ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – ВАРНА

ФАКУЛТЕТ ПО ИЗЧИСЛИТЕЛНА ТЕХНИКА И АВТОМАТИЗАЦИЯ

Катедра „Софтуерно Инженерство “

A blue and white logo

Description automatically generated with medium confidence

**Курсова Работа по**

**ППС#**

**Тема:**

Информационна система, подпомагаща управлението на фитнес.

**Разработил:**  **Проверил:**

Ивайло Пламенов Руменов  гл ас. д-р инж. ДимитричкаНиколаева

Фак. № 23651227

# Анотация:

В тази курсова работа е разработена система за фитнес позволяващо следенето на клиенти и техните абонаменти към фитнеса. Реализирано като уеб базирано приложение. Нуждата на фитнес заведения за структуриран начин на следене е задължителен. Тъй като набора от различни клиенти, който влизат и излизат е много броен и за да се разраства едно такова заведение то трябва да следи приходите и разходите си.

Друго приложение на системата включва следенето на персонала. Включително тяхното заклащане запазено по месеци. Това помага за отчитане и следене състоянието на работния персонал. Така лесно може да се проследява и техния прогрес и дали някой от персонала заслужава повишение или не. Както и да се структурират работни графици за работещия персонал. Нужен е работен график тъй като фитнеси отварят раните часове на деня и затварят в късните часове на деня. Това е така тъй като повечето посещаващи фитнеса си имат работа и или идват рано сутрин преди работа или късно вечер след работа. И така персонала да не работят по дванадесет часа на ден се нуждае от оптимален начин на следене на техните работни часове. Това позволява и следенето на треньорите да си тренират оптимално без да има препокривания на клиенти или големи разлики помежду записаните часове за трениране.

Когато се знае колко хора и по колко пъти ще тренират на ден фитнеса може да предложи на посетителите си различен набор от абонаменти. За демонстриране в системата вкаран разновиден набор от планове за посещение. От сутрини и вечерни, които позволяват посещение или в сутрешните часове или в вечерните, до премиум пакети включващи посещения по всякое време много кратно и супер премиум пакети който включват тренировка с треньор и спа услуги. При такова разнообрази следенето на всички клиенти и те как посещават фитнеса става много напрегнато. Затова е нужна подобна система да следи наплива и желанието на клиенти. По този начин един фитнес е по-конкретно способен на пазара. Тако може и да се следи и наглася в реално време ценоразписа на абонаментите. Както и предлагане на пакетни оферти и сезонни намаления.

# Увод

В тази курсова работа ще се разгледа система обслужване на фитнес. Разглежда се как се формира вю модел контролер използвайки С# като програмен език за логика на контролера. Модела е реализиран чрез релационна база от данни разгъната на облачен сървър използващ Posgres като база от данни. За интерфейс е избран Angular, която е рамка на вече популярния JavaScrip масово употребяван за реализация на уеб базирани интерфейси.

Популярността на фитнесите сред широка гама от популацията води до жестока конкуренция. Широкото желани и търсене на място за трениране се дължи на множествено фактори. От спортния дух на старото поколение което е научено че спорта е здраве и го предава на новото поколение което от друга страна се влияе и от по западната концепция на трениране в фитнеси. Едно е сигурно, и това е че фитнесите сближават млади и стари в едно място за едно нещо и това е тренирането. Поради това широко търсене един фитнес, който желае да се развива трябва да следи клиентите си и техните навици и предпочитания.

Когато един фитнес има статистики за кой кога и по колко пъти на денс се посещава, то лесно може да си направи изводи, отчети и стратегии за разгръщане на бизнеса. Както и когато се следят клиентите но може лесно да се планират промоции, пакетни сделки и допълнителни планове. Както и кога се знае потока от хора и натовареността то може лесно и треньорите в фитнеса да си планират тренировките с клиенти, така че да не се престъпват клиенти да не има голяма разлика помежду . Всичкото това допринася за работата на фитнеса като цяло.

Избрано е реализацията да е уеб базиране, поради множествено фактори. Първия от тях е че уеб базирано приложение позволява множествено интерфейси, което означава че приложението има място за развитие. Втория е свързан с очаквания на клиенти и потребители. Днешните времена е широко прието че всеки по един или друг начин е свързан с интернета и съответно приложенията. Така клиента може да си следи статистиките кога и колко често посещава фитнеса. Както и треньорите да следят кога имат тренировка и с кого. Уеб базираните приложени са голям бонус за всички страни. Те предоставят леснота при достъпване на данни от различни локации и устройства. В проекта е разгледан браузарен интерфейс но лесно в бъдещето може да се добави друг интерфейс, като мобилен, който да се използва повторено вече създадената логика.

Използването на C# предоставя големи бонуси като обектно ориентиран език с опцията на функционална програмиране при нужда. Програмния език позволява и преизползване на съществуващ код за предотвратяване на повторения и бъгове който вървят с тях. C# приложен с dot net core предоставя широка гама от сертифицирани библиотеки, който да се използват за осенения на програмиста при създаване на логиката на продукта.

# Обзор на съществуващите решения. Изводи. Цел и задачи.

Когато се започва такъв проекта трябва първо да се направи маркетинг анализ на програмните продукти съществуващи на позата. Техните идеи за продукт може да са еднакви но реализацията и работния модел може да е коренно различен.

Gymmaster един продукт на пазара, който е най-близката логиката и целта на проекта. Програмния продукт предоставя: създаван на графици за персонала, организиране на клиенти който ще тренират с треньор. От страна на клиента предоставя и мобилно приложение за планиране на тренировки с треньор и без, комуникация с треньор по чат и допускане до фитнеса чрез блутут карта за достъп до самия фитнес. Предоставя и онлайн плащане с карта за обновяване на членството на клиентите.

Мindbodyonline е още един пример за продукт свързан със същата логика като заданието. Но с изключението че предлагат групови тренировки за групови спортове като йога, пилатес, спининг даже и медитация. Както и сходно с продукта, който е по задание има следене на треньорите си и работния персонал. Този продукт поддържа и подновяване на членство чрез банков превод. И не се нуждае члена да е на физичното място на фитнеса за да си поднови членството.

Nike Training Club е по-различно от предходните приложения. Главно приложение е съсредоточено върху следенето на фитнес състоянието ти и научаване и имплементиране на нови тренировки. Както и служи за социална мрежа къде може да говориш с други фитнес практикуващи или с професионални треньори.

Exercise.com e софтуер съсредоточен върху треньорите и те как си организират тренировки с индивидуални хора или големи групи. Информационната страница описва софтуера гласи че е предназначен за висок набор от членове на трениращите групи. Поддържа записване на часове във всеки ден от годината, чатове с хората в групите, чатове в цялата група и записване на предварителни упражнения така че групата да е информирана преди началото на срещата.

Извлечените приложения са само малка извадка от най-популярните на пазара. Пазара е ситен в тази зона и влизането се оценява с голям риск и ника успеваемост за продукта. Множество продукти се представата върху различни зони. Но от проучване по темата се стига до извода че на българския пазар не се използват подобни приложения да менажирането на фитнес.

# Проектиране и описание на предлаганото решение

В тази глава е разгледано как е изпълнен проекта в неговата цялост. Започвайки от планиране на стъпките за осъществяване на проекта, през изпълнението и методология на вътрешните алгоритми до финалното приключване и планиране за подобрения и поддръжка на приложението. След уточняване на изискванията, целта, и е направена пазарна оценка може да премине към планиране на проекта и очертавайки нужните стъпки за да успее в определен период от време. Организирането и планирането на един проект е ключова за неговото завършване на време. Когато чисто и ясно са разписани условията за завършеност то се преминава към разписване на функциалноста, която ще покрие всичките условия за завършеност. Проекта е организиран използвайки свободен за ползване софтуер наречен Notion. В него е употребено функциалноста за разписване на задачи под формата на кан-бан където всяка задача е строго свързана с функция от проекта която ще трябва да бъде имплементирана. Първо се разписва общата функционалност след което се записват неговите детайли като подзадачи. Всяка група от задачи бе разпределена на по две седмични за изпълнение и след изтичане на периода е оформена справка с какво да подобрим за следващата седмица и какво би могло да се започне за следващите. И така до изчерпване на условията за завършване.

## Изисквания към програмната система.

Програмните изисквания представляват софтуерната рамка под която ще се организира програмния код. Тази рамка включва: вътрешна-външна комуникация, имплементация на базата от данни и как общува с данните обекти и начина на сигурност.

Фундаментален елемент за всяко приложения е начина на обработване на данни, или така наречения „Data interpretation and handling “ или на български структури от данни и обработката им. По условие е дефинирано базата от данни и какви да са таблиците в базата. Базата от данни е релационно базирана. За изпълнението на тази задача е избрана базата Posgres. Posgres е широко известна база от данни разработена Калифорнийски университет Бъркли и е система за управление на релационни бази данни с отворен код.

След установяване на базата може да се премине към как данновите обекти ще се обработват и записват в базата. За целта е избран програмния език C# поради неговата стабилност при писане на обектно ориентирани приложения. В случая приложението е предназначено да се употребява в уеб базирана среда затова за контролер е решено да се използва dot net core разработен от Microsoft и служи за отличен начин за употреба на C#. Dot net core силно имплементира стандартния начин на писане на уеб приложения по обектно ориентиран начин. Сходно на java enterprise или още познато като java EE при писането на логика в този модул се декомпозира в само изолирани модули с конкретно имплантация по интерфейс.

И за целта на потребителски интерфейс е избрано да се имплементира чрез Angular скеле, изграден от JavaScript и TypeScript. Това е избрано поради начина по който се прехвърля информация помежду един интерфейс модел, така нареченото „View“, и контролера реализиран на dot net cote. Метода за прехвърляне на информация помежду интерфейса и контролера е чрез http rest заявки, който се класифицират на GET за вземане, POST за подаване, PUT за поставяне, UPDATE за подновяване, PATCH за поправка, DELETE за премахване, HEAD вземане на адрес и OPTIONS за вземане на опциите който са достъпни за клиента. При използването на тези методи най-често се прикача тяло (‘body’) към заявката. За целта на положението е уточнено че се използват заявките с Json тяло. Съществуват множество варианти за изпращане на тяло по REST заявка. Тези тела включват text, JavaScript, HTML и XML но за улеснение и лесна работа е избран стандарта с Json. Благодарение на json можем да изпращаме бързо и лесно специфични обекти предефинирани, подредени и готови за употреба в интерфейса на приложението.

## Логически модел на програмната система

Логика и метод на програмиране е ключова за изпълнение на програмен продукт който ще е устойчив след време но и бърз за изпълнение в периода на задание. Това колко е успойчив един продук е строго свързано със законите на обектното ориентирано програмиране. Затова е заложено за модела на програмиране той да следва принципите на обектното ориентирано програмиране. Обектното ориентирано програмиране (ООП) е парадигма в програмирането, която използва обекти – инстанции на класове, за да организира кода. Принципите на обектното ориентирано програмиране са основните концепции и идеи, които лежат в основата на тази парадигма. Ето някои от тези принципи:

Енкапсулация (Encapsulation) Енкапсулацията се отнася до създаването на обекти, които обединяват данни и методи, които работят с тези данни, в една единица. Целта на енкапсулацията е да скрие вътрешните детайли на обекта и да предостави интерфейс, чрез който външният свят може да взаимодейства с обекта.

Наследяване (Inheritance) Наследяването позволява на един клас да придобие свойства и методи на друг клас, така че да се избегне повторното писане на код. Наследяването подпомага в създаването на йерархии от класове, което дава възможност за повторна употреба на код и лесно разширяване на функционалността.

Полиморфизъм (Polymorphism) Полиморфизмът позволява на обекти от различни класове да бъдат третирани като обекти от един и същи клас. Полиморфизмът улеснява създаването на гъвкав и разширяем код. Един и същ метод може да се използва за обработка на обекти от различни типове.

Абстракция (Abstraction) Абстракцията се отнася до скриването на детайлите на реализацията и предоставянето на опростен интерфейс за взаимодействие. Целта на абстракцията е да улесни разработката на софтуер и да управлява сложността чрез предоставяне на ясен и опростен интерфейс.

Обекти (Objects) Обектите са инстанции на класове, които обединяват данни и методи. Чрез използването на обекти, програмистите могат да моделират реални обекти или концепции в програмния код, което прави програмите по-интуитивни и лесни за разбиране.

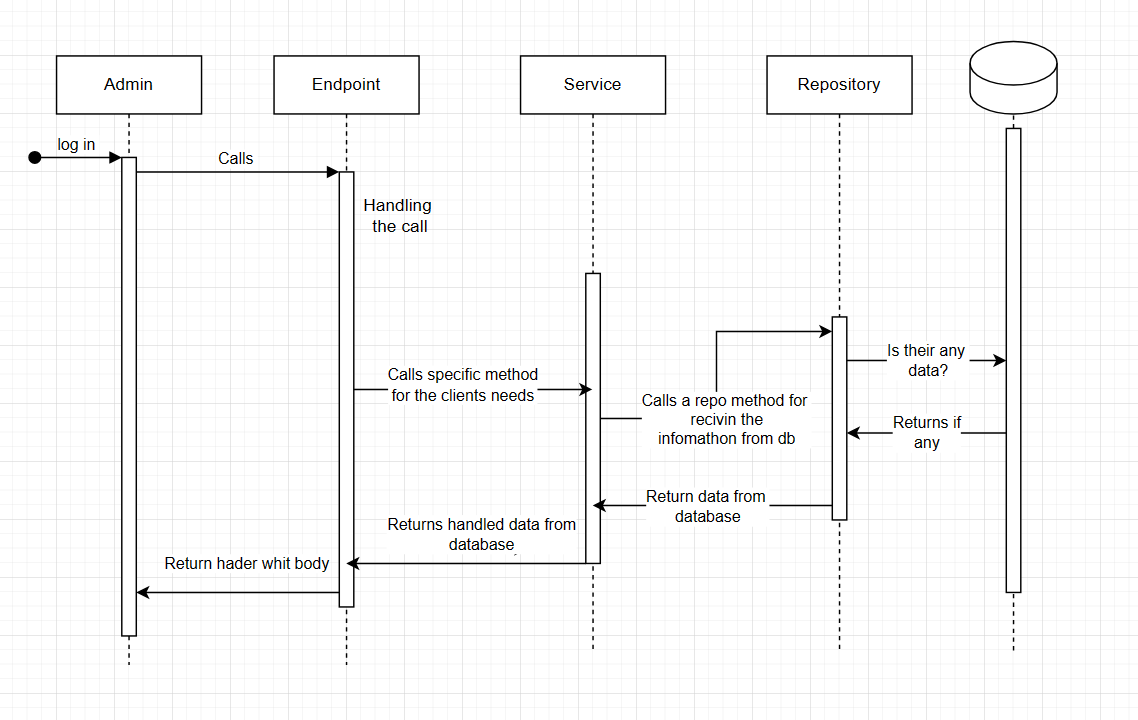
Тези принципи образуват основата на обектното ориентирано програмиране и помагат за създаването на поддръжаем и разширяем код. Използването на тези принципи води до по-ефективно и устойчиво програмиране.

Потоците на данни както прегледано в предходната точка се състои от HTML заявки съдържащи Json пакети в тялото на заявката. Това се отнася и за двете посоки на процеса. Тоест от интерфейса към сървъра и обратно. След обработката на тялото на заявката, данните биват транслирани от Json към вътрешен data object съдържащи сходни или еднакви полета като Json пакета. При поличаване на данните те се използват за оформяне на заявката към базата от данни. След получаване на отговор от базата данни, то данните се обработва пак в обект и се изпраща към интерфейса (Фиг №1,2).

A diagram of a company

Description automatically generated

Фиг. №.1- Графика изобразяваща процеса на потребителския интерфейс.

A diagram with black text and white text

Description automatically generated

Фиг. №.2 – Графика изобразяваща как се прехвърля информация за всички връзки към сървъра.

## Архитектура на системата

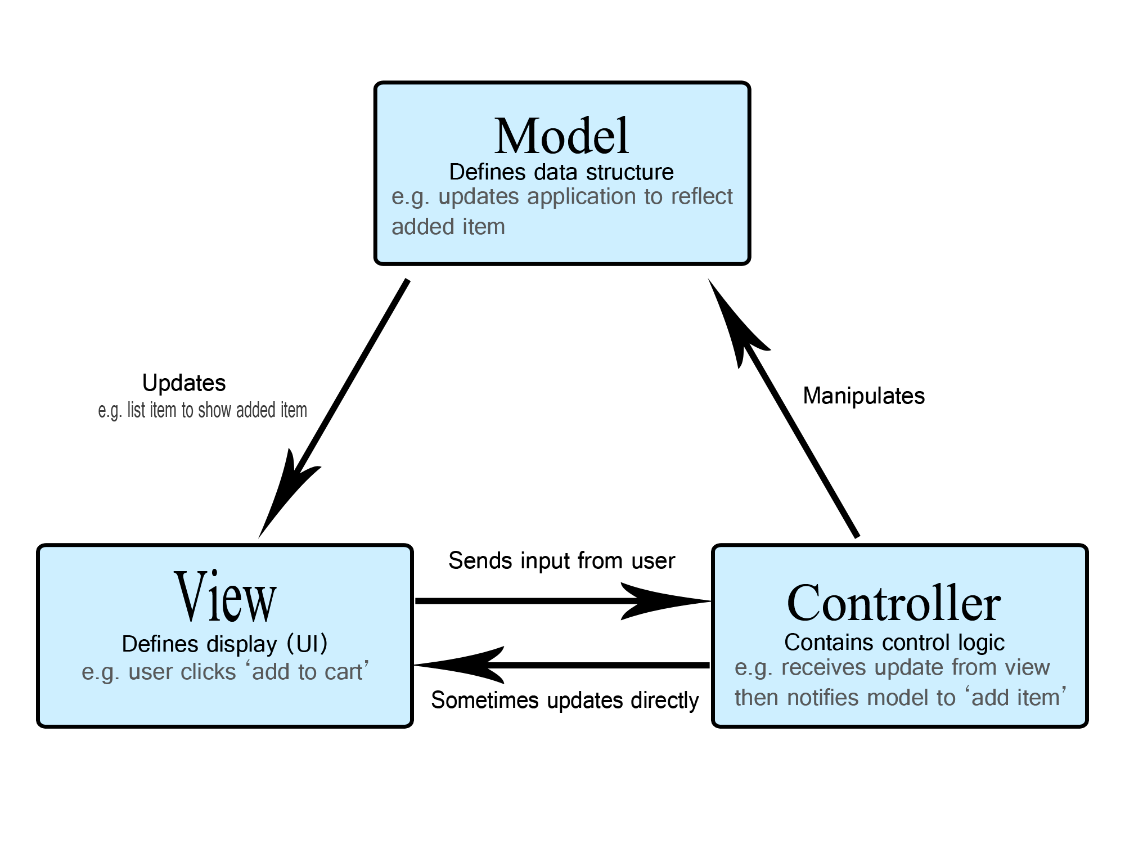
Архитектура на системата представлява как бизнес логиката по спецификация ще се разгърне като програмен код. Тук абстракцията на бизнес логиката трябва да се инспектира за да се знае с конкретност как да се имплементира логиката. Но след множество разглеждания, включително на потока от данни и те как трябва да си взаимодействат един с друг е стигнато до решението да се използва широко разпространения модел-вю-контролер (Фиг. №3) метод на организиране на системата.

Модел-Вю-Контролер (MVC) е архитектурен шаблон, който се използва за проектиране на софтуерни приложения. Той разделя компонентите на приложението на три основни части: Модел, Вю и Контролер. Този шаблон има за цел да подобри управлението на данните, потребителския интерфейс и бизнес логиката, като ги отделя и прави по-лесно разширяеми и поддържаеми.

Моделът (Model) представлява данните и бизнес логиката на приложението. Той отговаря за съхранението, обработката и манипулирането на данните, както и за известяването на Вю-то и Контролера при промени в данните. Ако разглеждаме уеб приложение за управление на задачи, моделът може да включва класове и функционалности, свързани със задачите, като списък от задачи, добавяне, изтриване и обновяване на задачи.

Вю-то (View) е отговорно за представянето на данните на потребителя и за визуалния интерфейс на приложението. То получава информация от Модела и я представя по начин, удобен за визуално взаимодействие с потребителя. Вю-то не трябва да съдържа логика за обработка на данни. В уеб приложение за управление на задачи, Вю-то би могло да бъде интерфейсът на потребителя, където той вижда списък със задачи и бутони за добавяне или маркиране на задачи като изпълнени.

Контролерът (Controller) действа като посредник между Модела и Вю-то. Той получава вход от потребителя през Вю-то, обработва го и актуализира Модела. Също така, Контролерът може да получава обратна връзка от Модела и да я предоставя на Вю-то. В контекста на управление на задачи, Контролерът може да обработва заявки за добавяне, изтриване или обновяване на задачи, изпращайки съответните команди към Модела и обновявайки Вю-то.

MVC архитектурата подпомага разделението на отговорности и улеснява разработката и поддръжката на софтуер. Този шаблон се използва в много различни видове приложения, включително уеб приложения, десктоп приложения и мобилни приложения.

Фиг. №3- Диаграма изобразяваща архитектурата на Модел-Вю-Контролер.

## Организация на данните

Организацията на данни, както е споменато многократно и в другите глави, е в релационна база от данни. Избрано е за изпълнение на условието на заданието да се използва PosgreSQl.

Изборът между PostgreSQL и други системи за управление на бази данни (СУБД) зависи от конкретните изисквания и нужди на проекта. PostgreSQL (познат още като Postgres) обаче се счита за мощна и разширяема релационна база данни, и има няколко качества, които го правят предпочитан избор в много сценарии:

Отворен код - PostgreSQL е свободен и отворен код, което означава, че можете да сваляте, използвате и променяте кода му според вашите нужди, без да плащате лицензни такси. Това осигурява гъвкавост и контрол върху софтуера.

Пълна Поддръжка на SQL- PostgreSQL има пълна поддръжка на стандартния SQL, който прави миграцията от други релационни бази данни по-лесна и преносима. Това също така улеснява обучението на нови разработчици и администратори.

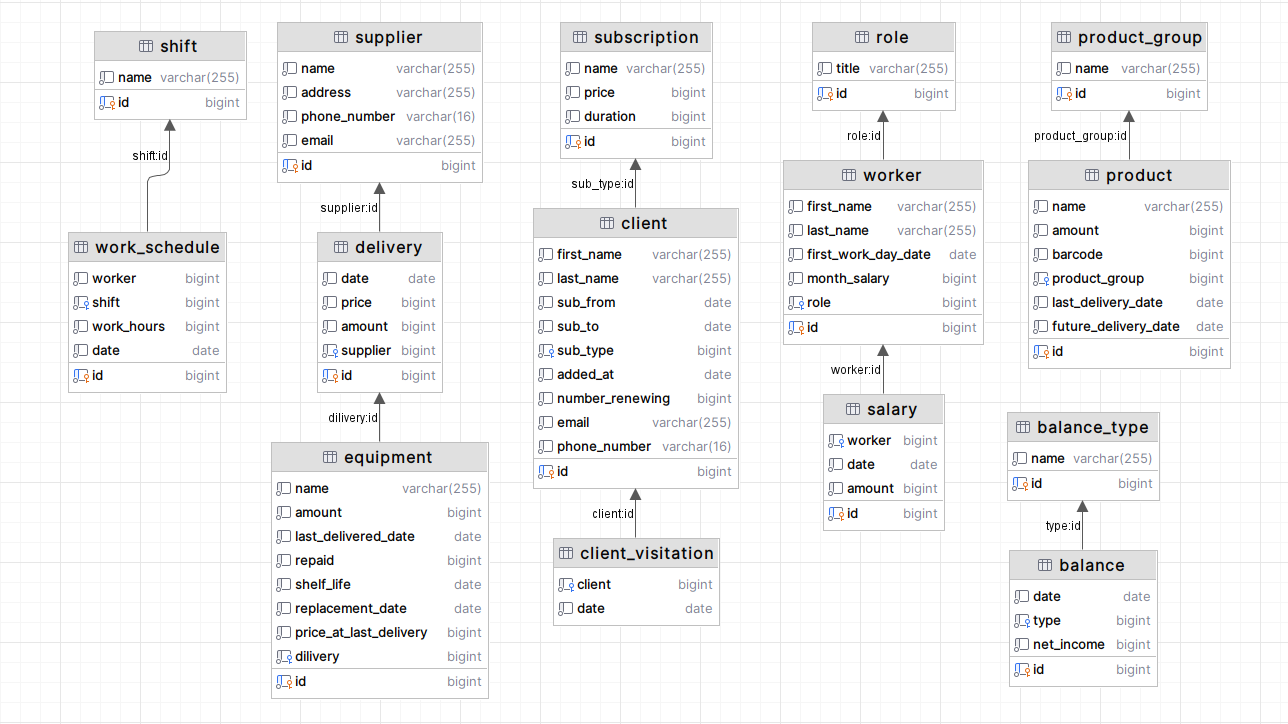
Широк набор от функционалности- Postgres предлага богати възможности и функционалности, включително поддръжка на JSON и JSONB, географски типове данни, тригери, процедури, транзакции и други. Този богат набор от възможности го прави подходящ за разнообразни сценарии.

Многозадачност и Скалируемост- PostgreSQL е известен с добрата си поддръжка на многозадачност и скалируемост. Той може да обработва големи обеми от данни и заявки, като предлага оптимизации за използване на памет и процесорни ресурси.

Голямо общество и Общностно Развитие- Поради своя отворен код, PostgreSQL има голямо общество от потребители и разработчици, което означава, че има активно обновление, поддръжка и развитие на софтуера. Това също така предоставя достъп до множество ресурси и информация в общността.

Надеждност и Транзакционна Поддръжка- PostgreSQL е известен със своята надеждност и поддръжка на транзакции. Той предлага ACID (Atomicity, Consistency, Isolation, Durability) свойства, които гарантират консистентността и надеждността на данните.

Въпреки тези предимства, изборът на база данни зависи също и от конкретните изисквания на проекта, предпочитанията на отбора за разработка и други фактори. В някои случаи други СУБД може да са по-подходящи в зависимост от конкретните нужди на проекта. Таблиците в базата са структорирани по изискванията и са надградени за допълнителна функционалност и нормализация на базата (Фиг. №4). В приложението на документацията е скрипта отговорен за създаване на база, всичките таблици в нея и примерни данни.



Фиг. №4- Диаграма на таблиците и техните релации.

## Избор на език и среда за програмиране

Избор на език за програмиране е ключов поради методологията по която езика е силен. Като за пример Java е предназнача единствено и само за обектно ориентирано писане на код, той позволява функционално програмиране но не и в чистата му форма. Език като C, при неговото създаване е предназначен за финално програмиране. Но език като C# поддържа и двете и ги съчетава. Въпреки че има много езици, които се използват за уеб програмиране, езикът C# често се избира поради следните предимства:

Интеграция с .NET Фреймуърк. C# се използва като основен език за програмиране в .NET фреймуърка, който предоставя богати библиотеки и инструменти за разработка на уеб приложения. .NET обхваща широк спектър от технологии, включително ASP.NET за уеб разработка, което улеснява създаването на мощни и ефективни уеб приложения.

Силна Типизация. C# е език със силна типизация, която допринася за сигурността и устойчивостта на кода. Предимството на силната типизация е, че грешки с типове се откриват по време на компилация, което прави кода по-предсказуем и по-лесен за поддържане.

Обширна Общност и Ресурси. C# и .NET имат голяма общност от разработчици, която предоставя източници, документация и подкрепа. Този фактор е от значение при решаването на проблеми, споделянето на знания и ученето от опита на други програмисти.

Многоплатформена Поддръжка. С появата на .NET Core и по-късно .NET 5 и .NET 6, C# стана многоплатформен и може да се използва за разработка на уеб приложения не само за Windows, но и за Linux и macOS. Това предоставя гъвкавост и възможност за избор на платформа.

ASP.NET MVC и ASP.NET Core. ASP.NET MVC (Model-View-Controller) и по-новият ASP.NET Core са фреймуърки за уеб разработка, които предоставят структурирани и мащабируеми модели за построяване на уеб приложения. Те предоставят решения за рутиране, управление на сесии, сигурност и други аспекти на уеб разработката.

Интеграция с Microsoft Технологии. C# и .NET са тясно интегрирани с другите технологии на Microsoft, като SQL Server, Azure Cloud и Visual Studio IDE. Това улеснява разработката и деплоя на уеб приложения в инфраструктурата на Microsoft.

Въпреки тези предимства, изборът на C# трябва да се направи в контекста на конкретните изисквания и цели на проекта. За някои приложения други езици (като JavaScript, Python, Ruby и други) или фреймуърци може да бъдат по-подходящи.

Средата за програмиране избрана е от създателите на IntelyJ, JetBreains и се казва Rider. От скоро на пазара Rider е единствената устойчива среда, която може да се при за конкуренция на Microsoft Visual studio и сестриният им продукт Visual Studio Code. Сравнено с версията на Microsoft, Rider позволява еднаква функционалност като Microsoft.

## Реализация на програмната система.

Реализацията на системата е стравнително кратка чат спрямо други стъпки на цялосния проект. Реализацията представява разписване на програграмния код по предвидени ораганизационни стъпки. Но в тази час най-често се случват спънки и преорганизиране на не добре изчистени спецификаций по проекта. Като на пример подаване и премане на дадени интерфейси или използване на абстракций извън техните целеви фукций. Но това се предвижда от програмиста и се отговара адеквтно, и по конкретно се отговара на проблем не с по-добро удобство на програмиста а с крайните цели на клиента в този слчай спецификацийте на заданието.

Относно реализацията на проекта се отбпвязва че имало трудности по пътя но е и имало множество боноси за убогатяването на знанията за програмния език и начина на абстрахиране на данни и процеси.

## Структура на данните - избор на ключови полета, тип и размер.

Структурирането на данни е ключова част от страна на разбиране на кода но и за избягване на поварения и спазване на принципите на обектното ориентирано програмиране. Структурата на проекта се състои от няколко главни елемента. Обекти които поемат информация от външния поток. И обекти които изпращат данни към външния поток.

Относно обектите които приемат информация чрез HTTP хедер или HTTP тяло. Те биват задължително проверени на входа на програмата. След първоначалната проверка че данните са в техния положителен вид, те не биват проверяване повторно, тъй като проекта следва концепцията на писане на код по контракт. След това са обектите, който служат за изпращане информация на външния канал. Те отново сливат конвенцията на писане по контракт (Design by Contract ). Те веднъж създадени не се проверяват за тяхната устойчивото, туй като това може да доведе по бъгове в системата. По-добре е да имаш една устойчива проверка при създаване отколкото множествено проверки които могат да прикрия бъг или да отворят програмата към допълнителни логически проблеми.

Design by Contract (DbC) е софтуерен дизайн подход, въведен от Бертран Майер в книгата му "Object-Oriented Software Construction" през 1988 г. Този подход се фокусира върху формалното специфициране на поведението на софтуерния компонент чрез дефиниране на контракт между компонентите. Идеята е да се осигури ясност и сигурност при разработката на софтуер, като се използват предварително дефинирани договори между компонентите на системата.

DbC включва три основни елемента:

Предусловия (Preconditions). Това са условията или ограниченията, които трябва да бъдат изпълнени преди извикването на определена операция. Ако предусловията не се изпълняват, то извикването на операцията е невалидно.

Постусловия (Postconditions). Това са условията, които трябва да бъдат изпълнени след изпълнението на определена операция. Постусловията описват очакваните резултати или състояние след успешното изпълнение на операцията.

Инварианти (Invariants). Това са условията, които трябва да са в сила преди и след извикването на операция и които не трябва да бъдат нарушени от самата операция. Инвариантите помагат да се гарантира съответствие на състоянието на обекта на системата.

Принципите на Design by Contract се използват за подобряване на документацията, разбираемостта и устойчивостта на софтуерните системи, като предоставят ясни правила за взаимодействие между компонентите. Този подход обикновено се свързва с обектно-ориентираното програмиране, където компонентите са обекти с методи и състояния.

## Описание на програмните модули - изпълнявана функция, интерфейс между отделните програмни модули, обобщен алгоритъм

В проекта съществуват два междинни интерфейса. Единия седи помежду сървъра и потребителския интерфейс а другия седи помежду базата от данни и сървъра. Започвайки от горе надолу помежду потребителския интерфейс и сървъра седи rest слоя.

REST (Representational State Transfer) е архитектурен стил за проектиране на софтуерни системи, които работят върху разпределена мрежа. REST слой представлява интерфейс, който използва принципите на REST за комуникация между клиенти и сървъри. Този слой дефинира начина, по който клиентите могат да извършват операции върху ресурси (например данни или услуги), които се предоставят от сървъра.

Основните принципи на REST включват:

Ресурсите (Resources) са основните елементи, които се управляват в системата. Всеки ресурс се идентифицира уникално, например, чрез URI (Uniform Resource Identifier).

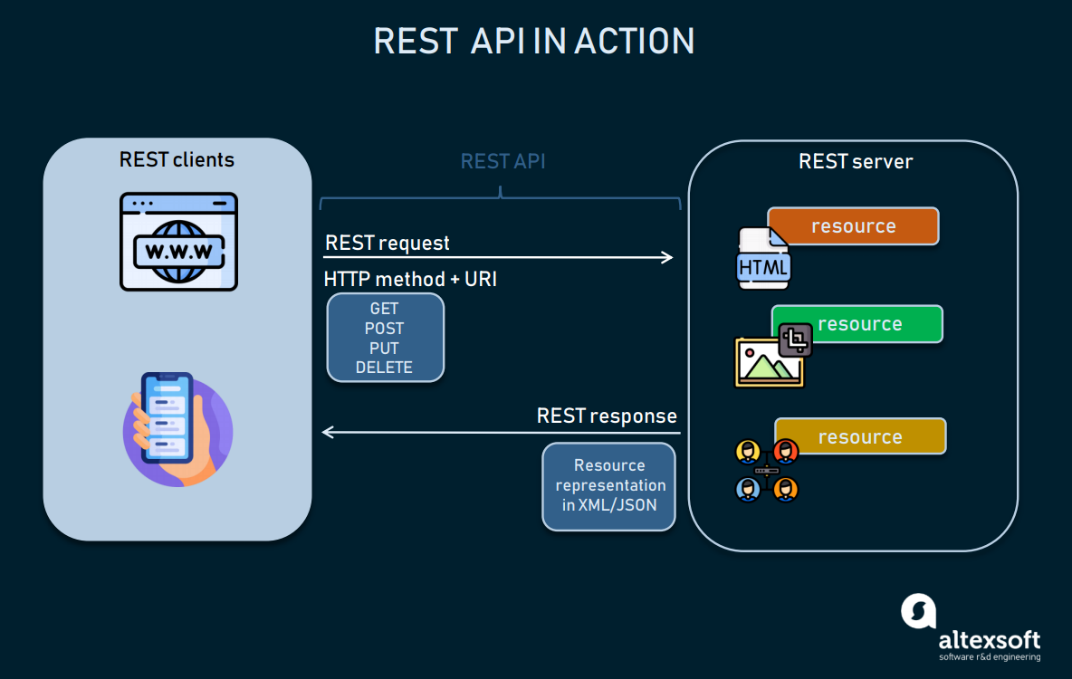
Представление (Representation). Ресурсите имат различни представления, които могат да бъдат предоставени на клиентите. Например, ресурсът може да бъде представен като HTML документ, JSON обект, XML файл и др.

Състояние (Stateless). Всяка заявка от клиента към сървъра трябва да съдържа всички необходими данни за изпълнение на заявката. Сървърът не трябва да съхранява състояние за клиента между заявките.

Еднозначност на интерфейса (Uniform Interface). Интерфейсът трябва да бъде унифициран, което прави взаимодействието между клиентите и сървърите по-прозрачно и разбираемо.

Система с кеширане (Cacheable). Отговорите от сървъра могат да бъдат кеширани, за да се подобри ефективността на системата.

REST слоят се използва за създаване на API (Application Programming Interface), което позволява различни приложения и услуги да обменят данни и функционират заедно по стандартизиран начин. Обикновено комуникацията между клиент и сървър в REST архитектурата се извършва чрез HTTP протокол, като операциите се изразяват с HTTP методи като GET, POST, PUT и DELETE.

Фиг. №5- Диаграма на REST модел за създаване на апликации.

## Формат на извежданите справки

Формата на издаващи справки представлява как една справка бива написана така че да се изведе нужната информация без да има нужда от много кратно достъпване до базата от данни. Повечето добри справки имат множество включвания (joins) без да има нужда от множество проверки(Where closes) при извличането на информация. Когато базата прави проверка тя я прави индексирано тоест минава праз всички зададени елементи от заявката да търси за дадения критерий. А когато има множество включвания (joins), базата оптимизирано вземе индекса, който трябва да свърже към дадения критерий. В приложението са вписани всички заявки по проекта и те как биват осъществени.

Фиг. №6- Диаграма показваща разновидности при вмъкване на таблици.

## Инструкции за работа с програмната система:

За да се стартира програмната система първо трябва да се стартира сървъра след компилиране на програмния код. За компилиране на C# код в изпулним се използва командата ` csc /t:library server.cs `. Но за dot net server ще трябва да се използва по сложен скрипт за констроиране на проекта.

` dotnet build [<PROJECT>|<SOLUTION>] [-a|--arch <ARCHITECTURE>]

[-c|--configuration <CONFIGURATION>] [-f|--framework <FRAMEWORK>]

[--disable-build-servers]

[--force] [--interactive] [--no-dependencies] [--no-incremental]

[--no-restore] [--nologo] [--no-self-contained] [--os <OS>]

[-o|--output <OUTPUT\_DIRECTORY>]

[-p|--property:<PROPERTYNAME>=<VALUE>]

[-r|--runtime <RUNTIME\_IDENTIFIER>]

[--self-contained [true|false]] [--source <SOURCE>]

[--tl:[auto|on|off]] [--use-current-runtime, --ucr [true|false]]

[-v|--verbosity <LEVEL>] [--version-suffix <VERSION\_SUFFIX>]

dotnet build -h|--help

` - Скрипт предоставен от Microsoft.

След конструиране на сървъра с стига до следващата част на която е базата от данни. За ще е нужда да се стартира магистративния панел на posgresql или да се използва софтуер за достъпване на дистанционни бази от данни. Тъй като базата свърза с проекта се намира на облачно пространство и е предефинирана в спецификациите на проекта. В приложението на документацията се намират всички скриптове за създаване на таблиците. И в приложението се намира всички примерни данни за приложението.

Последната стъпка е най-лесната и това е компилирането на потребителския интерфейс. Това става благодарение на прекрасния начин, по който се извършва това:

` ng build --watch`

След конструирането на програмните файлове в готови за разгъване, се преминава към инсталирането на уеб сървара с командата:

` lite-server --baseDir="dist/project-name"`

След инициализиране на уеб сървъра може да се прикачва каквато и да инстанция на Angular build. Тоест на тази инстанция може динамично да се подават нови и обновени версии на програмния продукт. Както и Angular web поддържа връзка с git repository, от което да се компилира и монтира динамично при всяка нова модификация върху репото.

## Ръководство за потребителя

Потребителското ръководство е едно от пипкавите работи за изготвяне. Тъй като един програмист и архитект на приложение в перфектния всят няма да има нужда от интродукции как да употребява приложението. Като за пример социалните мрежи които всеки знае как да ги ползва без да има нужда от инструкции как да използва инструментите създадени за него.

Започвайки от административния панел, който представлява единствени портал за статистики отговорни за клиенти и за персона (Фиг. №7). Той включва пет заявки към базата от данни четири от които са в табличен вид, едната под формата на текстов ред и последната представена като столчета диаграма.

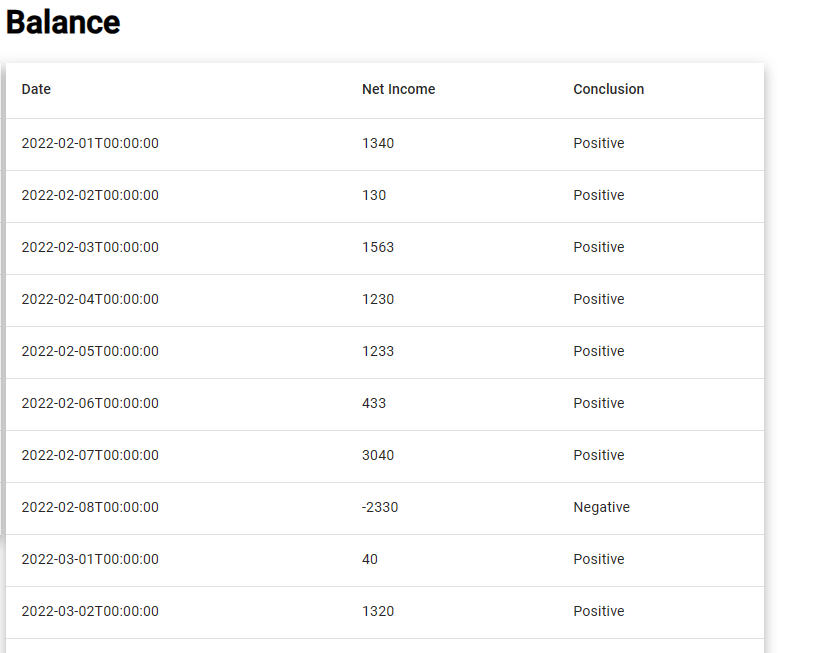
Фиг. №7- Административен портал

Първата озаглавенa „Workers“ служи за списък с всички служители и техните заплати по месеци (Фиг. №8).

A screenshot of a computer

Description automatically generated

Фиг. №8- Заявка за визуализиране на работещия персонал

Следващата справка озаглавена “Balance” представлява месечен отчет с приходите и разходите.

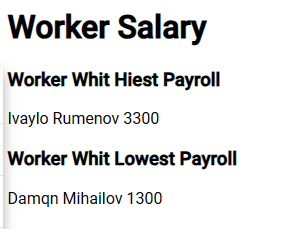
Фиг. №9- Заявка представяща месечния баланс на фирмата.

A screenshot of a web page

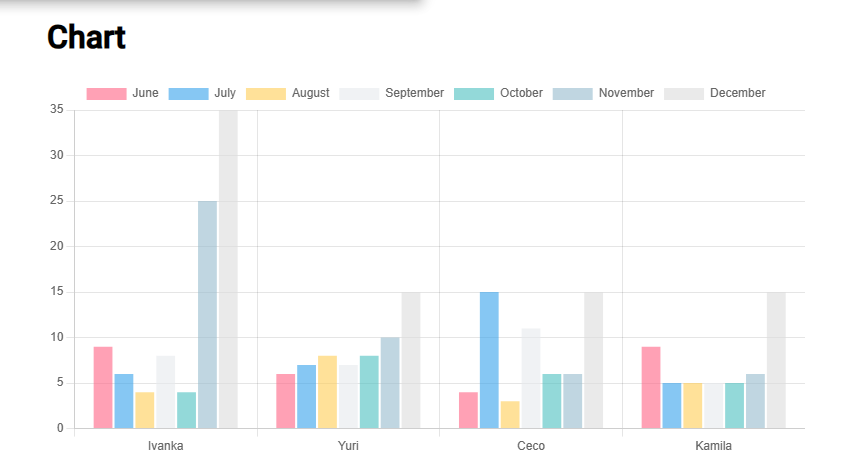
Description automatically generatedСледващата справка озаглавена „Monthly Visistations“ представлява месечен и дневен отчет за посещенията на залата по график.

Фиг. №10- Заявка показваща месечните посетители на фитнеса.

Следващата справка озаглавена „Worker Salary“ съдържа две полета едното с име “Worker Whit Hiest Payroll” и “ Worker Whit Lowest Payroll” и съдържат полета, които представляват Определяне на служителите с най-висока и най-ниска заплата за последните 5 години.



Фиг. №11- Заявка показваща работниците с най-висока и най-ниска заплата.

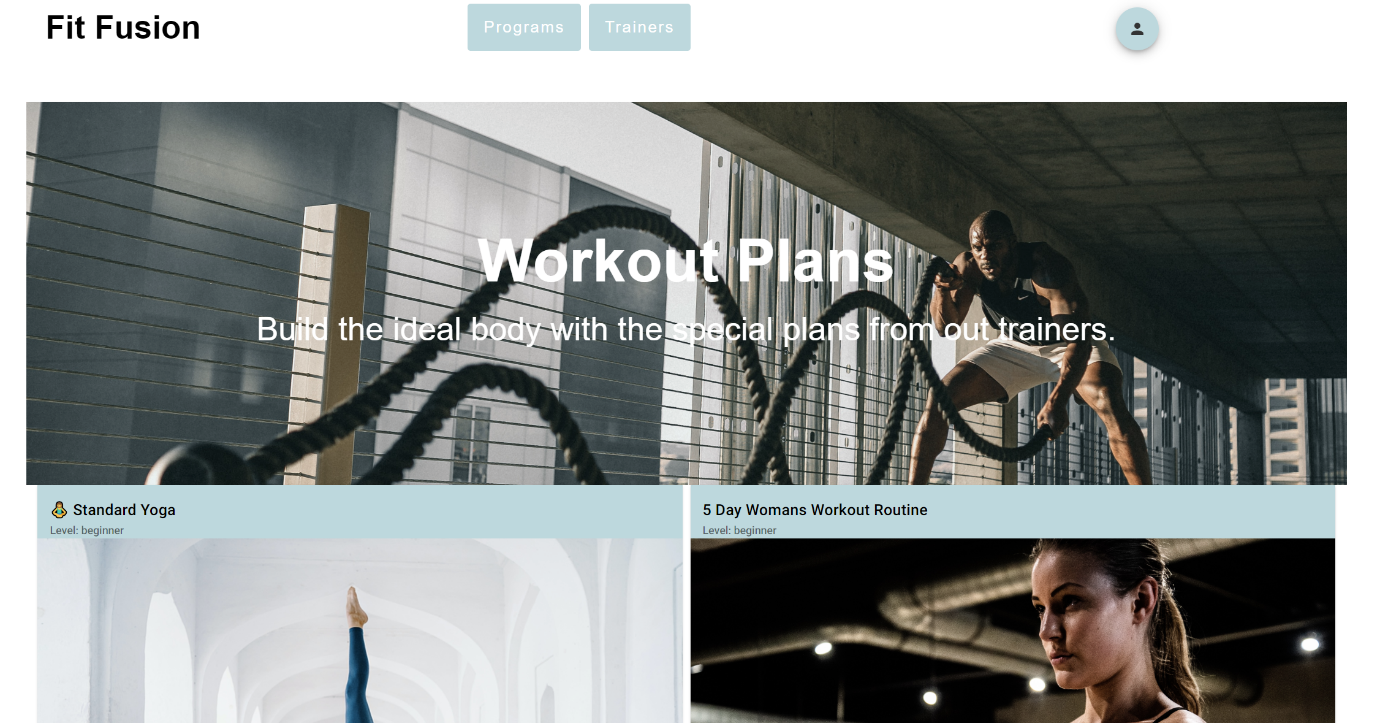
Последната справка показва стълбчета диаграма озаглавена „Chart“ и служи за Създаване на стълбчеста диаграма с посещаемостта по клиенти и дати.

Фиг. №12- Заявка показваща статистика за половин година посещаемост на клиенти под формата на стълбчета графика.

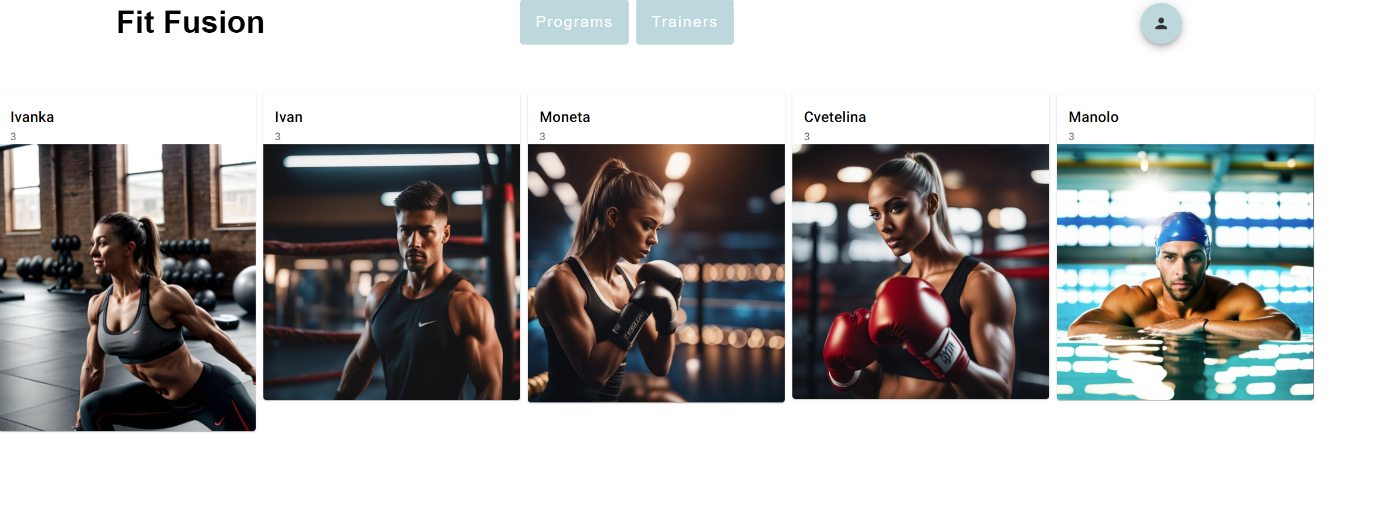
Следващия портал е на потребителска страна (Фиг №13, 14, 15). За потребителя е заложено той да може да има акаунт и да си следи тренировките и да може да си комуникира с тренира. За в момента клиента може да разглежда треньорите и програмите за трениране. За в бъдеще е предвидено да се имплементира тази функционалност защото до в момента само може да се види.

A person lifting weights in a gym

Description automatically generated

Фиг. №13- Начална страница за потребители на уеб приложението.

Фиг. №14- Страница за показване на фитнес програми в потребителския порта.

Фиг. №15- Страница показваща треньорите в фитнеса от страна на потребителския интерфейс.

## Изисквания към апаратното осигуряване

Изискванията към хардуерната конфигурация за уеб приложение могат да варират в зависимост от конкретните изисквания на самото приложение. Въпреки това, ето някои общи аспекти, които често се вземат предвид при проектирането на хардуерната инфраструктура за уеб приложение:

Процесор (CPU). Зависи от обема на обработката на данни и бизнес логиката на приложението. За по-големи и сложни уеб приложения може да бъде необходимо по-мощен процесор.

Памет (RAM). Количеството RAM е важно за ефективното изпълнение на приложението. По-големи приложения и тези, които обработват големи обеми данни, обикновено изискват повече RAM.

Хранилище (Storage). Размерът на хранилището зависи от обема на данните, които приложението трябва да съхранява. Може да се използва SSD за по-бърз достъп до данни.

Мрежова пропускателна способност. За уеб приложения, които изискват високо ниво на връзка с бази данни, облачни услуги или външни API, качеството на мрежовата свързаност е от съществено значение.

Безпека. Инфраструктурата трябва да бъде настроена за безопасност. Това включва защита от атаки, резервно копие на данни, криптиране на комуникацията и други мерки за сигурност.

Мащабируемост. Ако предвиждате растеж на трафика и потребителите, трябва да проектирате инфраструктурата така, че да бъде лесно мащабируема, може би с използването на облачни решения или клъстеризация.

Операционна система. Изборът на операционна система зависи от предпочитанията и изискванията на приложението. Linux базирани системи често се използват за уеб сървъри поради своята стабилност и сигурност.

Баланс на товара. За големи уеб приложения, където е необходимо да се управлява голям брой заявки, ефективното балансиране на товара е ключово, а това може да изисква специфични хардуерни или софтуерни решения.

Запомнете, че конкретните изисквания могат да се различават в зависимост от типа и характера на уеб приложението, както и от броя на потребителите и обема на данните, които обработва. Разработчиците и системните администратори трябва да проведат детайлен анализ и тестване, за да определят най-подходящата хардуерна конфигурация за конкретното уеб приложение.

# Резултати от тестване на системата. СЪЗДАВАНЕ на поне два Unit Теста

За тестване на програма в dot net core е нужно да се създаде паралелен проект само за тестване. В този ново създаден проект се систематизират всички нужни тестове.

Извинявам се за първоначалната погрешка в отговора, но важно е да отбележа, че няма нещо наречено "unitx" в света на .NET или в сферата на тестването. Вероятно има заблуда във въпроса. Ако имате предвид някакъв конкретен тестов фреймуърк, моля, предоставете правилното име, и ще бъда рад да ви помогна с информацията.

Въпреки това, ако говорим за използването на различни тестови фреймуърки в контекста на .NET, например, Microsoft Unit Testing Framework (MSTest) и xUnit, е важно да отбележим, че предпочитанията могат да зависят от конкретните нужди на екипа, предпочитанията на разработчиците и спецификите на проекта.

xUnit и MSTest са два популярни тестови фреймуърка за .NET, и двете са ефективни за използване. Ето някои отличителни характеристики на xUnit:

По-широко използване на концепцията за атрибути. xUnit използва атрибути за определяне на тестови случаи, тестове и факти, което може да го направи по-ясен и лесен за използване в сравнение с MSTest.

По-добра поддръжка на асинхронни тестове. xUnit е по-гъвкав по отношение на асинхронното тестване, което може да бъде предимство в съвременните приложения.

По-добра поддръжка на теории. xUnit предоставя теории за параметризирани тестове, които могат да бъдат полезни в различни сценарии.

Обаче MSTest също е стабилен и се интегрира добре с инструментите на Microsoft Visual Studio. Инструментите за тестване на Visual Studio например лесно интегрират MSTest, което може да е предимство за екипи, използващи Visual Studio.

Изборът между xUnit и MSTest обикновено зависи от предпочитанията на екипа, удобствата при използване и конкретните изисквания на проекта. Важно е да се отбележи, че и двете решения предоставят стабилна основа за писане на тестове и поддържат модерни подходи към тестване.

Резултатите от тестовете са положителни.

public class UnitTest1

{

    [Fact]

    public void Test1()

    {

        // Arrange

        var controller = new BalanceController();

        // Act

        var result = controller.GetmothlyVisistation();

        // Assert

        Assert.IsType<ActionResult<IEnumerable<DaylyVisitation>>>(result);

    }

    [Fact]

    public void Test2()

    {

        // Arrange

        var controller = new BalanceController();

        // Act

        var result = controller.GethalfAYear();

        // Assert

        Assert.NotEmpty(result);

        Assert.Equal(7,result.Count);

        Assert.IsType<List<ChartVisitorsMoths>>(result);

    }

}

# Използвана литература

1. Professional C# 7 and .NET Core 2.0" от Christian Nagel, et al
2. Pro ASP.NET Core MVC 2" от Adam Freeman
3. Programming C# 8.0" от Ian Griffiths и Matthew Adams
4. Entity Framework Core in Action" от Jon Smith
5. Документацията на Microsoft за .NET и ASP.NET Core
6. GitHub репозиториите на .NET и ASP.NET Core
7. Design Patterns ExplainedA New Perspective on Object-Oriented Design

От Alan Shalloway, James R. Trott · 2004

# **Приложение**

## Sql Скриптове:

/\*

Creation script for fit\_fusion

\*/

create table Client

(

id BIGINT NOT NULL,

first\_name VARCHAR(255) NOT NULL,

last\_name VARCHAR(255) NOT NULL,

sub\_from DATE NOT NULL,

sub\_to DATE NOT NULL,

sub\_type BIGINT NOT NULL,

added\_at DATE NOT NULL,

number\_renewing BIGINT NOT NULL,

email VARCHAR(255) NOT NULL,

phone\_number VARCHAR(16) NOT NULL,

PRIMARY KEY (id)

);

create table Role

(

id BIGINT NOT NULL,

title VARCHAR(255) NOT NULL,

PRIMARY KEY (id)

);

create table Salary

(

id BIGINT NOT NULL,

worker BIGINT NOT NULL,

date DATE NOT NULL,

amount BIGINT NOT NULL,

PRIMARY KEY (id)

);

create table Worker

(

id BIGINT NOT NULL,

first\_name VARCHAR(255) NOT NULL,

last\_name VARCHAR(255) NOT NULL,

first\_work\_day\_date DATE NOT NULL,

month\_salary BIGINT NOT NULL,

role BIGINT NOT NULL,

PRIMARY KEY (id)

);

create table Product\_group

(

id BIGINT NOT NULL,

name VARCHAR(255) NOT NULL,

PRIMARY KEY (id)

);

create table Product

(

id BIGINT NOT NULL,

name VARCHAR(255) NOT NULL,

amount BIGINT NOT NULL,

barcode BIGINT NOT NULL,

product\_group BIGINT NOT NULL,

last\_delivery\_date DATE NOT NULL,

future\_delivery\_date DATE NOT NULL,

PRIMARY KEY (id)

);

/\*

The duration is in moths

\*/

create table Subscription

(

id BIGINT NOT NULL,

name VARCHAR(255) NOT NULL,

price BIGINT NOT NULL,

duration BIGINT NOT NULL,

PRIMARY KEY (id)

);

create table Delivery

(

id BIGINT NOT NULL,

provider BIGINT NOT NULL,

date DATE NOT NULL,

price BIGINT NOT NULL,

amount BIGINT NOT NULL,

PRIMARY KEY (id)

);

create table Supplier

(

id BIGINT NOT NULL,

name VARCHAR(255) NOT NULL,

address VARCHAR(255) NOT NULL,

phone\_number VARCHAR(16) NOT NULL,

email VARCHAR(255) NOT NULL,

PRIMARY KEY (id)

);

create table Shift

(

id BIGINT NOT NULL,

name VARCHAR(255) NOT NULL,

PRIMARY KEY (id)

);

create table Work\_schedule

(

id BIGINT NOT NULL,

worker BIGINT NOT NULL,

shift BIGINT NOT NULL,

work\_hours BIGINT NOT NULL,

date DATE NOT NULL,

PRIMARY KEY (id)

);

create table Balance\_type

(

id BIGINT NOT NULL,

name VARCHAR(255) NOT NULL,

PRIMARY KEY (id)

);

/\*

Daly otchet madatory

\*/

create table Balance

(

id BIGINT NOT NULL,

date DATE NOT NULL,

type BIGINT NOT NULL,

PRIMARY KEY (id)

);

create table Equipment

(

id BIGINT NOT NULL,

name VARCHAR(255) NOT NULL,

amount BIGINT NOT NULL,

last\_delivered\_date DATE NOT NULL,

repaid BIGINT NOT NULL,

shelf\_life DATE NOT NULL,

replacement\_date DATE NOT NULL,

price\_at\_last\_delivery BIGINT NOT NULL,

PRIMARY KEY (id)

);

ALTER TABLE

Client ADD CONSTRAINT fk\_client\_sub\_type FOREIGN KEY(sub\_type) REFERENCES Subscription(id);

ALTER TABLE

Worker ADD CONSTRAINT fk\_worker\_month\_salary FOREIGN KEY(month\_salary) REFERENCES Salary(id);

ALTER TABLE

Worker ADD CONSTRAINT kf\_worker\_role FOREIGN KEY(role) REFERENCES Role(id);

ALTER TABLE

Product ADD CONSTRAINT fk\_product\_product\_group FOREIGN KEY(product\_group) REFERENCES Product\_group(id);

ALTER TABLE

Work\_schedule ADD CONSTRAINT fk\_worker\_schedule\_shift FOREIGN KEY(shift) REFERENCES Shift(id);

ALTER TABLE

Balance ADD CONSTRAINT fk\_worker\_schedule\_shift FOREIGN KEY(type) REFERENCES Balance\_type(id);

insert into subscription(id, name, price, duration)

VALUES(1,'Standard morning monthly plan',65,1);

insert into subscription(id, name, price, duration)

VALUES(2,'Standard evening monthly plan',65,1);

insert into subscription(id, name, price, duration)

VALUES(3,'Standard monthly plan',85,1);

insert into subscription(id, name, price, duration)

VALUES(4,'Premium morning monthly plan',85,1);

insert into subscription(id, name, price, duration)

VALUES(5,'Premium evening monthly plan',85,1);

insert into subscription(id, name, price, duration)

VALUES(6,'Premium monthly plan',95,1);

insert into subscription(id, name, price, duration)

VALUES(7,'Standard morning personal trainer plan',85,1);

insert into subscription(id, name, price, duration)

VALUES(8,'Standard evening personal trainer plan',85,1);

insert into subscription(id, name, price, duration)

VALUES(9,'Standard personal trainer plan',95,1);

insert into subscription(id, name, price, duration)

VALUES(10,'Premium morning personal trainer plan',105,1);

insert into subscription(id, name, price, duration)

VALUES(11,'Premium evening personal trainer plan',105,1);

insert into subscription(id, name, price, duration)

VALUES(12,'Premium personal trainer plan',115,1);

/\* quoter plans are -20%\*/

insert into subscription(id, name, price, duration)

VALUES(13,'Standard morning quoter plan',159,3);

insert into subscription(id, name, price, duration)

VALUES(14,'Standard evening quoter plan',159,3);

insert into subscription(id, name, price, duration)

VALUES(15,'Standard quoter plan',204,3);

insert into subscription(id, name, price, duration)

VALUES(16,'Premium quoter plan',228,3);

/\*

CLIENTS

\*/

insert into Client(id, first\_name, last\_name, sub\_from, sub\_to, sub\_type, added\_at, number\_renewing, email, phone\_number)

VALUES (1,'Ivan','Ivanov','2023-10-28','2023-11-28',1,'2023-03-28',4,'Vanko43@mail.com','0897674673');

insert into Client(id, first\_name, last\_name, sub\_from, sub\_to, sub\_type, added\_at, number\_renewing, email, phone\_number)

VALUES (2,'Maria','Ilieva','2023-11-22','2023-12-22',3,'2022-01-22',24,'Mariika@mail.com','089743243');

insert into Client(id, first\_name, last\_name, sub\_from, sub\_to, sub\_type, added\_at, number\_renewing, email, phone\_number)

VALUES (3,'Zaro','Panov','2023-11-12','2023-12-12',2,'2023-03-12',9,'ZaroEgotin@mail.com','089234343');

insert into Client(id, first\_name, last\_name, sub\_from, sub\_to, sub\_type, added\_at, number\_renewing, email, phone\_number)

VALUES (4,'Ivanka','Yordanova','2023-12-03','2024-01-03',4,'2022-01-03',25,'QkaVanka@mail.com','089795343');

insert into Client(id, first\_name, last\_name, sub\_from, sub\_to, sub\_type, added\_at, number\_renewing, email, phone\_number)

VALUES (5,'Yuri','Boika','2023-10-15','2024-01-15',13,'2023-10-15',2,'Turikata@mail.com','086423424');

insert into Client(id, first\_name, last\_name, sub\_from, sub\_to, sub\_type, added\_at, number\_renewing, email, phone\_number)

VALUES (6,'Kamila','Yurdanova','2022-10-28','2023-01-28',16,'2022-10-28',4,'KamilaNaMogila@mail.com','084236743');

insert into Client(id, first\_name, last\_name, sub\_from, sub\_to, sub\_type, added\_at, number\_renewing, email, phone\_number)

VALUES (7,'Ceco','Kokumarov','2023-09-21','2023-12-21',13,'2023-09-21',4,'CecoBoreco23@mail.com','089483734');

/\*

Role

\*/

insert into Role(id, title) VALUES (1,'Boss');

insert into Role(id, title) VALUES (2,'Trainer');

insert into Role(id, title) VALUES (3,'Cashier');

insert into worker(id, first\_name, last\_name, first\_work\_day\_date, month\_salary, role)

VALUES (1,'Ivaylo','Rumenov','2022-01-01',3300,1);

insert into worker(id, first\_name, last\_name, first\_work\_day\_date, month\_salary, role)

VALUES (2,'Damqn','Mihailov','2022-01-01',1300,3);

insert into worker(id, first\_name, last\_name, first\_work\_day\_date, month\_salary, role)

VALUES (3,'Mihail','Hristov','2022-01-01',1900,2);

insert into worker(id, first\_name, last\_name, first\_work\_day\_date, month\_salary, role)

VALUES (4,'Ivanka','Milanova','2022-03-03',1900,2);

insert into worker(id, first\_name, last\_name, first\_work\_day\_date, month\_salary, role)

VALUES (5,'Ivan','Zlatareb','2022-10-01',1400,2);

insert into worker(id, first\_name, last\_name, first\_work\_day\_date, month\_salary, role)

VALUES (6,'Cvetelina','Georgieva','2022-10-01',1800,2);

insert into worker(id, first\_name, last\_name, first\_work\_day\_date, month\_salary, role)

VALUES (7,'Manolo','Yordanov','2022-11-11',1960,2);

insert into worker(id, first\_name, last\_name, first\_work\_day\_date, month\_salary, role)

VALUES (8,'Monika','Peneva','2023-01-01',1300,2);

insert into Salary(id, worker, date, amount) values (1,1,'2023-12-01',3300);

insert into Salary(id, worker, date, amount) values (2,2,'2023-12-01',1300);

insert into Salary(id, worker, date, amount) values (3,3,'2023-12-01',1900);

insert into Salary(id, worker, date, amount) values (4,4,'2023-12-01',1900);

insert into Salary(id, worker, date, amount) values (5,5,'2023-12-01',1400);

insert into Salary(id, worker, date, amount) values (6,6,'2023-12-01',1800);

insert into Salary(id, worker, date, amount) values (7,7,'2023-12-01',1960);

insert into Salary(id, worker, date, amount) values (8,8,'2023-12-01',1300);

/\*

Products

\*/

insert into Product\_group(id, name) VALUES (1,'Protein');

insert into Product\_group(id, name) VALUES (2,'Vitamin');

insert into Product\_group(id, name) VALUES (3,'Drink');

insert into Product\_group(id, name) VALUES (4,'Coffee');

insert into Product\_group(id, name) VALUES (5,'Merchandise');

insert into Product(id, name, amount, barcode, product\_group, last\_delivery\_date, future\_delivery\_date)

VALUES(1,'Super Humen Protein',10,834765865734,1,'2023-09-10','2024-02-10');

insert into Product(id, name, amount, barcode, product\_group, last\_delivery\_date, future\_delivery\_date)

VALUES(2,'Vitamin A to Z',2,84234234,2,'2023-09-10','2024-02-10');

insert into Product(id, name, amount, barcode, product\_group, last\_delivery\_date, future\_delivery\_date)

VALUES(3,'Coka cola',32,908263946,3,'2023-11-20','2023-12-28');

insert into Product(id, name, amount, barcode, product\_group, last\_delivery\_date, future\_delivery\_date)

VALUES(4,'Lavasa Gold',1,23423423,4,'2023-12-01','2023-12-15');

insert into Product(id, name, amount, barcode, product\_group, last\_delivery\_date, future\_delivery\_date)

VALUES(5,'Mens tank top',8,48568739448736,5,'2023-06-10','2024-06-10');

/\*

Supplier

\*/

insert into supplier(id, name, address, phone\_number, email)

VALUES (1,'Ivan Gym supplies','Hristobotel23 Sofiq','089865785','IvanGymSupplies@mail.com');

insert into supplier(id, name, address, phone\_number, email)

VALUES (2,'Roshkov','Kapintan Marian Zlatov 23 Sofiq','08842394','Roshkov@mail.com');

insert into supplier(id, name, address, phone\_number, email)

VALUES (3,'Tim','General Parensov Varna','089423425','Tim@mail.com');

/\*

Product delivery

\*/

insert into Delivery(id, date, price, amount, supplier)

VALUES(1,'2023-09-10',2100,20,1);

insert into Delivery(id, date, price, amount, supplier)

VALUES(2,'2023-09-10',320,20,1);

insert into Delivery(id, date, price, amount, supplier)

VALUES(3,'2023-11-20',40,42,3);

insert into Delivery(id, date, price, amount, supplier)

VALUES(4,'2023-12-01',43,4,3);

insert into Delivery(id, date, price, amount, supplier)

VALUES(5,'2023-06-10',240,20,2);

/\*

equipment delivery

\*/

insert into Delivery(id, date, price, amount, supplier)

VALUES(6,'2022-02-01',310,4,1);

insert into Delivery(id, date, price, amount, supplier)

VALUES(7,'2022-02-01',220,4,1);

insert into Delivery(id, date, price, amount, supplier)

VALUES(8,'2022-02-01',40,1,1);

insert into Delivery(id, date, price, amount, supplier)

VALUES(9,'2022-02-01',1300,1,1);

insert into Delivery(id, date, price, amount, supplier)

VALUES(10,'2022-02-01',540,1,1);

insert into equipment(id, name, amount, last\_delivered\_date, repaid, shelf\_life, replacement\_date, price\_at\_last\_delivery, dilivery)

VALUES(1,'Dumbell 20kg',4,'2022-02-01',0,'2037-02-01','2032-02-01',310,6);

insert into equipment(id, name, amount, last\_delivered\_date, repaid, shelf\_life, replacement\_date, price\_at\_last\_delivery, dilivery)

VALUES(2,'Dumbell 5kg',4,'2022-02-01',0,'2037-02-01','2032-02-01',220,7);

insert into equipment(id, name, amount, last\_delivered\_date, repaid, shelf\_life, replacement\_date, price\_at\_last\_delivery, dilivery)

VALUES(3,'Barbell 20kg',1,'2022-02-01',0,'2037-02-01','2032-02-01',143,8);

insert into equipment(id, name, amount, last\_delivered\_date, repaid, shelf\_life, replacement\_date, price\_at\_last\_delivery, dilivery)

VALUES(4,'Skrepec',1,'2022-02-01',2,'2027-02-01','2028-02-01',1300,9);

insert into equipment(id, name, amount, last\_delivered\_date, repaid, shelf\_life, replacement\_date, price\_at\_last\_delivery, dilivery)

VALUES(5,'Bench press',1,'2022-02-01',1,'2037-02-01','2032-02-01',540,10);

/\*

balance

\*/

insert into balance\_type(id, name) VALUES (1,'Positive');

insert into balance\_type(id, name) VALUES (2,'Negative');

insert into balance(id, date, type) VALUES(1,'2022-02-01',1);

insert into balance(id, date, type) VALUES(2,'2022-02-02',1);

insert into balance(id, date, type) VALUES(3,'2022-02-03',1);

insert into balance(id, date, type) VALUES(4,'2022-02-04',1);

insert into balance(id, date, type) VALUES(5,'2022-02-05',1);

insert into balance(id, date, type) VALUES(6,'2022-02-06',1);

insert into balance(id, date, type) VALUES(7,'2022-02-07',1);

insert into balance(id, date, type) VALUES(8,'2022-02-08',2);

/\*

work\_schedule

\*/

insert into shift(id, name) VALUES (1,'Morning');

insert into shift(id, name) values (2,'Evening');

INSERT INTO work\_schedule(id, worker, shift, work\_hours, date)

VALUES (1,1,1,8,'2022-02-01');

INSERT INTO work\_schedule(id, worker, shift, work\_hours, date)

VALUES (2,2,1,8,'2022-02-01');

INSERT INTO work\_schedule(id, worker, shift, work\_hours, date)

VALUES (3,3,1,8,'2022-02-01');

INSERT INTO work\_schedule(id, worker, shift, work\_hours, date)

VALUES (4,4,2,8,'2022-02-01');

INSERT INTO work\_schedule(id, worker, shift, work\_hours, date)

VALUES (5,5,2,8,'2022-02-01');

INSERT INTO work\_schedule(id, worker, shift, work\_hours, date)

VALUES (6,5,2,8,'2022-02-01');

INSERT INTO work\_schedule(id, worker, shift, work\_hours, date)

VALUES (7,6,2,8,'2022-02-01');

INSERT INTO work\_schedule(id, worker, shift, work\_hours, date)

VALUES (8,1,1,8,'2022-02-02');

INSERT INTO work\_schedule(id, worker, shift, work\_hours, date)

VALUES (9,2,1,8,'2022-02-02');

INSERT INTO work\_schedule(id, worker, shift, work\_hours, date)

VALUES (10,3,1,8,'2022-02-02');

INSERT INTO work\_schedule(id, worker, shift, work\_hours, date)

VALUES (11,4,2,8,'2022-02-02');

INSERT INTO work\_schedule(id, worker, shift, work\_hours, date)

VALUES (12,5,2,8,'2022-02-02');

INSERT INTO work\_schedule(id, worker, shift, work\_hours, date)

VALUES (13,5,2,8,'2022-02-02');

INSERT INTO work\_schedule(id, worker, shift, work\_hours, date)

VALUES (14,6,2,8,'2022-02-02');

INSERT INTO work\_schedule(id, worker, shift, work\_hours, date)

VALUES (15,1,1,8,'2022-02-03');

INSERT INTO work\_schedule(id, worker, shift, work\_hours, date)

VALUES (16,2,1,8,'2022-02-03');

INSERT INTO work\_schedule(id, worker, shift, work\_hours, date)

VALUES (17,3,1,8,'2022-02-03');

INSERT INTO work\_schedule(id, worker, shift, work\_hours, date)

VALUES (18,4,2,8,'2022-02-03');

INSERT INTO work\_schedule(id, worker, shift, work\_hours, date)

VALUES (19,5,2,8,'2022-02-03');

INSERT INTO work\_schedule(id, worker, shift, work\_hours, date)

VALUES (20,5,2,8,'2022-02-03');

INSERT INTO work\_schedule(id, worker, shift, work\_hours, date)

VALUES (21,6,2,8,'2022-02-03');