**ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - ВАРНА**

Катедра „Софтуерни и интернет технологии“

ДИПЛОМНА РАБОТА

**Тема:** Проектиране и разработка на уеб система за управление на фитнес верига

**Студент**: *Славян Христов Христов*

**Специалност***: Софтуерни и интернет технологии*

**Факултетен номер***: 18621793*

**Ръководител**: *ас. Антоанета Иванова*

**СЪДЪРЖАНИЕ**

[**1.** **УВОД** 3](#_Toc106097057)

[**1.1.** **Избрана тема** 3](#_Toc106097058)

[**2.** **Анализ на проблема** 5](#_Toc106097059)

[**2.1. Функционални изисквания** 5](#_Toc106097060)

[**2.2. Архитектурен шаблон** 7](#_Toc106097061)

[**2.3. Нефункционални изисквания** 10](#_Toc106097062)

[**2.3.1. Технологични средства за реализацията** 11](#_Toc106097063)

[**2.4. Структура и проектиране на системата** 17](#_Toc106097064)

# **УВОД**

В нашето съвремие уеб базираният софтуер се определя като неизменима част зад всеки бизнес, който се стреми да бъде успешен. Една от основните причини за това е наблюдаваното повишение в обвързаността към днешните или модерните информационни и комуникационни технологии. Всекидневието ни през последното десетилетие налага по-забързан живот, изискващ постоянна и свободна достъпност до всякакъв вид информация. Разпространителите на тази информация от своя страна, като бизнеси, компании, корпорации и т.н., имат нуждата да управляват и споделят своите услуги по възможно най-гъвкавия начин, а тази гъвкавост може да се постигне чрез осигуряването на мобилност и платформена независимост. Интернет средата може да се определи като отлично и евтино решение за тези две изисквания. Чрез достъп до интернет се получава цял набор от различни бизнес предимства и едно от тях е именно получаване на възможността да се разработи уеб приложение или уебсайт задоволяващ изискванията на даден бизнес. Уеб базираните приложения притежават множество позитиви над обикновените настолни приложения, които се инсталират локално на машината, а едно от тях, което ги прави изключително привлекателни, е, че са удобни. Нужна е само интернет връзка и клиент(браузър) за да се постигне отдалеченият достъп до желания ресурс.

## **Избрана тема**

Избраната тематика на уеб базираната апликация е свързана с управление на бизнес или по-обширно казано менажиране на фитнес верига, имаща за цел да предлага услуги, които се стремят да осигурят физически активен и здравословен начин на живот за своите клиенти.

С течение на времето се наблюдава въвеждане на все по-заседнало ежедневие. Повечето процеси в средностатистическото работно място за автоматизирани, хората прекарват все повече време в среди, които не само ограничават физическата активност, но и изискват продължително седене на едно място. Работните обекти, училища, домове и обществени пространства вече се конструират по такива начини, че да минимизират драстично движението на човека и мускулната активност. В резултат на това се наблюдава бум на посещенията във фитнес центровете и зали. Хората започват да проявяват все по-голяма загриженост за своето здравословно състояние и следствие на това се разглежда висока заинтересованост във фитнес индустрията, която включва към себе си различни бизнеси, които се възползват от този факт. Това пък води и до по-голям интерес за имплементация на софтуерно решение, което да улесни управлението на такъв вид бизнес.

Този проект има за цел да създаде оптимална софтуерна резолюция на често срещаните бизнес нужди на компания, която се намира в тази сфера.

Измислената фитнес верига наречена *„Fit-M”*(играеща роля на модел за повечето фитнес компании) притежава фитнес зали на различни локации като всяка зала има нает мениджър, който отговаря за управлението на зададените му зали и също за служителите на дадената фитнес зала.

За да може клиент да използва предоставените му услуги, той трябва първо да се регистрира и след това да заплати определена такса за членство, която му дава право да използва избрана фитнес зала и също допълнителните фитнес услуги.

Разработеният софтуер позволява също управление на цялата административна работа върху бизнеса, което го прави широко достъпен от единично, централизирано пространство към своите действия и операции.

# **Анализ на проблема**

## **2.1. Функционални изисквания**

Функционалността на системата трябва да бъде съобразена с нуждите на потребителите, които ще я използват. Интерфейсът на системата трябва да бъде прост, приятен и да позволява лесно и безпрепятствено използване, без грешки в процеса на работа.

Потребителите, които ще използват системата са три вида: администратор, мениджър, фитнес инструктор и клиент. Те трябва да могат да изпълняват следните действия:

1. Администратор

* Влизане в системата
* Добавяне на фитнес зали
* Промяна на параметри на фитнес зали(редактиране)
* Премахване на фитнес зали
* Назначаване на управители
* Извършване на различни видове справки
* Получаване на различни видове известия

1. Управител

* Влизане в системата
* Назначаване на служители към фитнес залите, които менажира.
* Преглед на служители (включва сортиране, филтриране)
* Редактиране на служителите (напр. заплата, работно време и т.н.)
* Достъп до справки за всички зали, за които отговаря.
* Получаване на различни видове известия
* Редактиране на собствения си профил(лимитирани права)

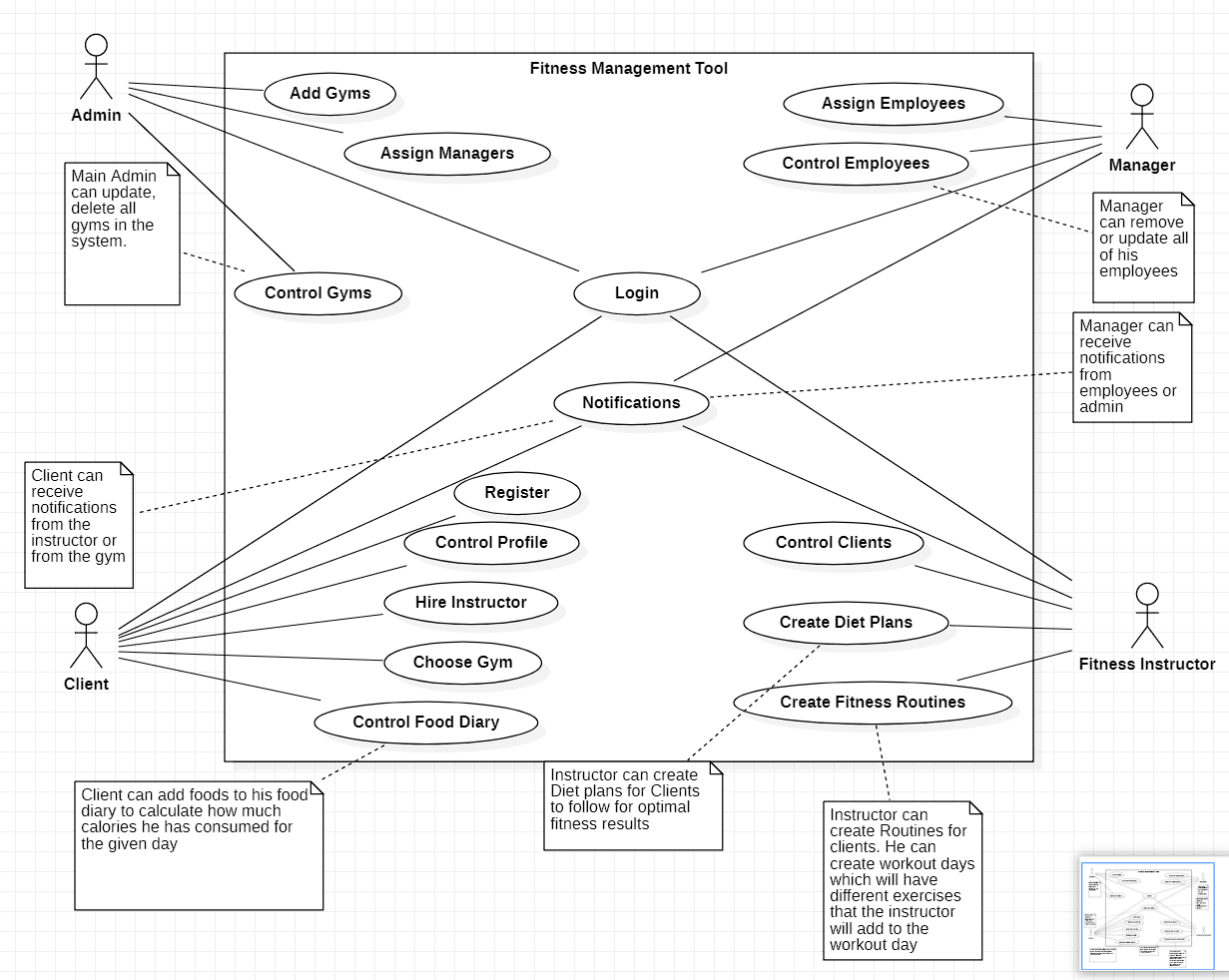
1. Фитнес инструктор

* Влизане в системата
* Изготвяне на тренировъчни програми за свои клиент/и (съдържащи различни физически упражнения)
* Изготвяне на хранителни режими
* Управление на своите клиенти
* Получаване на различни видове известия
* Преглед на клиенти (сортиране + филтриране)
* Притежание на рейтинг
* Достъп до справки, които известяват инструктора за прогреса на своите клиенти
* Редактиране на собствения си профил(лимитирани права)

1. Клиент

* Регистрация в уеб системата
* Влизане в системата
* Избор на цели като качване на мускулна маса, сваляне на килограми, повишаване на физическата издръжливост и т.н.
* Създаване на членство за фитнес зала, като всяко членство има различни месечни такси, инструктори и допълнителни екстри
* Избор на фитнес инструктор
* Получаване на известия при излизане на ексклузивни оферти, неплатена месечна такса, различни съобщения от фитнес инструктора и т.н.

Системата предоставя възможност за изпълнение на всички действия описани горе. Допълнителни функционалности също са имплементирани за валидиране на данните при регистрация или при актуализация на обект/профил. Автентикацията и оторизацията са също изисквания, които са спазени.



***Фигура 1***: Това е Use case диаграма, която показва по опростен начин какви функционалности съдържа приложението и кой ги извършва.

## **2.2. Архитектурен шаблон**

Архитектурният шаблон е планът или чертежът за изграждане на софтуер. Той показва цялостната структура на софтуера, колекцията от компоненти в него и как те взаимодействат един с друг, докато стоят зад реализацията.

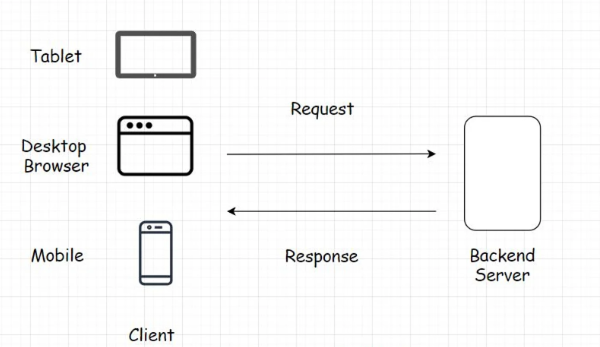
Използваните шаблони са няколко(поради нивото на абстракция в разработката), но като основните могат да се определят:

* Клиент-сървър (Client-server)
* Модел-Изглед-Контролер (Model-View-Controller, MVC)

**Клиент-сървър архитектурен шаблон**

Тъй като това е уеб базиран софтуер и е изграден чрез принципа на RESTful*(набор от правила използвани за създаването на уеб услуги, позволяващи на системите за заявки да достъпват и манипулират уеб ресурси)*, връзката между клиент и сървър е неизбежна.

Архитектурата работи по модела заявка-отговор. Клиентът изпраща заявката до сървъра, и според тази заявка се връща подходящия отговор.

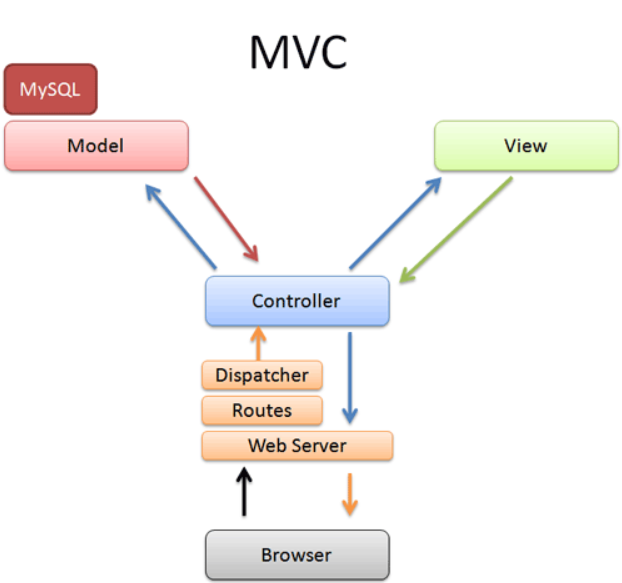


***Фигура 2***: Базова диаграма илюстрираща взаимодействието, между сървър и клиент

Когато използваме интернет браузър за достъп до която и да е уеб страница, ресурс, услуга клиентът е компютърът, работещ с браузърския софтуер, който е поискал уеб ресурс от уеб сървър. Тази комуникация се осъществява чрез строго написани правила или протокол, който дефинира начина на извършване на това комуникиране. Общо приетият протокол е HTTP протокола за заявка и отговор.

**Модел-Изглед-Контролер архитектурен шаблон**

Това е архитектурен модел, който разделя приложението на три основни логически компонента: Модел, Изглед, Контролер. Оттук и идва съкращението MVC (Model-View-Controller). Всеки компонент на архитектурата е създаден, за да се справи със специфичен аспект при разработено софтуерно приложение. MVC разделя слоя на бизнес логиката и слоя на презентацията един от друг. Традиционно се използва при изработването на настолни графични интерфейси (GUI), но в днешно време MVC архитектурата намира приложение и в проектирането на уеб базирани апликации, както и мобилни приложения.



***Фигура 3***: Диаграма показваща компонентите и взаимодействията в MVC архитектура

Изглед (View) – Това е визуалното представяне на данните, които са получени от контролера.

Контролер (Controller) – Той свързва модела и изгледа. Играе ролята на интерфейс между тези два компонента. Контролера реагира на събития от клиента и извършва взаимодействия с модела или изгледа на база на получения вход.

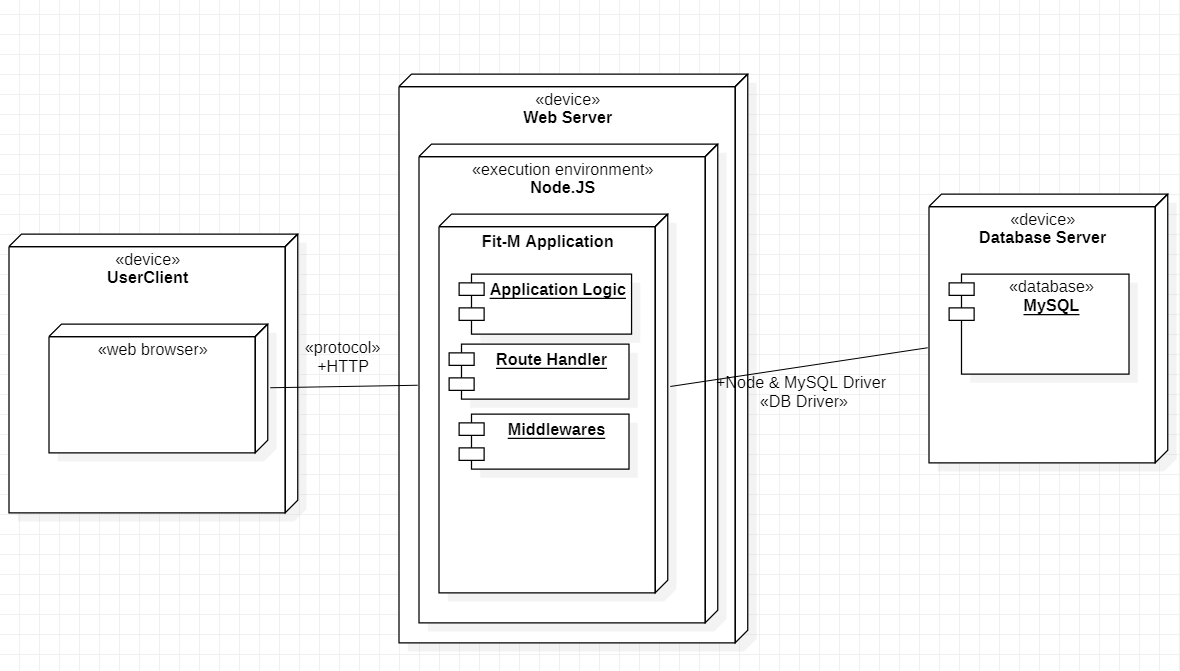
Модел (Model) – Съответства на цялата логика свързана с данните на апликацията. Основната характеристика на модела е да опише бизнес логиката на компонент в системата. Характерното при MVC е, че модела има пряка връзка с базата данни за разлика от изгледа и контролера, чрез контролера се извършват всякакви преки операции към базата от данни.

MVC е решение, което спомага за изработката на по-разбираем софтуер, чрез включване на сепарация между основните абстракции в една съвременна апликация.

## **2.3. Нефункционални изисквания**

Докато функционалните изисквания описват какво точно системата трябва да съдържа в себе си, нефункционалните изисквания пък уточняват как и по какъв начин ще се реализират.

За да се разбере как работи уеб системата в технологичен аспект, трябва да се представят използваните технологични средства и как си взаимодействат. Това най-лесно може да се представи визуално чрез диаграма за внедряване (Deployment Diagram)



***Фигура 4***: Диаграма на внедряване описваща архитектурата на изпълнение.

## **2.3.1. Технологични средства за реализацията**

## 

**А)** Използван език за програмиране: **JavaScript**

**Б)** Среда за изпълнение: **Node JS**

Node e крос платформена среда за изпълнение с отворен код за разработване на сървърни и мрежови приложения. Приложенията на Node JS са написани на езика JavaScript и могат да се изпълняват в средата на Node, което позволява употребата на тази технология в различни операционни системи.

Node.js също предоставя богата библиотека от различни JavaScript модули, което опростява разработването на уеб приложения, използващи Node до голяма степен. Node е с най-голямата екосистема от библиотеки сравнение с конкуренцията. Подобни среди за разработване на сървърни и мрежови приложения са: ASP.NET, Django, Ruby on Rails и други.

Едно от големите предимства на Node е, че позволява на JavaScript разработчиците да пишат код от страна на сървъра в допълнение на кода от страна на клиента, без да е необходимо да научават напълно различен език

**В)** Софтуерна рамка за изграждане на backend на уеб приложение : **Express JS**

Express JS e framework (софтуерна рамка) изграден на Node JS, който улеснява работата ни с Node. Express e предназначен за изграждане на уеб приложения и API-та и е определен, като най-популярната сървърно ориентирана софтуерна рамка. Чрез използване на софтуерна рамка като Еxpress, разработчиците могат да спестят време и да се съсредоточат върху други важни задачи, които касаят самия проект.

**Г)** Система за управление на база от данни: **MySQL**

MySQL е система за управление на релационни бази от данни. Основава на езика за структурни заявки (SQL), който се използва за манипулация, контрол и дефиниране на информация в базата данни.

**Д)** Софтуерна рамка за изграждане на frontend частта на уеб приложението – **Vue.js**

Vue e софтуерна рамка и също екосистема, която покрива основните характеристики, които са нужни за разработването на потребителски интерфейси

Vue предоставя декларативен и компонентно-базиран програмен модел, който позволява да се постигне максимална продуктивност за минимални разходи. Едни от основните черти на Vue са:

* Декларативно изобразяване (Declarative Rendering) – Vue разширява стандартния HTML със синтаксиса “template”, който позволява декларативно да опишем нашия HTML изход въз основа на състоянията на JavaScript кода.
* Реактивност (Reactivity) – Vue автоматично проследява промените в състоянието на JavaScript и ефективно актуализира документно обектния модел или DOM на уеб приложение, когато настъпят промени.

Vue също осигурява разширени функционалности, необходими за изграждането на комплексно приложение като такива функционалности са, routing, state management и build tooling.

* Маршрутизиране (routing) – механизъм, който насочва HTTP заявки към кода, който ги обработва. С прости думи маршрутизирането определя какво ще се случи, когато потребител посети дадена уеб страница или въведе определен URL адрес.
* Управление на състоянията (state management) - е имплементация на шаблон за дизайн, който ни позволява да синхронизираме нашите състояния с всички компоненти на приложението, което разработваме. Състояние може да се определи като нещо, което е част от нашата апликация. Като например списък с потребители, списък с продукти или всякакви данни, които идват от някаква база от данни.
* Инструмент за построяване (Build tooling) – Vue също поддържа инструмент, който автоматизира създаването на проект като изтегля всички необходими файлове, които са нужни за правилната работоспособност на Vue и ги структурира по специфичен начин. Известни инструменти за изграждане за Vue са “webpack” и “Vite”. Характерно също за тези инструменти е, че идват и със сървър за development. Основната употреба на тези сървъри е, когато искаме да виждаме резултатите на нашия проект, когато работим по него локално, т.е. използва се, когато приложението ни е в процес на разработка.

Vue най-често се прилага, когато е решено да се създаде „Едностранично приложение“ (Single Page Application, SPA), но не са изключени и възможностите да се създадат приложения със сървърно странично изобразяване (Server-Side Rendering), статично генериране (Static-Site-Generation, SSG), мобилни апликации, WebGL или терминални.

В тази уеб разработка е изградена на принципа едностранично приложение.

Това са уеб приложения, които взаимодействат с потребителя чрез динамично пренаписване на документно обектния модел с нови данни от уеб сървъра. Характерното е, че при първото посещение на определения уеб сайт се извлича скелет на HTML и в самия файл са описани CSS и JavaScript файловете. Според действията на потребителя, страницата се пренаписва чрез JavaScript логика, без да се презарежда страницата на ново.

Структурата на \*.vue файла е подобна на HTML файлов формат, наречен Single-File-Component (SFC). VUE SFC, както подсказва името, обединява логиката на компонента (JavaScript), шаблона (HTML) и стилизирането (CSS) в един файл.

Картина, която съдържа текст

Описанието е генерирано автоматично

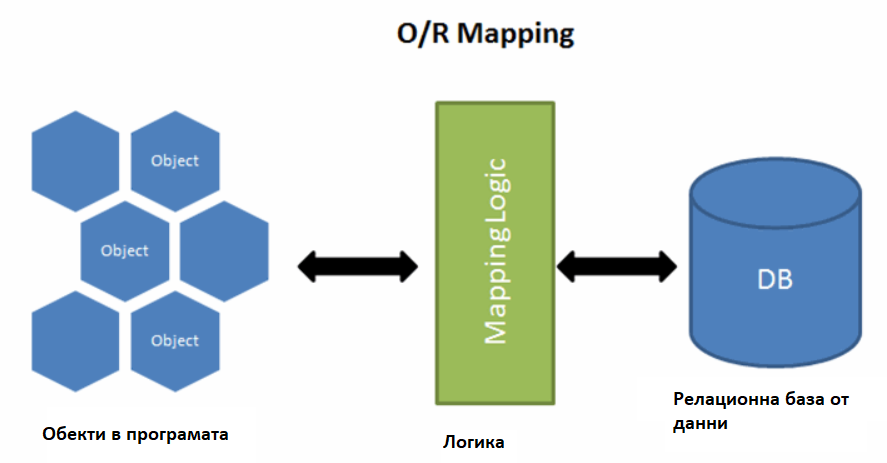
***Фигура 5***: SFC формат или базовата структура на един vue файл.

**E)** Мениджър на пакети (инструмент за управление на зависимости) – **NPM**

NPM (Node Package Manager) e мениджър на пакети за езика за програмиране JavaScript. Състои се от command-line interface (команден интерфейс) наричан npm и и онлайн база данни с публични и частни пакети, наречена регистър на npm. Достъпът до регистъра се осъществява чрез клиента, а наличните пакети могат да бъдат преглеждани и търсени чрез уебсайта на npm. Всички външни пакети, които не са част от екосистемата на Node/JavaScript, се структурират като зависимости във файла „package.json”. Този файл може да се създаде ръчно или чрез командата „npm init”, която се изпълнява от командния интерфейс на npm. Самият файл също съдържа важни метаданни, които описват проекта като номер на версията, име на автор, скриптове за стартиране на разработката, описания, лиценз и други. Други такива инструменти в различните среди за програмиране са: Maven (Java), Pip (Python), NuGet (C#) и други.

**Ж)** ORM(Object Relational Mapper), Обектно-релационно картографиране – **Sequelize**

Sequelize е модул, който позволява на JavaScript разработчиците да работят с релационна база от данни по по-продуктивен начин. Този модул е съвместим с повечето системи за управление на релационни бази от данни. Обектното-релационно картографиране е техника, която позволява да се манипулират, дефинират и контролират данните в база от данни по обектно ориентиран начин. Самите операции върху база от данни могат да се извършват чрез дефинирането на модели. Моделът в Sequelize може да се определи като репрезентация на таблица в базата от данни. Всяка инстанция на модел съдържа подробности за обекта, който представлява. Sequelize е един от предпочитаните обектно-релационни картографери, поради голямата му поддръжка на команди от езикът за структурирани заявки, лесното осъществяване на връзка със сървъра за база данни и лесното мигриране към друга система за управление на релационна база от данни.



***Фигура 6: ORM логика***

**З)** Инструмент за защитена трансмисия на данни между няколко партии (автентикация и оторизация в уеб базирани приложения) – **JWT**

JWT или JSON Web Token е отворен стандарт, използван за безопасно споделяне на информация между две страни – клиент и сървър. Това е кодиран JSON съдържащ набор от претенции (claims) и подпис (signature). JWT е популярен начин за удостоверяване/упълномощаване на потребители в уеб системи. JWT удостоверяването е токен базиран механизъм за удостоверяване без състояние (stateless). JWT се използва като клиентско базирана сесия без състояние, което означава, че сървърът не трябва да разчита изцяло на база данни за съхранение или на запазване на информацията за сесията.

JSON уеб токените могат да бъдат криптирани, но обикновено са кодирани и подписани. Целта на JWT не е да скрие данните, а да гарантира автентичността им. Препоръчително е този вид токени да се използват с HTTPS протокол.

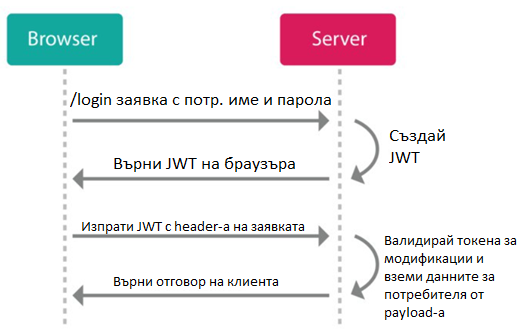
Структурата на JWT е разделена на три части: header, payload, signature.

Заглавната част се дели на два раздела. В първия се обозначава използваният алгоритъм за подписване на токена и във втория се определя типът на токена (JWT).

В payload-а се съдържат претенциите (потребителски атрибути) и допълнителни данни като издател, време на изтичане на валидността на токена и т.н.

Подписа е хеша на секциите за заглавната част и данните(payload) на JWT. Алгоритъмът, който се използва за създаване на подписа е същия алгоритъм, споменат в заглавната секция на токена. Подписът се използва за да се потвърди, че токенът не е бил модифициран по време на транзита. Използва се също и за валидиране на подателя на JWT.

Най-прост пример за употреба на JWT е, когато имаме Front-end програма, комуникираща с API чрез HTTP заявки. С помощта на JWT ще е възможно да се оторизират потребителите, използващи това API. След това може да се направи една стъпка напред и да се използва JWT за извършване на проверка на роли ( напр. когато определен API маршрут трябва да е достъпен само за потребители с администраторски права).



***Фигура 7***: Жизнен цикъл на JWT

**И)** Тестване – **JEST**

Jest е софтуерна рамка за изпълнение и структуриране на тестове върху JavaScript базиран код. Jest характерно се използва най-често за Unit-Testing. Това пък е софтуерно тестване, при което се тестват отделни единици на софтуер. Целта на тестването на единици (units) е да се потвърди, че всяка единица от софтуера работи както е проектирана. Единицата е най-малката част от всеки софтуер, която може да бъде тествана. Обикновено се комбинира с други технологии, които позволяват по обширно тестване като Integration test и End-to-end test.

## **2.4. Структура и проектиране на базата данни**

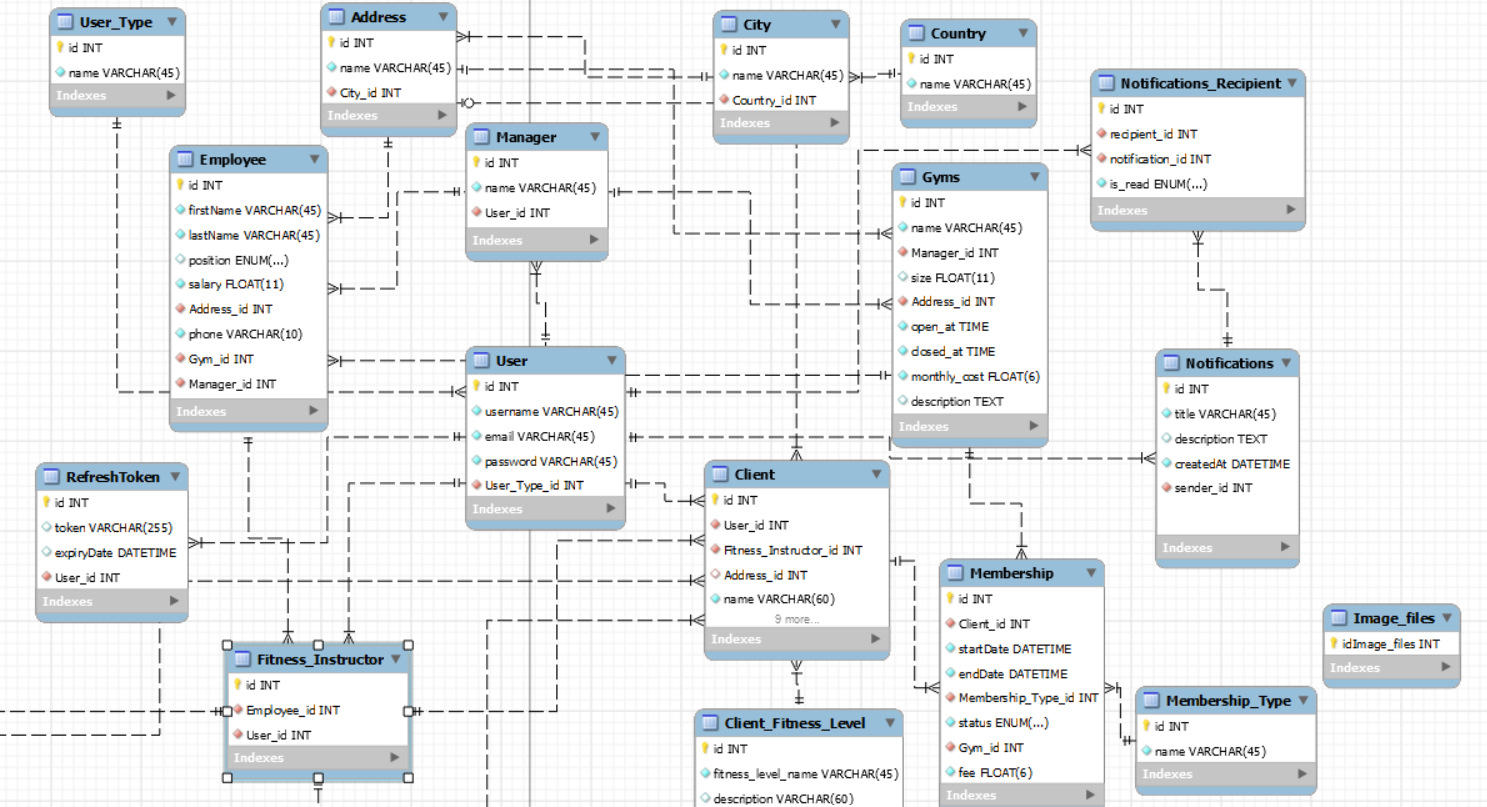
**Entity Relationship Diagram (ERD) – Диаграма на субектите и взаимоотношенията**

Базата данни на този проект се дели на 3 основни секции.

Първата секция съдържа всички основни актьори (потребител, клиент, инструктор, мениджър и т.н.) и техните различни взаимоотношения.

Във втората секция са всички таблици и връзки, които осъществяват хранителния режим за клиент в системата.

Третата включва тренировъчната програма.

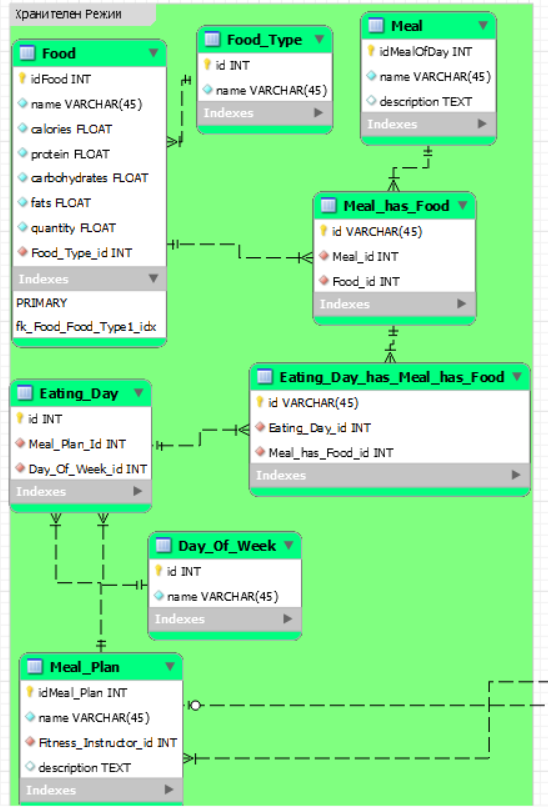
**Секция 1 от ERD:**

***Фигура 8***: ER диаграма на секция 1 от базата данни.

**Обяснение на всяка таблица:**

* **User** – Тук се съдържат всички регистрирани потребители, които имат право да използват уеб системата „Fit-M“. Съдържа колони за потребителско име, имейл, парола и чужд ключ от таблица **User\_Type**, служеща за определя на ролята на потребителя.
* **Employees –** Всички регистрирани служители в системата. Притежава име, фамилия, какъв служител е (позиция), заплата, адрес, телефонен номер и чужд ключ от таблица **Gym** (определя в кой фитнес работи).
* **Client –** Описва клиента на фитнес зала. Всеки регистриран клиент е **User**, има име, години, височина, тежест (в кг.), телефонен номер (незадължителен), фитнес програма (чужд ключ), хранителен режим ( чужд ключ), избран фитнес инструктор (чужд ключ) и още допълнителни детайли.
* **Membership** – Всеки клиент за да може да използва пълноценно фитнес системата трябва да има създадено членство във избрана фитнес зала. Всяко членство съдържа начална дата, крайна дата, тип на членството (годишно, месечно, еднодневно), статус на членството (активно или изтекло), притежател на членството (чужд ключ от табл. **Client**), такса на членството и за коя фитнес зала се отнася самото членство (чужд ключ от **Gym)**.
* **Gym** – име на фитнес залата, размер, работно време, цена на членство, описание, мениджър на залата (чужд ключ от таблица **Manager**) и адрес.
* **Manager** – име на управител и връзка към таблица **User** (защото мениджър също може да използва системата).
* **User\_Type** – описва всички типове потребители в системата (мениджър, клиент, администратор, фитнес инструктор).
* **Fitness\_Instructor** – съдържа само два чужди ключа: към таблица **Employee** и към таблица **User**.
* **Notification –** потребителите в системата имат възможност да получават различни известия. Всяко известие има получател, изпращач, заглавие, описание, дата на създаване и статус на нотификацията (дали е видяна).
* **Refresh\_Token –** Съхранява токени за потребители, които са удостоверени от системата. Служат за презареждане на JWT токените, които характерно са с краткосрочно време на валидност (поради съображения за сигурност от атаки на крадене на сесии на потребители) и влезлият потребител за да не се логва наново при всяко изтичане на JWT, системата прави заявка за нов JWT токен, като проверява дали има съществуващ запис на refresh токен, който отговаря на потребителя, който се нуждае от нов JWT. В противен случай, ако такъв запис не съществува, потребителя е подканен да се впише наново в уеб системата. Съдържа колона за токен, и дата на изтичане (като практиката е да се задава за няколко месеца).
* **Address –** Включва адрес на местонахождение и чужд ключ от таблица **City**
* **City –** Име на град, държава (чужд ключ)
* **Country –** Име на държава
* **Client\_Fitness\_Level –** Ниво на напредналост на клиент
* **Membership\_Type -** Тип на членството
* **Notificaiton\_Recipient –** Съдържа получателя на дадена нотификация и самата нотификация (чужд ключ от таблица **Notification**)
* **Image\_Files –** Връзка към всички изображения използвани в системата

**Секция 2 от ERD:**

****

***Фигура 9***: ER диаграма на секция 2 от базата данни

**Обяснение на всяка таблица:**

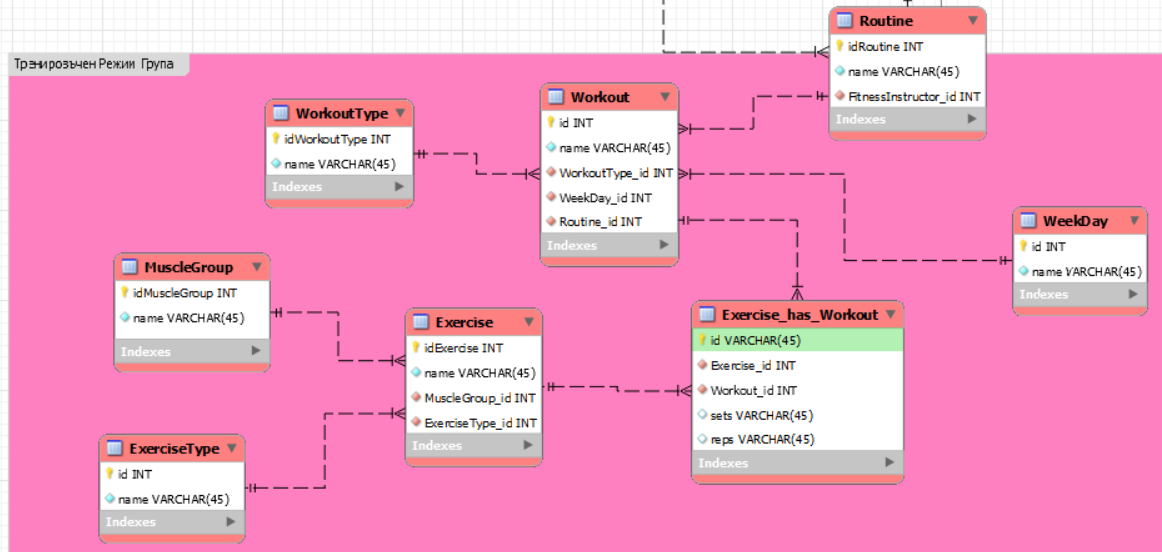
* **Food –** Описва дадена храна (напр. банан, яйца, ориз и т.н.). Всяка храна съдържа определен брой калории, макронутриенти (белтъци, въглехидрати, мазнини), тип на храната(чужд ключ от табл. **Food\_Type**) и количество (ще се измерва в грамове).
* **Meal –** Ястие (закуска, обяд, вечеря и т.н.).
* **Food\_Type –** Съдържа имената всички типове храни.
* **Meal\_has\_Food –** Свързваща таблица за връзката много към много на таблиците **Meal** и **Food** (Едно ястие съдържа много храни, също дадена храна може да се съдържа в много ястия).
* **Eating\_Day –** Ден от хранителния режим. Характерното при създаване на такива режими е, дните да са равни на броя дните в седмицата. Съдържа чужд ключ от таблица **Day\_Of\_Week** (ден от седмицата) и също от табл. **Meal\_Plan** за да се определи за кой хранителен режим отговаря дадения ден.
* **Meal\_Plan** – Тази таблица съдържа имената на създадените хранителни режими, описание и от кой инструктор е създаден режимът (чужд ключ от табл. **Fitness\_Instructor**).
* **Eating\_Day\_has\_Meal\_has\_Food -** Свързваща таблица отнасяща се за връзката много към много за таблиците Eating\_Day и Meal\_has\_Food. (Един ден съдържа много ястия, също дадено ястия може да се съдържа в много дни).
* **Day\_Of\_Week –** Съдържа имената на дните в седмицата.

Картина, която съдържа маса

Описанието е генерирано автоматично

***Фигура 10***: Примерен хранителен режим за даден клиент

**Секция 3 от ERD:**

****

***Фигура 11***: ER диаграма на секция 3 от базата данни

**Обяснение на всяка таблица:**

* **Exercise –** Таблицата описва дадено физическо упражнение. Съдържа име на упражнението, тип на упражнението (чужд ключ от таблица **Exercise\_Type**) и върху коя мускулна група работи то (**Muscle\_Group**)
* **Workout –** Тренировка. Включва име на тренировката, тип на тренировката (**Workout\_Type**), към коя тренировъчна програма принадлежи тренировката (чужд ключ от **Routine** таблицата) и в кой ден от седмицата се извършва самата тренировка (**Week\_Day**)
* **Exercise\_Type –** Съдържа имената на всички типове упражнения
* **Muscle\_Group –** Съдържа всички мускулни групи
* **Workout\_Type –** Всички видове тренировки
* **Exercise\_has\_Workout –** Свързваща таблица за връзката много към много на таблиците **Workout** и **Exercise**. (Дадено упражнение може да се съдържа в много тренировки и дадена тренировка може да съдържа много упражнения). Включва в себе си и допълнителни колони за детайли, които измерват повторения за упражнение в тренировка.
* **Routine –** Тук се намират всички тренировъчни програми в системата. Всяка програма има име, и чужд ключ от таблицата **Fitness\_Instructor**, който служи за определяне на създателя на програмата.

Картина, която съдържа маса

Описанието е генерирано автоматично

***Фигура 12***: Примерна тренировъчна програма за клиент