%B8" \o "База данни), която се състои, както от самата информация в таблици, така и от система за управлението и (RDBMS).

**2.1.4. ASP.NET Core** е безплатна софтуерна рамка за уеб разработка, с отворен код. Също така тя се явява и следващата стъпка в еволюцията ASP.NET. Тя е разработена съвместно от Microsoft и общността, която е събрала през годините на своето развитие. ASP.NET Core е модуларна софтуерна рамка, която може да върви както на пълната .NET рамка, така и на крос-платформената .NET Core. Въпреки, че е нова софтуерна рамка, изградена върху нов web stack, тя има висока степен на съвместимост с ASP.NET MVC.

Спрямо своите предшественици ASP.NET Core поддържа нова функция – т.нар. „side by side versioning”. При нея различни приложения, които използват една и съща машина, могат да таргетират различни версии на ASP.NET Core, в зависимост от версиите (и нуждите) си. Това не е възможно с по-стари издания на ASP.NET.

**2.1.5. HTML** е основният маркиращ език за описание и дизайн на уеб страници. HTML е стандарт в интернет, а неговите стандарти се определят от международния консорциум W3C. Текущата версия на стандарта е HTML 5.0 (от 28 октомври 2014 г.), а предходната стабилна версия е HTML 4.1.

Описанието на документа става чрез специални елементи, наречени HTML елементи или техните маркери, които се състоят от тагове и съответстващите етикети (HTML tags) и ъглови скоби (като например елемента <html>). HTML елементите са основната градивна единица на кода, който изграждат уеб страниците. Чрез тях се форматира, графично оформя текста и неговите отделните части в рамките на една уеб страница, като например заглавия, цитати, текстови раздели, хипертекстови препратки и т.н. Най-често HTML елементите са групирани по двойки <h1> и </h1>.

В повечето случаи HTML кодът е написан в текстови редактори, с файлов формат .html, .htm, dhtml и се качва и хоства на [сървъри](https://bg.wikipedia.org/wiki/%D0%A1%D1%8A%D1%80%D0%B2%D1%8A%D1%80" \o "Сървър), които са онлайн в интернет или са част от [www](https://bg.wikipedia.org/w/index.php?title=Www&action=edit&redlink=1) мрежата. Тези .html файлове съдържат програмно на таговете на HTML и текстово съдържание със маркери и коментари – също инструкции за [браузъра](https://bg.wikipedia.org/wiki/%D0%91%D1%80%D0%B0%D1%83%D0%B7%D1%8A%D1%80" \o "Браузър), за това какъв точно тип е .html страницата, а също за това как да се показва текстът, особено що се отнася до езиковите характеристики. За да се илюстрира как се включва текст в HTML код: <маркер> Някакъв текст. </край на маркера>. [уеб браузърите](https://bg.wikipedia.org/wiki/%D0%A3%D0%B5%D0%B1_%D0%B1%D1%80%D0%B0%D1%83%D0%B7%D1%8A%D1%80" \o "Уеб браузър) са програмирани от своя страна така, в повечето случаи, макар че някои браузъри могат да имат съответно проблеми на версията, за да могат да прочетат HTML документите и да ги покажат на екрана като уеб страници. Браузърите не показват самите HTML тагове, освен ако не се отиде в менюто за да се направи това, така че те „интерпретират“ (тоест [парсват](https://bg.wikipedia.org/w/index.php?title=Parsing&action=edit&redlink=1" \o "Parsing (страницата не съществува))) съдържанието на страницата като код и текст за да могат след работа на процесора да покажат желаното уеб-съдържание.

Основното предимство на HTML е, че уеб-страниците, които са го включват в кода си, могат да се разглеждат чрез показването им от браузъра на екрана на повечето устройства.

Уебстраницата може да има дизайн, който дори изглежда с добър дизайн с помощта на [CSS](https://bg.wikipedia.org/wiki/CSS) или „правилно оформен“ (например с помощта на [C#](https://bg.wikipedia.org/wiki/C)), както върху монитора на [персоналния компютър](https://bg.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D1%81%D0%BE%D0%BD%D0%B0%D0%BB%D0%B5%D0%BD_%D0%BA%D0%BE%D0%BC%D0%BF%D1%8E%D1%82%D1%8A%D1%80" \o "Персонален компютър), но също и върху миниатюрния дисплей на [пейджър](https://bg.wikipedia.org/w/index.php?title=%D0%9F%D0%B5%D0%B9%D0%B4%D0%B6%D1%8A%D1%80&action=edit&redlink=1" \o "Пейджър (страницата не съществува)) или дисплея на [мобилен телефон](https://bg.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B1%D0%B8%D0%BB%D0%B5%D0%BD_%D1%82%D0%B5%D0%BB%D0%B5%D1%84%D0%BE%D0%BD" \o "Мобилен телефон).

HTML може да прикрепя скриптове писани на езици като [JavaScript](https://bg.wikipedia.org/wiki/JavaScript), който е помощен за HTML, и това променя поведението на дадена уеб страница. [Cascading Style Sheets](https://bg.wikipedia.org/wiki/Cascading_Style_Sheets) (CSS) се използват, като това се прави за да се определя изгледа и оформлението на текста и други включени в страницата изображения и илюстриращи материали. [World Wide Web Consortium](https://bg.wikipedia.org/wiki/World_Wide_Web_Consortium) (W3C) поддържа както HTML, така и CSS, и насърчава използването на CSS в HTML страниците още от 1997. Това допринася за разделяне съдържанието и структурата на уеб страниците от тяхното визуално представяне.

  
 **2.1.5. CSS** е език за програмиране и също описание на уеб дизайн програмни стилове (език за стилови файлове, style sheet language) – използва се основно за описание на онлайн представянето на уеббазиран документ, който написан на език за маркиране. Най-често се използва допълнително към чистия HTML, но се прилага и върху XML уебстраници и документи. Спецификацията на CSS официално се поддържа от W3C.

CSS още в началото на развитието на www започва да се добавя към стандартния HTML с цел да бъдат разделени съдържанието и структурата на уеб страниците отделно от тяхното визуално представяне. Преди стандартите за CSS, установени от W3C през 1995 г., съдържанието на сайтовете и стила на техния дизайн са писани в една и съща HTML страницата. В резултат на това HTML кодът се превръща в сложен и нечетлив, а всяка промяна в проекта на даден сайт изисквала корекцията да бъде нанасяна в целия сайт страница по страница. Използвайки CSS, настройките за форматиране могат да бъдат поставени в един-единствен файл и тогава промяната ще бъде отразена едновременно на всички страници, които използват този CSS файл.

**2.1.6. JavaScrip** представлява обектно-ориентиран скриптов език, който се използва в множество платформи. Той е малък и лек език, който е безполезен,ако се ползва самостоятелно. Проектиран е за лесно вграждане в други продукти и приложения като уеб браузърите. Ядрото на JavaScript съдържа множество от обекти като масиви, дати, математически функции и стандартни елементи като оператори, структури и декларации(statements).

Най-често използваната форма на JavaScript e като език от страна на клиента(client-side). В този случай скриптът се включва в HTML документ или е рефериран в него, за да бъде интерпретиран от браузъра. Потози начин една уеб страница престава да съдържа само статичен HTML код, но може да включва и програмни кодове, които да осъществяват взаимодействие с потребителя, да указват контрол върху браузъра и динамично да генерират HTML код.

JavaScript кода се изпълнява, когато потребител изпраща резултата от попълнена от него форма. Попълнените данни се изпращат до сървъра само в случай, че всички полетана форматаса попълнени коректно (напр. имейл адрес). Този език позволява прихващането на събития като натискане на бутон, пренасочване към друга страница от линкове и други

**Bootstrap -** Това е CSS рамка. Bootstrap е сред най-популярните HTML, CSS и JS рамки. Това е безплатен набор от инструменти за създаване на уебсайтове, който включва типографски елементи, готови уеб форми, бутони, менюта и други уеб елементи. Основният фокус на рамката признава развитието на сложни мобилни проекти. Важно предимство на Bootstrap е голяма общност, която е пет пъти по-голяма от конкурентитеси.

Основната характеристика на Bootstrap е, че тя не е само CSS рамка, но също така и Javascript библиотека. Bootstrap е разработил готови за използване стилове и скриптове, които са свързани с документа чрез писане на необходимите класове и атрибути на HTML елементи.

Bootstrap е разпознат като много полезен инструмент за мобилно оформление, защото мрежата от елементи, разработени в нея, е доста гъвкава, за да покаже сайта на различни диагонали с високо качество.

**2.1.7. База от данни** (БД, още база от данни) представлява колекция от логически свързани данни в конкретна предметна област, които са структурирани по определен начин. В първоначалния смисъл на понятието, използван в компютърната индустрия, базата от данни се състои от записи, подредени систематично, така че компютърна програма да може да извлича информация по зададени критерии.

Поддръжката на база от данни се осъществява от т.нар. [система за управление на бази от данни](https://bg.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D0%B7%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BD%D0%B0_%D0%B1%D0%B0%D0%B7%D0%B8_%D0%BE%D1%82_%D0%B4%D0%B0%D0%BD%D0%BD%D0%B8" \o "Система за управление на бази от данни) (СУБД).

Система за управление на бази данни е компютърно приложение ([софтуер](https://bg.wikipedia.org/wiki/%D0%A1%D0%BE%D1%84%D1%82%D1%83%D0%B5%D1%80" \o "Софтуер)) създадено за комуникация между потребителя, други приложения, както и други БД, с цел да се сравнят и анализират данни. Общото специфично предназначение на СУБД е да позволи определянето, създаването, заявки, актуализацията и администрирането на бази данни. Добре известни СУБД включват MySQL, PostgreSQL, Microsoft SQL Server, Oracle, Sybase, SAP HANA, и IBM DB2. Бази данни не са съвместими с различните СУБД, за това различните СУБД работят със стандартни като SQL и ODBC или JDBC, за да позволи на всяко приложение да работи с различни СУБД, а така и с различни БД. Управлението на БД често се избира от модела им, които те подкрепят. Най-използвани системи от бази данни от 1980 г. насам са всички поддържани релационния модели на езика SQL. Често срещано е СУБД да се нарича само „база данни“.

Съществуват три типа организация на данните и връзките между тях (наричани модели на бази от данни): йерархичен (на [английски](https://bg.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8_%D0%B5%D0%B7%D0%B8%D0%BA" \o "Английски език): *hierarchical model*), мрежови (network model) и [релационен](https://bg.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BB%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%B5%D0%BD_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB" \o ") (relational model), като така се различават йерархични бази от данни, мрежови бази от данни и релационни бази от данни.

**2.2.** Изисквания към софтуерният продукт

**2.2.1.** **Софтуерни изисквания**

**Windows Server 2019** е деветата версия на операционната система Windows Server от Microsoft, като част от семейството операционни системи Windows NT. Това е втората версия на сървърната операционна система, базирана на платформата Windows 10, след Windows Server 2016. Windows Server е операционната система, която свързва локалните среди с услуги на Azure, което разрешава хибридни сценарии и максимизират съществуващите инвестиции, включително:

* Уникални хибридни възможности с Azure за разширяване на вашия център за данни и максимизиране на инвестициите
* Усъвършенствана многослойна защита за подобряване на положението със защитата
* По-бързи иновации за приложения, което позволява на разработчиците и ИТ специалистите да създават нови приложения и да ги модернизират, както и
* Несравнима хиперконвергентна инфраструктура за развиване на вашата инфраструктура на център за данни

**Internet Information Services(IIS)** е пакет от сървърни услуги, включващ уеб сървър, пощенски сървър и сървър за споделяне на файлове. Създаден от Microsoft, той е конкурентът на Apache и sendmail при Unix системите. Най-важния от горепосочените компоненти е уеб сървърът. Той поддържа всички стандарти за уеб като HTTP 1.0/1.1. За разлика от Apache, той може да работи само под Windows. IIS Express, олекотена (4,5–6,6 MB) версия на IIS, се предлага като самостоятелен безплатен сървър и може да се инсталира на Windows XP със Service Pack 3 и следващите версии на Microsoft Windows. IIS 7.5 Express поддържа само HTTP и HTTPS протоколи. Той е преносим, съхранява конфигурацията си за всеки потребител, не изисква административни привилегии и се опитва да избегне конфликт със съществуващи уеб сървъри на същата машина. IIS Express може да бъде изтеглен отделно или като част от WebMatrix или Visual Studio 2012 и по-нови версии. По подразбиране IIS Express обслужва само локален трафик.

 **Microsoft SQL Server** е система за управление на релационни бази данни, разработена от Microsoft. Като сървър на база данни, това е софтуерен продукт с основна функция за съхраняване и извличане на данни, както се изисква от други софтуерни приложения - които могат да работят или на същия компютър, или на друг компютър в мрежа (включително Интернет). Microsoft SQL Server 2019 е предназначено за фирми и организации от всякакъв мащаб – малките, средни и корпоративни предприятия могат да избират измежду различни издания на SQL Server, подходящи за техните нужди.

**SQL Server Management Studio**(**SSMS**) – (Студио за управление на [SQL Server](https://bg.wikipedia.org/wiki/Microsoft_SQL_Server)) е софтуерно приложение, стартирано за първи път с [Microsoft](https://bg.wikipedia.org/wiki/Microsoft) [SQL Server 2005](https://bg.wikipedia.org/w/index.php?title=SQL_Server_2005&action=edit&redlink=1), което се използва за конфигуриране, управление и администриране на всички компоненти в SQL Server. То съчетава широка група от графични инструменти с голям брой текстови редактори, осигуряващи на разработчиците и администраторите всички нива на достъп до сървъра. Водещ елемент в SSMS е [Object Explorer](https://bg.wikipedia.org/w/index.php?title=Object_Explorer&action=edit&redlink=1), който позволява на потребителя да търси, избира и да работи с всеки от обектите на сървъра. Приложението има и „експресна“ версия, която може да бъде изтеглена безплатно.

**Azure Web Apps** е базирана на облачни изчисления платформа за хостване на уебсайтове, създадена и управлявана от Microsoft. Това е платформа като услуга (PaaS), която позволява публикуване на уеб приложения, работещи на множество рамки и написани на различни езици за програмиране (.NET, node.js, PHP, Python и Java), включително собствени на Microsoft и такива на трети страни. Той предлага автоматично мащабиране и висока достъпност, поддържа както Windows, така и Linux и позволява автоматизирани внедрявания от GitHub, Azure DevOps или всяко Git repo.

**Система за имена на (DNS)** представлява [разпределена база от данни](https://bg.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B7%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8_%D0%B1%D0%B0%D0%B7%D0%B8_%D0%BE%D1%82_%D0%B4%D0%B0%D0%BD%D0%BD%D0%B8) за компютри, услуги или други ресурси свързани към [интернет](https://bg.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82) или частни мрежи, с чиято помощ се осъществява преобразуването на [имената](https://bg.wikipedia.org/wiki/%D0%94%D0%BE%D0%BC%D0%B5%D0%B9%D0%BD) на хостовете в [IP адреси](https://bg.wikipedia.org/wiki/IP_%D0%B0%D0%B4%D1%80%D0%B5%D1%81).

**2.2.2. Интерфейсни изисквания**

Графичния интерфейс е състоящ от няколко страници. Страниците могат да бъдат разделени на такива, които са видими за обикновения потребител и такива – видими за администратора. В тази са описани всяка една от тях. Заглавната част на всички страници е с емблемата на сайта за здравословното хранене. Дизайнът на потребителския интерфейс, както и използваните технологии са съобразени с функционалността на приложението. Уеб платформата се състои от публична част, в която са включени:

* ***Начална страница***

Първата страница, която виждат потребителите, е страница с меню на темите, както и тяхното описание. Под менюто може да се види анимация и името на платформата, под която е поставен мотивиращ цитат, както и бутон, отвеждащ към ,,Регистрация”. След това е представена обща информация за идеята на сайта. Обяснена е кратка информация относно това, какво представлява хранителният режим и бутон, водещ към страница ,,За нас”. Също така са представени някои от треньорите и техните професии, също с бутони, водещи към страница ,,Треньори”, където могат да видят по-подробна информация. Накрая на страницата е показан пример с резултатите на клиент, който е пробвал план, изработен спрямо неговите изисквания.

* ***Страница ,,Контакти”-***Когато дадено лице иска да попита нещо, за което няма информация в сайта или не му е достатъчно ясно му е предоставена форма, която трябва да попълни. Формата е с полета за имейл, парола, съобщение(въпроса, който иска да зададе), пол и държава.
* ***Страница ,,За нас”*** -в тази страница е представена подробна информация за екипа. Мисията на екипа, с какво е по-различен от другите и как би могъл да помогне на клиентите. Също така и принципите, които са важни при изпълнението на хранителния план.
* ***Страница ,,Треньори”-*** в тази страница е представена подробна информация за треньорите и техните програми. Какво представляват програмите, тяхната трудност и какво е необходимо за тях.

Тази информация помага на клиента да избере своя план и след като направи избор си е нужно да натисне бутон ,,Свържи се”, който го отвежда към страница ,,Регистрация”.

* **Страница ,,Регистрация”-** Посредством тази страница даден посетител може да се регистрира като потребител на платформата. За целта трябва да предостави определена информация за себе си като попълни формата, която му е дадена. Формата има задължителни полета за попълване (потребителско име, парола, e – mail адрес). При неправилно въведена информация, излиза съобщение което указва къде е грешката. Възможни грешки са: непозволени символи за потребителско име и парола, потребителското име вече заето, неправилен e – mail адрес, първоначално и повторно въведените пароли не съвпадат. При регистрация дадения потребител получава осведомително писмо, което дава информация за потребителското име и парола, на указания от него e – mail адрес при регистрацията.
* ***Страница ,,Вход” -*** Всеки път когато клиента иска да види своя план, той ще трябва да влезе в своя акаунт. За да влезе е необходимо да се попълни форма с полетата имейл и парола. Също така ако е забравил паролата си, клиента може да влезе в линка ,,Забравена парола”. А ако не е регистриран, има линк, който го отвежда в страница ,,Регистрация”.

**Страници достъпни за администратора.**

Всички страници достъпни за обикновения потребител са такива и за администратора. Когато даден посетител влезе в страниците на форума като администратор в долния край на всяка страница е достъпна препратката Администраторски панел. Страницата Администраторски панел дава достъп до разширените администраторски функции.

Администраторът има следните възможности: да търси даден потребител по неговия e – mail и потребителско име, да изтрие даден потребител, да забрани или разреши добавянето на теми от даден IP адрес, да спре e-mail осведомяването до даден адрес.

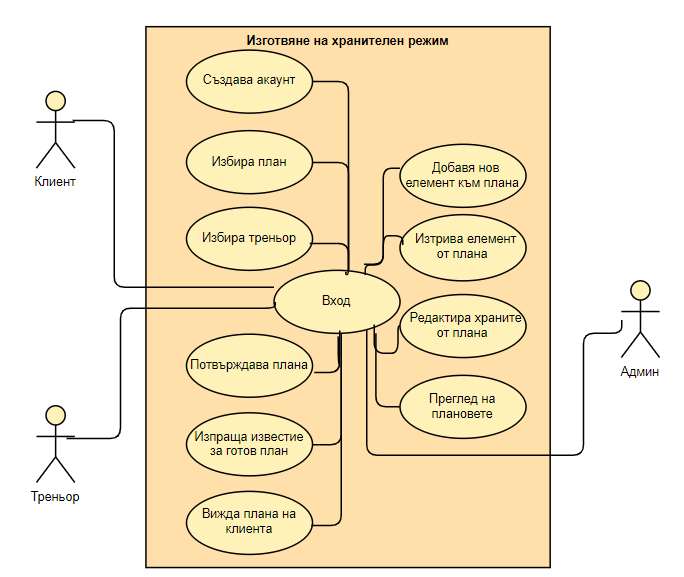
Също така администраторът има достъп до всеки един индивидуален план, като има право да създава, премахва, променя по хранителния план.

**2.3. Изготвяне на Use Cases Diagrams**

В процеса на проектиране на софтуерния продукт, диаграмата на потребителските случаи е първата диаграма, която се създава от проектантите, когато се започне проект. Тази диаграма позволява да се опишат на най-високо ниво целите на потребителя, които системата трябва да изпълнява. Тези цели не е необходимо да са задачи или действия, а може да са по общи изисквания към функционалността на системата. С други думи това е техника за определяне на функционалните изисквания на една система. Те описват типичните взаимодействия между потребителите и системата, предоставят описание на начина, по който тя се използва.

Диаграмите на случаите на употреба (Use-case Diagrams) са основен вид

диаграми за специфициране и документиране на различните потребителски изисквания към една система.



*фиг.2 Use Cases диаграма*

* Изтриване на елементи: Този случай на използване позволява на админа да изтрива елементи от плановете на клиентите.
* Добавяне на елементи: Този случай на използване позволявана админа да добавя елементи към плановете на клиентите.
* Редактиране на храната: Този случай на използване позволява на админа да променя определена храна от плана с друга.
* Преглед на плановете: Този случай на използване позволява на админа да преглежда всеки един индивидуален план

**ГЛАВА 3.** - **ПРОГРАМНА РЕАЛИЗАЦИЯ НА WEB БАЗИРАН ИЗГОТВЯНЕ НА ХРАНИТЕЛЕН РЕЖИМ**

**3.1. Структурата на базата от данни**

Базата данни е храниище за данни, мястото където/от което може да се съхраняват/извличат данни, когато е необходимо. Съществуват много начини за съхранение и извличане на данни. Един от начините, използван в настоящата дипломна работа е използването на SQL заявки.

***Първични и външни ключове*** – за да има достъп до данните, съхранени в таблици, трябва да може да се различава уникално всеки от редовете на таблицт. Затова идентификаторът трябва уникално да определя всички данни в таблицата. В примерен случай за таблица MealPlan се създава допълнителна колонa, която има уникална стойнст за всеки от редовете в таблицата-(MealPlan\_id). Този уникален идентификатор се нарича първичен ключ. Аналогично се определят първичните ключове за останалите таблици.

Външен ключ се наричa колона или няколко колони, които задават връзка между двете таблици. Добавянето на колоната с първите ключове на едната таблица към друга таблица задава връзката между двете таблици(напр.: в таблица Meal първичният ключ е Meal\_id, докато в таблица MealDay този атрибут е с наменование: MealDay\_Meal\_id).

***Нормализация*** – тя е процес на намаляване на излишъка от данни. Представлява разделяне на даннте в две или повече таблици, докато повтарящите се групи от данни се позиционират в две отделни таблици.

В случая постигнатите резултати са:

1. Всички таблици имат уникални идентификатори
2. Таблиците не съдържат повтарящи се стойности или колони
3. Всички таблици съдържат данни от еднотипнти обекти
4. За всички таблици се избягват колони, позволяващи празни стойности

Тези неща за нас ще бъдат направени автоматично от EntityFramework, използвайки CodeFirst подход.

**3.1.1. Code-First** е полезен главно при проектиране, управлявано от домейн. При подхода Code-First се фокусираме върху домейна на нашето приложение и започваме да създаваме класове за обекта на нашия домейн, вместо първо да проектираме нашата база данни и след това да създаваме класовете, които отговарят на дизайна на нашата база данни.

Вместо първо да проектираме таблици на бази данни, започваме със създаването на класове, както и когато е необходимо. Едни от класовете са Потребител и Треньор, където всеки потребител е свързан с един треньор, както е показано по-долу. Това се нарича връзка "едно към много".

public class User:IdentityUser

{

[DisplayName ("First name")]

[MaxLength(25)]

public string FisrtName { get; set; }

[DisplayName("Last name")]

[MaxLength(25)]

public string LastName { get; set; }

public int TrainerInfoId { get; set; }

public TrainerInfo TrainerInfos { get;set; }

}

public class TrainerInfo

{

public int TrainerInfoId { get; set; }

public string Picture { get; set; }

public string Description { get; set; }

ICollection<User> Users { get; set; }

}

Атрибутът MaxLength указва максималната дължина на стойността на данните, разрешена за свойство, което от своя страна задава размера на съответната колона в базата данни.

Атрибутът DisplayName Получава показваното име за свойство, събитие или публичен void метод, който не приема аргументи, съхранени в този атрибут.

Подходът Code-First също изисква контекстен клас, който трябва да бъде извлечен от класа DbContext. Той произлиза от класа DBContext и излага DbSet свойства за типовете, които искаме да бъдат част от модела, напр. Потребители и класове в този случай. DbSet е колекция от класове на обекти (известен още като набор от обекти), така че сме дали името на свойството като множествено число на името на обекта.

public class MealPlanDbContext : IdentityDbContext<User>

{

public MealPlanDbContext(DbContextOptions<MealPlanDbContext> options)

: base(options)

{

}

public DbSet<Meal> Meals { get; set; }

public DbSet<MealDay> MealDays { get; set; }

public DbSet<MealPlan> MealPlans { get; set; }

public DbSet<MealType> MealTypes { get; set; }

public DbSet<PendingRequest> PendingRequests { get; set; }

public DbSet<TrainerInfo> TrainerInfos { get; set; }

public DbSet<TrainerUser> TrainerUsers { get; set; }

}

Конвенциите са набори от правила по подразбиране, които автоматично конфигурират концептуален модел въз основа на нашите класове на домейн, когато работим с подхода Code-First. EF API конфигурира PrimaryKeys, ForeignKeys, релации, типове данни на колони и т.н. от класовете на домейна без никакви допълнителни конфигурации. Това се дължи на конвенциите за първи код на EF. Ако те се следват в класовете на домейна, тогава схемата на базата данни ще бъде конфигурирана въз основа на конвенциите. Тези EF 6.x Code-First конвенции са дефинирани в пространството от имена System.Data.Entity.ModelConfiguration.Conventions.

**3.1.2. Изпълнение на миграциите**

За да създам базата данни, използвах функцията EF Core Migrations. Миграциите са набор от инструменти, които създават и актуализират база данни, за да съответстват на модела на данни.

От менюто Инструменти избрах NuGet Package Manager > Package Manager Console.

В конзолата за управление на пакети (PMC) въведох следните команди:

* Add-Migration InitialCreate: Генерира миграционен файл Migrations/{timestamp}\_InitialCreate.cs. Аргументът InitialCreate е името на миграцията. Може да се използва всяко име, но по конвенция се избира име, което описва миграцията. Тъй като това е първата миграция, генерираният клас съдържа код за създаване на схемата на базата данни. Схемата на базата данни е базирана на модела, определен в класа MvcMovieContext.
* Update-Database: Актуализира базата данни до най-новата миграция, създадена от предишната команда. Тази команда изпълнява метода Up във файла Migrations/{time-stamp}\_InitialCreate.cs, който създава базата данни.

С EF Core достъпът до данни се извършва с помощта на модел. Моделът се състои от класове обекти и контекстен обект, който представлява сесия с базата данни. Контекстният обект позволява заявки и записване на данни. Контекстът на базата данни е извлечен от Microsoft.EntityFrameworkCore.DbContext и указва обектите, които да бъдат включени в модела на данни.

Scaffolding създава контекстния клас на база данни Data/ MealPlanDbContext.cs, показан по-горе.

ASP.NET Core е изграден с инжектиране на зависимости (DI). Услуги, като например контекста на базата данни, са регистрирани с DI в Program.cs. Тези услуги се предоставят на компоненти, които ги изискват чрез параметри на конструктора.

Във файла Controllers/MealPlansController.cs конструкторът използва Dependency Injection, за да инжектира контекста на базата данни MvcMealPlansContext в контролера. Контекстът на базата данни се използва във всеки от CRUD методите в контролера.

Scaffolding генерира следния подчертан код в Program.cs:

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddDbContext<MealPlanDbContext>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString("MvcDietContext")));

Конструкторът използва Dependency Injection, за да инжектира контекста на базата данни (MvcMovieContext) в контролера. Контекстът на базата данни се използва във всеки от CRUD методите в контролера.

public class MealPlansController : Controller

{

private readonly MealPlanDbContext \_context;

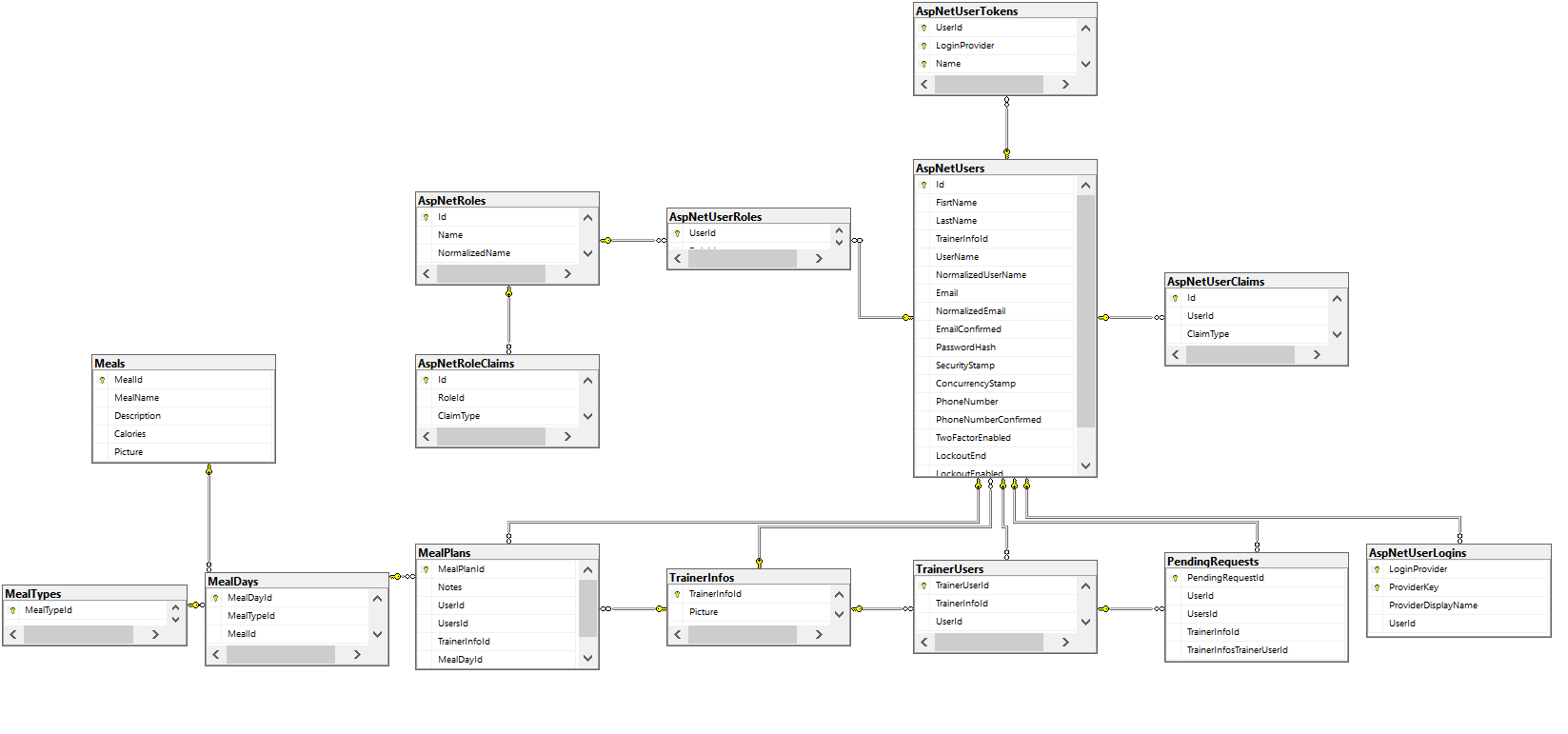
public MealPlansController(MealPlanDbContext context)

{

\_context = context;

}

***3.1.3.*** ***Дизайн на базата данни***

 След прилагането на всички концепции описани дотук, се получава структурата на базата данни във вида, показан на фигурата:

*фиг.3 Диаграма на базата от данни*

Тук ясно се виждат и генерираните от ASP.NET Identity колони втаблицата AspNetUsers(пр. PasswordHash), както и таблици (пр. AspNetUserRoles). Може лесно да се проследят релациите между таблиците заедно с типа им, както и кои полета са първичните ключове на таблиците. По време на разработката физическото местоположение на базата данни се намира в App\_Data папката на PTO\_System.Web проекта на приложението.

* Таблица AspNetUsers – съдържа информация за потребителите.
* Таблицата Meals – съдържа информация за храната.
* Таблицата MealTypes – определя типа (закуска, обед, вечеря).
* Таблицата MealDays – съдържа информация за всеки ден от седмицата.
* Таблицата MealPlans – съдържа обща информация от таблици Meal, MealTypes и MealDays
* Таблицата TrainerInfos – съдържа информация за всеки треньор.
* Таблицата TrainerUsers – съдържа информация за клиентите на треньорите.
* Таблицата PendingRequests – съдържа информация за заявките на потребителите

**Връзка 1:1(One to One)**– Един обект от едната релация ще бъде асоцииран с точно един обект от другата и обратно. Например един шеф е шеф само на един отдел, а един отдел има точно един шеф от таблицата с шефове на отдели.

**Връзка 1:М(One to Many)** – Един обект от лявата релация може да бъде асоцииран с повече от един обекти от дясната релация. Един обект от дясната се асоциира с точно един обект от лявата. Например един човек притежава много коли. Една кола се притежава от един човек.

**Връзка М:1(Many to One)**– Повече от един обекти от лявата релация могат да бъдат асоциирани с един обект от дясната релация. Един обект от дясната се асоциира с повече от един обект от лявата релация. Например един работник се управлява от един мениджър. Един мениджър управлява много работници.

**Връзка М:М(Many to Many)** – Повече от един обекти от лявата се асоциират с повече от един обекти от дясната таблица и обратно. Например един лекар може да лекува много пациенти. Един пациент може да бъде лекуван от много лекари.

**3.2. Описание на CRUD операциите**

CRUD функциите обхващат всички основни функции, които трябва да са имплементирани в приложенията, базирани на релационна база данни. Всяка една от буквите на акронима е свързана със Structured Query Language (SQL) изявление - "C" с "Insert", "R" със "Select", "U" с "Update" и "D" с "Delete". Съществуват вариации на модела, но всяка една от тях включва тези четири основни функционалности под една или друга форма. Такива вариации са:

-**Create операция** - прави това, което подсказва името. Това означава създаване на запис в базата данни. Този запис може да бъде акаунт, потребителска информация, публикация или задача.

// GET: MealPlans/Create

public IActionResult Create()

{

ViewData["MealDayId"] = new SelectList(\_context.MealDays,"MealDayId", "MealDayId");

ViewData["TrainerInfoId"] = new SelectList(\_context.TrainerInfos, "TrainerInfoId", "TrainerInfoId");

return View();

}

// POST: MealPlans/Create

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult>

Create([Bind("MealPlanId,Notes,UserId,TrainerInfoId,MealDayId")] MealPlan mealPlan)

{

if (ModelState.IsValid)

{

\_context.Add(mealPlan);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

ViewData["MealDayId"] = new SelectList(\_context.MealDays, "MealDayId", "MealDayId", mealPlan.MealDayId);

ViewData["TrainerInfoId"] = new SelectList(\_context.TrainerInfos, "TrainerInfoId", "TrainerInfoId", mealPlan.TrainerInfoId);

return View(mealPlan);

}

-**Edit операция** - операцията READ означава получаване на достъп до входовете или записите в потребителския интерфейс. Тоест, виждайки го. Отново записът може да бъде всичко - от потребителска информация до публикации в социални медии и други.

Този достъп може да означава, че потребителят получава достъп до създадените записи веднага след създаването им или след търсенето им. Търсенето е реализирано, за да позволи на потребителя да филтрира записите, от които не се нуждае.

// GET: MealPlans/Edit/5

public async Task<IActionResult> Edit(int? id) {

if (id == null || \_context.MealPlans == null)

{

return NotFound(); }

var mealPlan = await \_context.MealPlans.FindAsync(id);

if (mealPlan == null) {

return NotFound(); }

ViewData["MealDayId"] = new SelectList(\_context.MealDays, "MealDayId", "MealDayId", mealPlan.MealDayId);

ViewData["TrainerInfoId"] = new SelectList(\_context.TrainerInfos, "TrainerInfoId", "TrainerInfoId", mealPlan.TrainerInfoId);

return View(mealPlan);

}

-**Delete операция** - Изтриването означава отърваване от запис от потребителския интерфейс и базата данни.

// GET: MealPlans/Delete/5

public async Task<IActionResult> Delete(int? id) {

if (id == null || \_context.MealPlans == null) {

return NotFound();

}

var mealPlan = await \_context.MealPlans

.Include(m => m.MealDays)

.Include(m => m.TrainerInfos)

.FirstOrDefaultAsync(m => m.MealPlanId == id);

if (mealPlan == null) {

return NotFound();

}

return View(mealPlan);

}

-**Update операция** - UPDATE е операцията, която ви позволява да променяте съществуващи данни. Тоест редактиране на данните.

За разлика от READ, операцията UPDATE променя съществуващите данни, като прави промени в тях.

//GET: MealPlans/Details/5

public async Task<IActionResult> Details(int? id)

{

if (id == null || \_context.MealPlans == null) {

return NotFound(); }

var mealPlan = await \_context.MealPlans

.Include(m => m.MealDays)

.Include(m => m.TrainerInfos)

.FirstOrDefaultAsync(m => m.MealPlanId == id);

if (mealPlan == null) {

return NotFound();}

return View(mealPlan); }

**3.2.1. Как работи шаблона на Contoller с GET и POST заявки**

***-GET метод*** - Методът GET се използва за извличане на информация от даден сървър с помощта на даден URI. Заявките, използващи GET, трябва да извличат само данни и не трябва да имат друг ефект върху данните. Същото като GET, но прехвърля само реда на състоянието и заглавната секция.

Всеки публичен метод в контролер може да се извика като HTTP крайна точка. В примерите по-горе и всички GET методи връщат низ.

HTTP крайна точка:

* Е целеви URL адрес в уеб приложението, като например https:// https://localhost:7050//MealPlan.
* Комбинира:
* Използван протокол: HTTPS.
* Мрежовото местоположение на уеб сървъра, включително TCP порта: localhost:7050.
* Целевият URI: MealPlan.

При коментар GET: /MeaPlan/ гласи, че това е HTTP GET метод, който се извиква чрез добавяне на /MeaPlan/ към основния URL адрес.

При коментар /MealPlan/Create/ указва HTTP GET метод, който се извиква чрез добавяне на /MealPlan/Create/ към URL адреса.

MVC извиква класовете на контролера и методите за действие в тях, в зависимост от входящия URL адрес. Логиката за маршрутизиране на URL по подразбиране, използвана от MVC, използва формат като този, за да определи какъв код да се извика:

/[Контролер]/[Име на действие]/[Параметри]

Форматът за маршрутизиране е зададен във файла Program.cs.

app.MapControllerRoute(

name: "default",

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

app.MapRazorPages();

***-POST*** е HTTP метод, предназначен да изпраща данни към сървъра от HTTP клиент. Методът HTTP POST изисква от уеб сървъра да приеме данните, включени в тялото на POST съобщението. Методът HTTP POST често се използва при изпращане на формуляри за вход или контакт или качване на файлове и изображения на сървъра.  
От горните примери:

* Третият URL сегмент съответства на id на параметъра на маршрута.
* Методът Welcome съдържа идентификатор на параметър, който съответства на URL шаблона в метода MapControllerRoute.
* Проследяването ? (в id?) показва, че параметърът id е незадължителен.

Напр. при метод Edit aтрибутът HttpPost указва, че този метод за редактиране може да бъде извикан само за POST заявки. Може да се приложи атрибута [HttpGet] към първия метод за редактиране, но това не е необходимо, тъй като [HttpGet] е по подразбиране.

Атрибутът [Bind] е един от начините за защита срещу прекомерно публикуване. Трябва да се включат само свойства в атрибута [Bind], които се иска да се променят.

**3.3. Работа с View моделите**

В този раздел модифициме класа MealPlansController, за да използваме файлове за преглед на Razor. Това ясно капсулира процеса на генериране на HTML отговори на клиент.

Шаблоните за преглед се създават с помощта на Razor. Razor - базирани на шаблони за изгледи:

* Има файлово разширение .cshtml.
* Осигурен е елегантен начин за създаване на HTML изход с C#.

В момента методът Index връща низ със съобщение в класа на контролера.

public IActionResult Index()

{

return View();

}

Предходният код:

* Извиква метода View на контролера.
* Използва шаблон за изглед за генериране на HTML отговор.

Методи на контролера:

* Наричат се методи на действие. Например методът за действие Index в предходния код.
* Обикновено връща IActionResult или клас, извлечен от ActionResult, а не тип като низ.

Оформлението на менюто е реализирано във файла Views/Shared/\_Layout.cshtml.

Шаблоните за оформление позволяват:

* Указване на оформлението на HTML контейнера на сайт на едно място.
* Прилагане на оформлението на HTML контейнер в множество страници в сайта.

<div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">

<ul class="navbar-nav flex-grow-1">

<li class="nav-item"> <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">Начало</a></li>

<li class="nav-item"><a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Contact">Контакти</a></li>

<li class="nav-item"><a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="About">За нас</a></li>

<li class="nav-item"><a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Trainers">Треньори</a></li>

<li class="nav-item"><a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="MyAccount">Моят план</a></li>

</ul>

<partial name="\_LoginPartial" />

</div>

Действията на контролера се извикват в отговор на входяща URL заявка. Клас на контролер е мястото, където е написан кодът, който обработва входящите заявки на браузъра. Контролерът извлича данни от източник на данни и решава какъв тип отговор да изпрати обратно към браузъра. Шаблоните за преглед могат да се използват от контролер за генериране и форматиране на HTML отговор към браузъра.

Контролерите са отговорни за предоставянето на данните, необходими, за да може шаблонът на изглед да изведе отговор.

Преглед на шаблони не трябва:

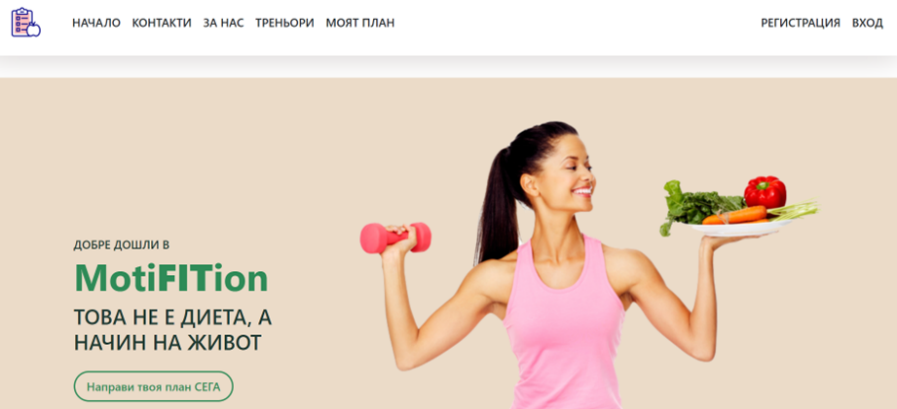
* Да прави бизнес логика
* Да взаимодейста директно с база данни.

Шаблонът за изглед трябва да работи само с данните, които са му предоставени от контролера.

Речникът ViewData е динамичен обект, което означава, че може да се използва всеки тип. Обектът ViewData няма дефинирани свойства, докато не се добави нещо. Системата за обвързване на MVC модел автоматично картографира именуваните параметри name и numTimes от низа на заявката към параметрите в метода.

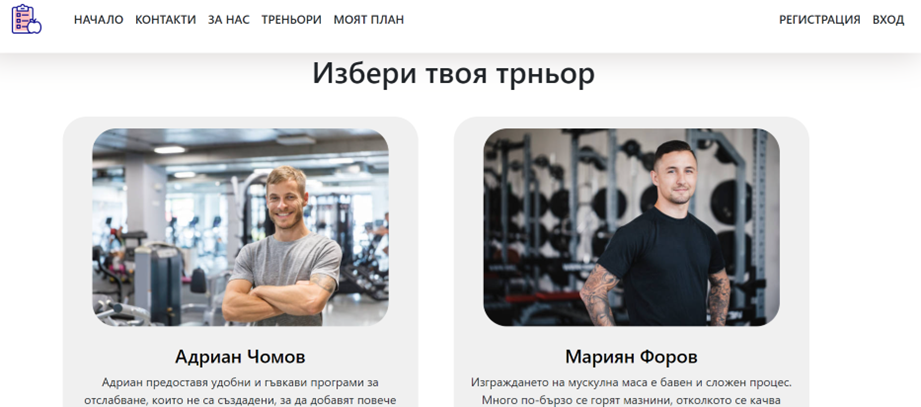
**3.3.1. Описание на сайта (едни от най-важните страници)**

При стартиране на сайта от потребителя, се зарежда началната страница index с кратко описание на същността на уеб приложението.От тази страница клиента може директно да получи своя план, натискайки бутона ,,Направи своя план СЕГА”, който преди това ще го отведе към ,,Регистрация”



*фиг.4 Начална страница*

* **Страница ,,Треньори” –** тук клиента има възможността да избере своя треньор и хранителен режим, като за целта е предоставена подробна информация за това как може да помогне всеки един от треньорите. След избора си, клиента ще трябва да се регистрира, за да се изготви неговия план.



*фиг.5 Страница ,,Треньори”*

**ГЛАВА 4.** – **АНАЛИЗ НА ПОЛУЧЕНИТЕ РЕЗУЛТАТИ**

Разработената дипломна работа на тема: “ Разработка на уеб платформа за изготвяне и следене на хранителен режим” представлява цялостен и завършен проект, отразяващ процеса на анализ и проектиране за изготвяне на хранителен режим на уеб сайт и всички особености, свързани с него. По време на разработването този проект премина през два етапа.

Първият етап представлява анализ на същността, характеристиките и архитектурата на изготвянето на хранителен режим и определяне на основните изисквания, на които трябва да отговаря един такъв план, в зависимост от индивидуалните цели на клиента, за когото е предназначен. Този анализ доказва все по-голямото интегриране на Интернет и WEB във все повече сфери от реалния живот и бизнеса, както и ползата от използването му за различни приложения. Чрез излагането на тази информация се постига изясняването на възможностите за изготвянето на хранителните режими, техните особености и използваните понятия. Това от своя страна спомага за правилното разбиране на същността на хранителния режим, който е обект на проектиране и разработка във втората част на настоящата дипломна работа.

Вторият етап представлява проектирането на изготвянето на хранителен режим. Разгледани са средствата за нейното разработване, концептуалният модел на системата и някои концепции за нейната сигурност. Проектирането на базата данни представя структурата и “гръбнака” на системата. Системата е изградена на модулен принцип, което позволява разширяване на нейната функционалност и добавяне на нови модули, в зависимост от потребностите, които могат да възникнат в последствие, като например добавяне на допълнителен език на интерфейса и съдържанието, търсачка, анкети, календар и други.

С този задълбочен анализ за изготвянето на хранителен режим се постига максимална информираност относно тяхната работа и ползите от прилагането им. Доказва се, че по този начин се предоставят услуги, даващи възможност за лесно изпълнение на плановете, за разширяването на сферата на предлаганите услуги и повишаване на нейната конкурентоспособност, гарантиране на сигурността на данните и актуалността на информацията, така необходими и важни за потребителите.

**ЗАКЛЮЧЕНИЕ**

Дипломната работа представлява вид информационна система, обслужваща обикновени потребители и администратори. WEB базираната система е реално функционираща и покрива нуждите на потребителите. Информацията е отсята внимателно, поднесена е чрез разнообразни примери като се акцентира на най – важните моменти от процеса на обучението по MVC. Разбира се в приложението има какво още да се доусъвършенства, с напредването на науката и техниката ще се появи нуждата от по-голяма интерактивност , по – голям достъп до учебния материал, и т. н. Смятам, че материята на сайта ще благоприятства подобни нововъведения - една от основните ми идеи е да бъде отворен за подобни проекти. Да позволява да бъде променян в бъдеще. Да е достъпен за новостите, а не да ги възпрепятства.

**ИЗПОЛЗВАНА ЛИТЕРАТУРА**

**ASP.NET:източник:** [**https://bg.wikipedia.org/wiki/ASP.NET**](https://bg.wikipedia.org/wiki/ASP.NET)

**C#:източник:** <https://bg.wikipedia.org/wiki/C_Sharp>

**CRUD:източник:** <https://softuni.bg/blog/what-is-crud-mssql>

**HTML:източник:** <https://bg.wikipedia.org/wiki/HTML>

**CSS:източник:** <https://bg.wikipedia.org/wiki/CSS>

**JavaScrip:източник:** [**https://bg.wikipedia.org/wiki/JavaScript**](https://bg.wikipedia.org/wiki/JavaScript)

**Bootstrap:източник:** [**https://bg.wikipedia.org/wiki/Bootstrap**](https://bg.wikipedia.org/wiki/Bootstrap)