**ДИПЛОМЕН ПРОЕКТ**

**ЗА ДЪРЖАВЕН ЗРЕЛОСТЕН ИЗПИТ**

по професия код 481030 „Приложен програмист“

специалност код 4810301 Приложно програмиране“

ТЕМА: „СИСТЕМА ЗА ПОРЪЧКА И ДОСТАВКА НА ХРАНА И РЕЗЕРВАЦИИ“

Автор:

Християн Христов Кимрянов, клас XII В

Ръководител:

Даниела Стаматова

БургасСЪДЪРЖАНИЕ

[1 Увод 2](#_Toc163480931)

[2 Цели и обхват на софтуерното приложение 3](#_Toc163480932)

[3 Анализ на решението 3](#_Toc163480933)

[3.1 Потребителски изисквания и работен процес 3](#_Toc163480934)

[3.2 Примерен потребителски интерфейс 6](#_Toc163480935)

[3.3 Диаграми на анализа 6](#_Toc163480936)

[3.4 Модел на съдържанието / данните 14](#_Toc163480937)

[4 Дизайн 15](#_Toc163480938)

[4.1 Реализация на архитектурата на приложението 15](#_Toc163480939)

[4.2 Описание на слоевете, предназначението им, библиотеки и методи включени в съответния слой. 15](#_Toc163480940)

[4.3 Организация и код на заявките към база от данни 24](#_Toc163480941)

[4.4 Наличие на потребителски интерфейс (конзолен, графичен, уеб) 26](#_Toc163480942)

[5 Ефективност и бързодействие на решението 29](#_Toc163480943)

[6 Тестване 30](#_Toc163480944)

[7 Заключение и възможно бъдещо развитие 31](#_Toc163480945)

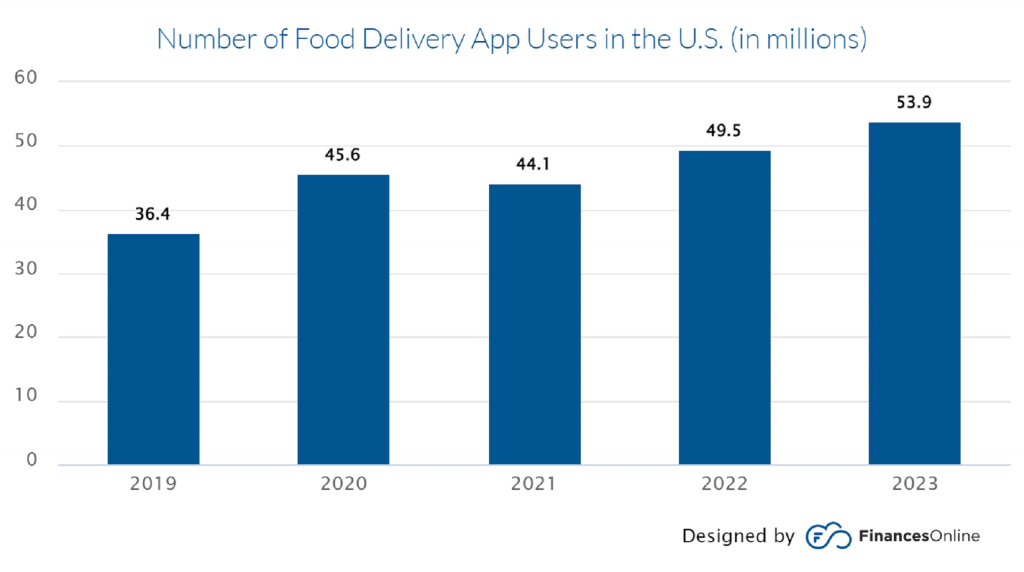
[8 Използвани литературни източници и Уеб сайтове 31](#_Toc163480946)

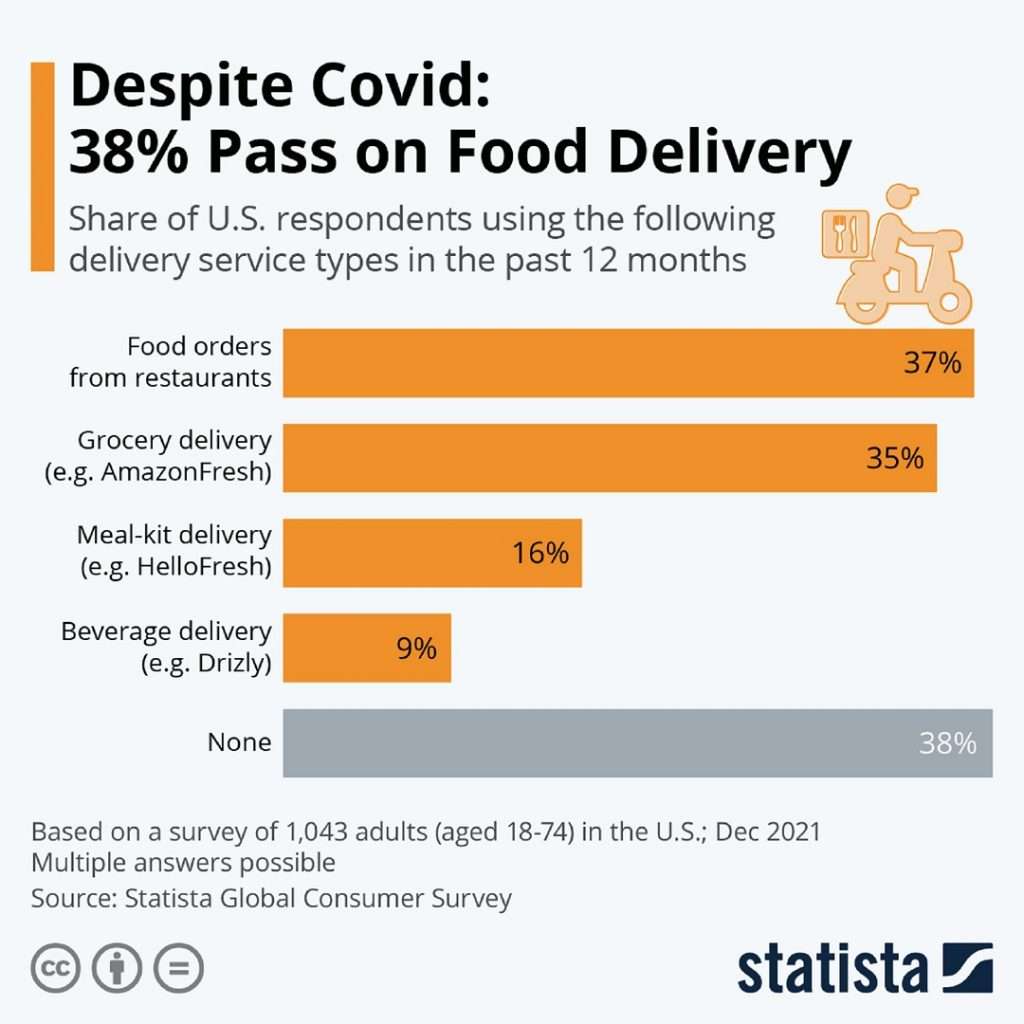
[9 Приложения 31](#_Toc163480947)

# Увод

Темата на моя проект е разработване на система, с която хората могат да си поръчат храна от вкъщи и тя да им бъде доставена без те изобщо да се налага да излизат навън или да направят резервация, ако предпочитат да излязат в приятната обстановка на заведение.

Проблемът, който съм решил да засегна в този проект, е желанието на хората да получат бърза и вкусна храна без да се налага да излизат от техните домове и да трябва да приготвят нещо, което може и да не се получи. Консумацията на готова храна ще спести време и ресурси, необходими за приготвяне в домашни условия.

Проблемът е значително актуален, тъй като в съвременното забързано ежедневие все повече хора предпочитат да консумират вкусна и разнообразна храна, приготвена от специалисти, пред това да отделят значително време и средства за приготвяне в домашни условия. Има и други решения на този проблем, но за разлика от тях това, предоставя опция за резервация на маса в дадения обект, за тези, които една вечер навън ще бъде доста по стойностна от колкото да си стоят на вкъщи.



Чрез средата за моделиране Lucid Chart е възможно да се представят някои от основните за проекта части, под формата на различни диаграми, като това какви класове са използвани в системата, какви таблици има базата данни, как те са свързани една с друга и какви атрибути има всяка от тях и от какъв тип са. Друга част от проекта, която може да се опише чрез диаграмите са това какви роли има в системата и какво може да прави всяка една от тях и също така в каква последователност се случва това.

В документацията са описани целите на това приложение, по-подробна информация за използваните технологии, някой от по-горе споменатите типове диаграми, потребителския графичен интерфейс, тестовете и стратегията за тестване, как се представя нашето решение спрямо други подобни и др. и накрая заключение от цялата представена информация за проекта от този файл.

# Цели и обхват на софтуерното приложение

Целта на проекта е да се разработи система за подпомагане на хората за тяхното изхранване, като им спести време и средства за приготвяне на храна. Разработената от мен система една страна предоставя възможност за избор и поръчка на любими ястия за вкъщи, а от друга да направят резервация в предпочитан от тях ресторант.

Цялостната идея на приложението е да улесни работата на хората по отношение на приготвянето на бърза и вкусна храна и да им даде възможност да намерят разнообразни и непознати за тях заведения за хранене в техния град, като това води и до откриването на нови кухни. Друга идея, която е осъществена в проекта е тази за улесняване на запазване на маса.

Една важна част при определянето на целите е каква ще е нашата целева група. След като си зададем този въпрос на повечето ни идва на ум първо по-младите хората. Целевата аудитория на това приложение са именно те. По-голямата част от младите хора в днешно време предпочитат да се развиват и да прекарват колкото се може повече време в това си начинание и да не се занимават с неприсъщи за тях и отнемащи много време занимания. Добре, но не може по цял ден да седиш и да правиш едно и също без да си дадеш време за почивка и отдих, а какво е по-добро от това да излезете навън с приятели на някое приятно заведение. Системата разрешава проблема и на тези хора.

Основните функционалности, които предоставя система са избиране на ресторант, от който имаш възможността да си поръчаш любима храна по каталог или да си резервираш маса в заведението или извън него. Има няколко различни нива на достъп като системен потребител, администратор, служител, мениджър и доставчик.

# Анализ на решението

## Потребителски изисквания и работен процес

В тази точка от документацията са представени различните видове нива на достъп и техните функционалности. Това е визуализирано на следната диаграма:

Картина, която съдържа текст, диаграма, скица, рисунка

Описанието е генерирано автоматично

От лявата страна може да забележите, че имаме 5 основни роли и една базова, като базовата представлява основните неща, които може да прави всеки потребител без значение каква роля има. Всеки нов акаунт се регистрира с роля системен потребител, нужно е само да въведете име, фамилия, имейл и парола. Също така всеки един потребител може да влиза в системата, чрез въвеждане на имейл и парола. Друга функционалност, която е достъпна за всяка една роля е възможността за управление на профила. В тази опция се включват смяна на лична информацията – адрес и имена, смяна на паролата – трябва да се напише новата парола и след това старата парола и ако вече не искате да сте част от тази система, за да изтриете своя акаунт е необходимо само да си въведете имейл.

Първата роля е системен потребител*.* След като потребителят избере ресторант или друг вид заведение той има опции да направи резервация за дадено място. При това негово действие той трябва да избере дата и час, брой посетители, които ще бъдат включени в резервацията и това дали масата да бъде на закрито или на открито. Друга опция, която има потребителя е да добави ястия към поръчка, ако вече има това ястие в поръчката ново въведената бройка ще се добави към вече съществуващата, но ако нямаме поръчка се създава нова, в която има ястието. След като сме си направили резервация можем да отидем и да видим всички резервации, които сме направили и ако някоя, която все още не е минала, ние имаме опцията да я изтрием. Едно от нещата, което може да прави системният потребител е да управлява своите поръчки. В това се включва финализиране на поръчката, премахване на ястие цялостно от поръчката и промяна на броя на дадено ястие.

Следващата роля в системата е администраторът. Той има функциолността да управлява ресторантите. Под това се разбира да създава нови ресторанти или обекти, като при това създаване той включва име, телефон за контакт, адрес и цена на доставка, премахването им на цяло от системата, промяна на данните въведени при създаване и също така да назначава мениджър. При последното действие един системен потребител променя своята роля на мениджър.

Разликите между ролите мениджър и служител са малко. Единствената по съществена е, че мениджърът може да назначава служители и доставчици за ресторанта или обекта, към който е назначен мениджъра. След като пояснихме тази разлика ще разгледаме какво може да прави служителят, като всичко написано за служителя се отнася и за мениджъра. Един служител може да преглежда направените за деня резервации и също така да променя статусът на финализираните поръчки, с това негово действие уведомява потребителя, че неговата поръчка чака да бъде направена и приета от доставчик. Друга функционалност, която е позволена на тази роля е възможността да преглежда ястията, които са част от дадената поръчка.

Последната роля, която ще разгледаме, е доставчик. Той има право да приема поръчки и след като е доставил поръчката да я изпише, с което тя повече няма да се визуализира, а когато някой системен потребител иска да види такава поръчка, тя ще има статут на доставена и датата и часът, в което това е направено ще се изпишат.

## Примерен потребителски интерфейс

В тази подточка се разглежда примерният интерфейс на програмата от страна на различните нива за достъп, като ще бъдат представени само основните неща за всяка роля.

A plate of food on a table

Description automatically generatedСлед като един системен потребител влезе в системата той ще бъде отведен в началната страница, където са изложени малка част от заведенията намиращи се в градът. След това ще има възможността да избере да види повече информация за конкретно заведение. С това си действие ще бъде отведен в нова страница от която ще може да види предлаганите ястия и да си резервира маса. Други опции, които притежава самият потребител е да управлява своите поръчки и резервации. Той може да променя броят на различните ястия или напълно да ги премахне от поръчката.

Следващите две роли притежават почти едни и същи правомощия. Става дума за мениджърите и служителите. Разликата идва в това, че мениджърът може да назначава служители и доставчици. При влизането си в системата те биват отведени в страницата с поръчките. Други опции, които са им позволени са добавяне, премахване и промяна на ястията. Последното нещо, което могат да изпълняват те е приемане на поръчка, с което свое действие дават възможност и на следващата роля да изпълнява своята дейност.

A screenshot of a computer

Description automatically generated

Следващата роля, която ще бъде разгледа е доставчикът. Доставчикът може да приеме да достави дадена поръчка след като тя е била одобрена от мениджър или някой друг служител.

Последната и най-важна роля е тази на администраторът. Основното и единствено нещо, което може да прави е да създава заведения в системата и да ги управлява. Към това негово правомощие се включват и промяна на информацията за самият ресторант, премахването му и назначаването на нов мениджър.

## Диаграми на анализа

Следната ER диаграма представлява таблиците в базата данни, която е създадена чрез един от най-популярните ORM в наши дни, а именно Entity Framework, като първо съм създал моделите в проекта и след това таблиците, това се нарича code first.

AspNetUsers е таблица, която съхранява информация за потребителите на системата, като това включва всички потребители без значение от ролята им, като в нея се включват името, фамилията, адреса, външен ключ, който сочи към първичния ключ на таблицата с ресторанти, който ако потребителят не е мениджър, служител или доставчик ще липсва стойност към него (стойността му в полето ще бъде NULL). Другите полета, които се намират в нея са добавени при наследяването на един специфичен клас предоставен в една от библиотеките на Microsoft. Тези полета, включват някои от често срещаните необходими за една съвременна система полета, като имейл, потвърждение на имейл, хеширана парола и др.

Една от следващите таблици в базата данни е тази за ресторантите или различните обекти, които предоставят опцията за поръчка до домове. Полетата съдържащи се в нея са стандартните за повечето такива приложения като име на самото място, адрес, телефон за контакт, снимка на самото заведение и цена за доставка. Различното тук са последните четири полета. В тях се запазва информация, като кой е създал новия запис, кога е създаден, кой последно е правил промени по конкретния запис и кога е било това.

Резервациите (Bookings) е следващата таблица. В нея се съхраняват основната информация за направената от нас резервация. Полетата, които се намират в нея са колко хора ще присъстват, кога и в колко, дали ще е на закрито или на открито пространство в заведението. Последните две полета в таблицата са външни ключове, които сочат към първичните ключове на таблицата за потребителите (AspNetUsers) и тази за самите заведение (Restaurants). От това разбираме чия е резервацията и за кое конкретно заведение е.

Друга от таблиците, която е свързана по някакъв начин с потребителите е за предлаганите ястия (Dishes). В нея се съхранява информация относно това какво е името на конкретното ястие, колко струва, към коя категория храни се отнася, информация за самото ястие, която може да включва всичко, което е необходимо да знаем и не е представено като полета в таблицата. Както при ресторанта, така и тук може да се види кой е добавил ново ястие и кога го е направил, кои последно е променял записът и кога. И последното поле на тази таблица представлява външен ключ, той сочи към първичния ключ на таблицата със заведения (Restaurants)

Таблицата поръчки (Orders) съдържа основна информация за всички поръчки, като дата, на която поръчката е направена и също така на която е била доставена, текущото състояние на поръчката. Другите полета са три външни ключа, като два от тях сочат към таблицата с потребителите (AspNetUsers), а другото към тази със заведенията (Restaurants). Първите две показват съответно кой е направил и кой е доставил поръчката. Последният външен ключ показва на кой ресторант или заведение принадлежи тази поръчка.

При стандартното проектиране на бази данни, когато имаме връзка много към много трябва да се създаде таблица, която свързва двете отделни таблици. Това се наречена bridge table. Следващата таблица е подобна. Таблицата OrdersDishes има два външни ключа, които сочат към поръчките и към ястията. Ето къде едва разликата между стандартната свързваща и тази, за да не се налага таблицата да съхранява едни и същи записи, имаме полето брой (Count), което съхранява броя на даденото ястие в съответната поръчка. И накрая имаме един самостоятелен първичен ключ, а не както при някои стандарти свързващи таблици за създаване на композитен първичен ключ от външния ключ за едната и другата.

Таблицата AspUserRoles съдържа информация за основните роли в тази система. Тя има няколко полета като основните са името на самата роля и нормализираната форма на това име. И също така полето печат за едновременност (ConcurrencyStamp), което служи за избягване на проблеми, които могат да настъпят по време на промяна на данните в таблицата.

AspNetUserRoles служи като свързваща таблица между таблиците AspNetRoles и AspNetUsers. Тя показва каква роля притежава даден потребител, като той може да има повече от една роля

В AspNetRolesClaims с представени всички правомощия на различните роли. Какво представлява един запис в тази таблица? В един запис може да се види външен ключ, който сочи към конкретна роля от таблицата AspNetRoles, от какъв тип е самото правомощие.

В AspNetUserTokens се съдържа информация за ключовете(tokens) за необходимите операции като потвърждение на имейли, нулиране на пароли, двуфактурна ауторизация и др.

В AspNetUserLogins се създава запис, когато някой се впише чрез външен доставчик. Чрез тази таблица се създава връзка с неговата външна идентичност с тази в приложението. Пример за това, е когато използваме опцията за влиза с Facebook, Google и т.н.

След като представих какво съдържа всяка таблица, следва описание в коя нормална форма е базата данни. Таблиците в базата данни са в трета нормална форма. Какво означава това и какво представляват нормалните форми до трета:

* Първа нормална форма (1NF) – за да бъде нашата таблица в първа нормална форма във всяко поле на всеки отделен запис трябва да има само една и единствена стойност, без никакви изключения.
* Втора нормална форма (2NF) – за да бъде изпълнена втора нормална форма трябва задължително таблицата да е в първа нормална и всеки запис да зависи на първичен ключ.
* Трета нормална форма (3NF) – за да бъде изпълнена тази нормална форма таблицата първо трябва да има втора нормална и който и да е непървичен ключ да не зависи на друг такъв.

Всяка една от таблиците има по-една единствена стойност във всяко свое поле и първичен ключ, които да отговаря за записа. И последното и най-важно в последната версия на базата данни няма непървичен ключ, който да зависи на друг такъв.

A computer screen shot of a computer program

Description automatically generated

В следващата част описва какво представляват класовете и как са свързани по между си. Едно допълнение следните клас диаграми са създадени с помощта на Visual Studio и някои от връзките типични за стандартните клас диаграми липсват.

Следната клас диаграма представлява различните класове намиращи се в слоя за достъп на данни. Първото нещо, което се забелязва, че една част от класовете наследяват абстрактния клас BaseEntity, което съдържа само едно публично свойство. Това свойство е идентификаторът на всяка отделна единица. Този клас също така се наследява от друг абстрактен клас – AuditableEntity, което съдържа още четири публични полета. Полетата отговарят за съхранението на това кой и кога е създал дадената единица и кой и кога последно е извършил последната промяна. AuditableEntity се наследява единствено от класът Dish. Тези два абстрактни класа имплементират по един интерфейс съответно IEntity и IAuditableEntity.

Следващото, което разглеждам са различните единици. Класът Booking за нашата резервация. В него има няколко публични свойства, които отговарят за кога ще бъде резервацията, дали ще е вътре в заведението или не и броя на хората. Има още няколко свойства, които показват идентификатора на потребителя, който я е направил и в кое заведение. Има и още две, които са в случай, че искаме да вземем и самите обекти с тези идентификатори. Това не се отнасям само за този клас. Всеки клас, който има различни свойства за идентификаторите в него има и по още едно за обекта с този идентификатор.

Класът Order за нашата резервация. Всички свойства в него са публични и са както следват – кога е завършена поръчката, кога е доставена, идентификаторите на потребителят, на заведението и на доставчика, текущия статус на поръчката и колекция от всички ястия и броят им за конкретната поръчка (колекция от обекти от класът OrdersDishes).

Следващият клас е OrdersDishes. Той се състой от идентификатор на ястието и на поръчката. Също така има и едно свойство, което е за броя на конкретното ястие за дадената поръчка.

Класът Restaurant съдържа в себе си основните неща, които трябва да знаем за едно заведение, като наименованието, адреса, каква е цената за доставка, снимка на самото място и телефон, на който да се обадиш при необходимост. Друго, което има са различни колекции са резервациите, ястията и др.

Класът за ястията е Dish, в който може да се видят следните свойства – изображение, категория, информация за ястието, цената, наименованието и идентификатора за ресторанта.

Следващият клас, който се разгледаме е ApplicationUser. Този клас наследява IdentityUser<>, който съдържа различни полета, отнасящи се до упълномощаването и удостоверяването. Други свойства, които не се отнасят до IdentityUser са: адреса на потребителя, неговото име и фамилия и идентификатора за ресторанта, които ако потребителят не е служител остава NULL. Друго, което можем да видим са колекциите за резервациите и поръчките.

Един от последните класове е ApplicationDbContext. Първото нещо, което трябва да се каже за този клас е, че той наследява класа IdentityDbContext, което помага при създаването на базата данни. В ApplicationDbContext се съдържат няколко публични свойства DbSet<> всяко с различен клас от тези изброени по-горе. Има и няколко метода, които се използват при запазването на свойствата от AuditableEntity. Това са свойствата CreatedBy, CreatedOn, UpdateBy и UpdateOn.

Останалите класове с изключение на DependencyInjection се използват за конфигуриране на самите единици, за да може при създаването на базата данни всички добри практики да бъдат спазени и тя да бъде в правилната нормална форма.

DependencyInjection е статичен клас, който в себе си съдържа един статичен метод, който служи за конфигурирането на базата данни.

A screenshot of a computer screen

Description automatically generated

Следната клас диаграма представлява класовете, които се намират в основните услуги (services), които се използват или помагат за цялостната работа на приложението.

Първите два класа са статични. Първият съдържа един метод ConfidureJwt(), който се използва за конфигуриране на JSON web token, чрез който потребителят може да се удостовери. Другият е за регистриране на всички други класове. В него също има един единствен метод.

Първият от останалите класове е за поръчките на потребителят. Класът има две поверителни полета, които отговарят за контекстът (context) и за управлението на потребители (userManager). В този клас има основните CRUD операции. Като трябва да се обърне внимание на това, че има няколко метода за извличане от базата – спрямо потребителския, този на ресторанта или и двата идентификатора. Има и само за една единствена поръчка. Има и два метода за промяна, като единият е само за основната информация на поръчката, а другият е за промяна да броя на дадено ястие. Този клас имплементира IOrderServices.

Следващият клас е CurrentUser, който има едно публично свойство, което е за идентификатора на текущия удостоверен потребител.

Почти всички останали класове също както първия, който беше разгледан, от тази група имат основните методи, които изпълняват CRUD операции, заради това да споменава на всеки един клас, че ги изпълнява. Ако има повече от един метод за извличане, добавяне, изтриване или промяна ще се спра на основните разлики.

Класът UserServices имплементира IUserServices. В класа липсва метод за създаване на потребител, защото този метод принадлежи към други услуги (services). Има два метода за промяна, като единият е за цялостната информация на потребителя, а другият – промяна на паролата.

RestaurantServices е клас, в който няма методи, които да отговарят за подобна CRUD операция. Единственото по специално е при извличането. Ако потребителят, които иска да вземе ресторантите е администратор или няма зададен адрес, се връщат всички ресторанти. Ако има зададен адрес се връщат всички в неговия град. Класът изцяло имплементира IRestaurantServices интерфейс.

Следващият клас AuthServices отговаря за удостоверяването и създаването на нови потребители. Първият метод Register е за създаване на потребител. Следващият Login проверява дали потребителя съществува и ако създава ключ (token), чрез друг поверителен метод CreateToken(), който също се намира в класа. Има още два метода, които са за въвеждане на данни в базата данни при самото стартиране на програмата – AdminSeed() и SeedRoles(). Този клас имплементира IAuthServices.

Един от последните класове е DishService, който имплементира IDishService и всички методи отговарят на основните CRUD операции, без никакви други техни разновидности.

За следващият клас можем да каже, че имплементират IBookingServices. Този клас е BookingServices, който в себе си съдържа два метода за извличане, като те са съответно спрямо идентификатора на потребителя или този на заведението.

Последният клас е EmployeeServices съдържа два метода, които служат за създаването на мениджър, служител и доставчик. Тук трябва да се спомене, че те не създават нов потребител, намират вече съществуващия спрямо имейла му, премахват текущата му роля на системен потребител и му дават съответната роля.

## Модел на съдържанието / данните

В тази подточка ще разгледам различните модели, техните връзки и какъв е самият тип на данните.

Във всяка таблица има минимум по един идентификатор, който винаги е от тип Guid, защото предоставя повече сигурност, от колкото идентификаторите от числов тип. Числовият тип е много по-лесен за запомняне, което означава, че ако някой види идентификатора за него ще е много възможно да злоупотреби и да унищожи цялата система.

Друго нещо, което се забелязва е типът на статуса на поръчките и категорията на ястията. Те представляват изброявания (enums), които в базата данни се репрезентират като числов тип. Когато използвам цени, валути или др. се използва типът decimal. Единственият пример за това в тази база данни е цената на поръчка, която се намира в таблицата за заведенията.

Повечето други полета, като имена, наименования, адресът, пароли и др. са от масиви от символи. В базата данни това е възможно чрез използването на типът nvarchar, а в системата това се репрезентира от друг познат тип string.

Адресът е малко по-специфичен в този случай. Както може да се забележи от ER диаграмата има четири полета, които започват с Address\_. В системата има един клас Address, който се използва като обект, който заема ролята на стойност (ObjectValue).

Следващото, което ще се разгледа в тази точка е връзките между различните таблици. Първо да уточним какви различни връзки има в базата данни:

* Едно към едно – връзка между две таблици, в които записът от едната таблица е свързан с точно един от другата
* Едно към много и много към едно – когато записът от едната таблица се свързва с много от другата таблица
* Много към много – записът от първата таблица се свързва с много от втората таблица и обратното. В стандартно за осъществяването на такъв тип връзка се използва помощна свързваща таблица(bridge table).

Пример за едно към много са: един ресторант има много служители, един ресторант има много резервации, ястия и поръчки, един потребител има много резервации и поръчки.

Пример за много към много е поръчките и ястията. Една поръчка съдържа много ястия и едно ястие може да бъде в много поръчки.

# Дизайн

## Реализация на архитектурата на приложението

За реализирането на цялостното приложение съм използвал трислойна архитектура. Всеки един слой ще бъде разгледам по-подробно в следващата подточка, но сега ще представя основно неща свързани с архитектурата.

За реализирането на този проблем се използва .NET, с помощта на който са разработени необходимите функционалности на приложението и също така създаването на базата данни. За тази част се грижи един от най-популярните в наши дни ORM - Entity Framework. С негова помощ се създадат моделите, чрез които можем да запазваме информация в базата, но и също така можем да работим с нашите услуги (services). Реализирането на графичния интерфейс е с помощта една от най-често използваните JavaScript библиотеки ReactJS, с която може да се предостави на потребителите приятен и лесен за използване графичен потребителски интерфейс. Комуникацията между графичния интерфейс и услугите се осъществява чрез HTTP заявки, които са възможни чрез ASP.NET WEB API.

В предишните точки на няколко места споменавах някои от класовете или интерфейсите използвани в приложението, като в тази част на документацията ще поясня в кой слой попадат те и каква е тяхната цел в контекста на приложението.

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

Архитектурата, с която е разработено приложението е многослойна или по-конкретно трислойна. Всеки слой е разделен спрямо това каква функционалност се извършва в него. Погледнато от друг ъгъл приложението може да се раздели на две смислови част. Първата представлява нашия сървър или по-точно нашия програмен код (back-end). В него са включени два от слоевете – слоя за данни и този, които развива нашата бизнес логика. Другата част е клиентът, като в него попада последния слой - презентационния.

Първият слой е така наречения слой за достъпване на данните. Предназначението на този слой е да позволява както създаването на базата данни, така и достъпване на тази информация, изтриването, промяната, създаването и всичко друго, което се отнася до базата данни.

В този слой са включени класовете, които използвам при създаването на база данни. Как това е възможно? Няколко пъти до сега в документацията съм споменавал ORM, но какво точно е това. ORM (Object-relational mapping) представлява начин или техника за създаване на мост между базата данни и програмния код. Има два начина за осъществяване на това:

* **Database First** – първо се създават таблиците необходими за системата и чрез написването на няколко команди в конзолата тези таблици репрезентират по правилни начин в програмния код.
* **Code First** – таблиците или още по-добре познати като модели първо се създават в програмния код и пак чрез написване на няколко команди те се създават в базата данни, като детайли като връзките между различните таблици, колко да е големината на текстовото съдържание и други се определя чрез използването на атрибути или създаването на специални класове за конфигуриране

В проекта е използван подходът **Code First**. Microsoft предоставя един пакет, в който са включени различни класове, които помагат при работата както с потребители, така и с различните роли. Тяхната основана задача е да улеснят работата на програмиста при упълномощаването и удостоверяването на потребителите. Става дума за Microsoft.AspNetCore.Identity.

public class ApplicationUser : IdentityUser<Guid>

{

public string? FirstName { get; set; }

public string? LastName { get; set; }

public Address? Address { get; set; } = default!;

public ICollection<Booking>? Bookings { get; set; } = default!;

public ICollection<Order>? Orders { get; set; } = default!;

public Guid? RestaurantId { get; set; }

public Restaurant Restaurant { get; set; }

}

В този фрагмент от код може да се види как класа *ApplicationUser* наследява класа *IdentityUser<>*, като Guid пояснява какъв ще бъде типът на наследения идентификатор. Друго нещо, което прави впечатление в следния код е това, че имаме свойствата RestaurantId от тип Guid и Restaurant от тип Restaurant, който е един от другите класове в този слой, за който е споменато малко по-напред в документацията. Но защо са ни? RestaurantId е необходимо при създаването на базите данни и репрезентира външен ключ за таблицата Restaurants. Свойството Restaurant е при случаите, в които ще е необходимо да направим тип операция, която наподобява свързване на две таблици (JOIN), като ако имаме връзка едно към много е необходимо да се използва колекция.

Как също може да се види никъде в кода няма атрибути, които да поясняват размера на свойствата от символен тип или каквито и да е други специфични изисквания или ограничения на свойствата. Тогава къде става това? Пример за това е следният клас:

public class UserEntityConfiguration : IEntityTypeConfiguration<ApplicationUser>

{

public void Configure(EntityTypeBuilder<ApplicationUser> builder)

{

builder

.Property(u => u.FirstName)

.HasMaxLength(150)

.IsRequired();

builder

.Property(u => u.LastName)

.HasMaxLength(150)

.IsRequired();

builder

.OwnsOne(u => u.Address);

builder

.HasOne(u => u.Restaurant)

.WithMany(r => r.Employees)

.HasForeignKey(u => u.RestaurantId)

.OnDelete(DeleteBehavior.Cascade);

}

}

Както може да се види този клас имплементира *IEntityTypeConfiguration<>* интерфейса, който има един метод Configure, който служи за конфигуриране на класа. Тази конфигурация се прилага при създаването на базите данни. Нека разгледаме следната конфигурация. Първо се конфигурира свойството за име (FirstName), като му се задава максималния брой на символите, които трябва да се съдържат в него, а след това, че то трябва да съдържа някаква стойност, не може да бъде NULL. На следващото свойство се прилага същата конфигурация. Третото е малко по специално. То представлява обект, който заема ролята на стойност (ObjectValue). Класът, който се използва в този случай е Address. Той съдържа няколко полета – страна, град, улица и номер. При създаването на базите данни свойствата в този клас са репрезентирани като полета на таблицата AspNetUsers в следния формат – Address\_Country, Address\_City, Address\_Street и Address\_Number. Последното представлява създаването на връзката много към едно. Много потребители или по-точно в контекста на приложението могат да работят в един ресторант. След това казване, кое свойство в модела представлява нашия външен ключ в базата данни. И накрая казване какво се случва с другите полета, които са свързани с това, при неговото изтриване. В случая те също ще бъдат изтрити.

Картина, която съдържа текст, Шрифт, екранна снимка

Описанието е генерирано автоматично Картина, която съдържа текст, Шрифт, бял, екранна снимка

Описанието е генерирано автоматично

Горните две снимки представят останалите модели и класове, които служат за тяхното конфигуриране. Моделът OrdersDishes представлява модел, които при създаването на базите данни отговаря за свързващата таблица, за таблиците за поръчки (Orders) и за ястията (Dishes).

В този слой има още няколко класа. Един от тях е *ApplicationDbContext*.cs, който има няколко DbSet<>, които репрезентират различните таблици, които се създадат в базата данни. AuditableEntity и BaseEntity са абстрактни класове, които съдържат общите атрибути за двата вида единици. AuditableEntity наследява BaseEntity, който съдържа едно поле Id, и имплементира IAuditableEntity, което има четири свойства, които са за това кой и кога е създал единицата (entity) и кой и кога последно е правил промени по него. Остава само един клас DependencyInjection, който съдържа един разширен метод (extension method), който служи за конфигуриране на базата данни.

Следващият слой от трислойната архитектура представлява цялата бизнес логика на приложението. Ще разгледам отделните папки една по една както следват:

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

В проекта е наблегнато на използването на SOLID принципите, както е случая с първата папка. В нея се съхраняват различните интерфейси, които са имплементирани в следващите класове. Следващата папка (Identity) съдържа един клас, благодарение на който е възможно да вземем идентификаторът (Id) на текущия потребител, което помага в повечето методи в следващите класове за проверка дали дадената резервация или поръчка му принадлежи.

Първата папка, която разгледах съдържа само и единствено интерфейсите, в третата се съдържат и вече техните имплементации. Ще разгледам един от класовете и неговите методи, за да придобием някаква представа.

Картина, която съдържа текст, екранна снимка, Шрифт, номер

Описанието е генерирано автоматично

Както се вижда класът BookingSerices, който имплементира интерфейса IBookingSerices, съдържа шест метода, които отговарят на CRUD – един метод, който създава самата резервация, един за изтриване, няколко за извличане на записи от базата данни и един за промяна на информацията за резервацията

Има още два класа, JwtConfiguration и DependencyInjection. И двата класа съдържат по един разширен метод (extension method). Методът в JwtConfiguration, който служи за конфигурирането на удостоверяването чрез използването на Jwt(JSON web token), в DependencyInjection – регистрирането на различните услуги (services) чрез AddScoped<>().

В текущата част има още три проекта, като един от тях представлява тестове, които ще разгледам в друга точка от тази документация, така че ще се съсредоточа върху останалите два. За тези два проекта може да се каже, че са помощни или такива които осигуряват комуникацията между различните слоеве. Първият от тях е Share. В него са разпределени различни класове и интерфейси, които се използват във всички останали проекти. Пример за това са моделите, които служат за въвеждаме на нова информация и тези, които се използват във връщания от контролера резултат, но за това малко по-напред.

Картина, която съдържа текст, Шрифт, екранна снимка

Описанието е генерирано автоматично

В първата папка се намират всички интерфейси, които се използват не само в бизнес логиката, но и в контролерите. В случая тя съдържа само един единствен, който се имплементира от функционалността, която служи за взимането на идентификатора на текущия потребител, ако се е удостоверил, в противен случай връща NULL.

Друга функционалност, която се използва в няколко слоя или не е точно за определен слой са статичния клас Roles и изброяванията (enums). Цялостната роля на статичния клас е да съдържа имената на различните роли под формата на статично зададени свойства. Това се използва при създаването на самите роли и проверката дали даден потребител има дадената роля.

public static class Roles

{

public const string User = "User";

public const string Admin = "Admin";

public const string Employee = "Employee";

public const string Manager = "Manager";

public const string Deliver = "Deliver";

}

Изброяванията или още така известни като enums представляват дефинирани от потребителя смислово свързани константи, които помагат при работата с кода. Приложението съдържа две изброявания OrderStatus и FoodCategory, които отговарят съответно за текущото състояние на поръчката и за категориите на храните. Сега ще разгледам какво представлява едно изброяване:

public enum OrderStatus

{

Canceled = 0,

NotFinished = 1,

Finished = 2,

WaitCollection = 3,

Accepted = 4,

Delivered = 5,

}

Както е показано всяка от константите е смислово свързана с цялостната идея на изброяването. Всяка една от тях има стойност, с която се различава от другите и не е задължително да са с последователни числа, но за да придобие цялостен ред е хубаво да се използват. Текущото изброяване помага при извличането на поръчките, за конкретен потребител, служител на някое заведение или доставчик.

Друго, което съм разработил в този проект са различни модели. Тези модели представляват информацията, която се очаква да бъде въведена от потребителя или да бъде визуализирана, като те до голяма степен наподобяват моделите, които се използват за базата данни, но при тях свойствата, които съдържат важна информация или не могат да се виждат от всеки, липсват.

Следващият помощен проект служи за свързване и обмяна на информация между слоя за бизнес логиката и презентационния. Как става това? Типът на този проект е ASP.NET WEB API. За осъществяването на тази комуникация се използват HTTP заявки.

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

Както може да се види проектът има няколко различни папки и файлове. Основната папка и тази върху ще да се съсредоточим е Controllers. В нея се съдържат няколко различни проекта. Всеки един от тях съдържа методи, в които се необходимите методи от съответните услуги в бизнес логиката. Какво се има на предвид? Ако отворим папката с контролери се забелязва това:

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

Картина, която съдържа текст, екранна снимка, Шрифт

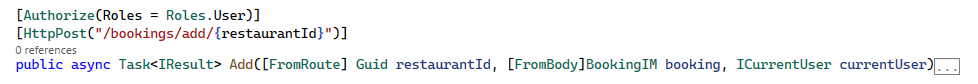
Описанието е генерирано автоматичноВсеки един от тях е наименуван спрямо различните услуги (services), за които отговаря. Ще разгледаме един от тях, структурата в самия клас и неговите методи:

За да се поясни, че този контролер се използва като такъв се слага атрибутът ApiController. Следващият атрибут се използва, за да зададе основна информация за маршрута, който ще се използва. Вече навлизаме в самия контролер. Както може да се забележи контролерът наследява класа ControllerBase, които служи за използване на MVC без възможността за изглед (View). Да погледнем първия метод. Той има два атрибута, първият показва, че този метод от контролера е достъпен само за потребители, които са удостоверени и имат роля User. В случай, че не са удостоверени методът връща статус код *Unauthorized*, а ако не е User – *Forbidden*. Но от къде се разбира това? Благодарение на Jwt(Json web token), които съдържа в себе си, идентификатора и ролята на потребителят и датата, до която е валиден и др.

Когато работим с HTTP заявки е важно да знаем за какво е метода. Това се разбира чрез следните ключови думи – GET, POST, PUT, DELETE и др.

* GET – извличане на необходимата информация. Методите от този тип не могат да имат тяло (body). Единственият начин да се даде някаква допълнителна информация, за да извлечем информацията спрямо някакви изисквания става чрез заявки добавени към маршрута (query string).
* POST – създаване на нов обект. При него и при останалите е възможно и тялото, и заявки добавени към маршрута.
* PUT – промяна на информацията в обекта или създава нов.
* DELETE – изтрива обектът.

За да поясня за какво е метода, използвам атрибутите – HttpGet, HttpPost, HttpPut, HttpDelete и др. Има и опцията да поясним маршрута, като използвам атрибутът Route() и в скобите напишем маршрута, който когато напишем ще изпълни метода, за който се отнася. Има друго място където можем да го зададем както е показано на снимката, направо в HttpGet, HttpPost, HttpPut или HttpDelete. Ако искаме да вземем информация от тялото на заявката или от форма е като даден на някой параметър един от следните атрибути – FromBody или FromForm.

Ако трябва да извлечем идентификатор или нещо друго включено в маршрута задаваме на някой от параметрите FromRoute, както е показано по-долу.

Във файла appsettings.json се съдържа информация, която се отнася за конфигурирането на приложението, като свързващи символни низове (Connection string), глобални променливи и др.

И последно, разглеждам Program.cs, то представлява началната точка, цялата сървърна част. В нея се създава уеб хоста, който служи за настройване на необходимите услуги (services) и междинни софтуери (middlewares). В Program.cs също така и се извиква метода Run, който стартира уеб сървъра и започва да чака идващите HTTP заявки.

Следващата голяма част представлява така наречения презентационен слой. Той е разделен на няколко част – компоненти, услуги, модели и стил. Папката съдържа различните компоненти, които се използват. Като тя също е разделена на няколко под папки. Всяка една от тях отговаря за компонентите на различните роли, като има някои, които използват компоненти от други под папка или от външни библиотеки за компоненти на ReactJS.

В папката услуги (services) се намират класове, в които се викат HTTP заявките, за които е написано по-нагоре и както при контролерите, така и тук всеки един от тях е наименуван спрямо различните услуги, за които отговаря. Следният код представлява как презентационни слоя изпраща информация на бизнес логиката:

async AddBooking(booking: BookingIM, restaurantId: string): Promise<string>

{

const requestOptions =

{

method: 'POST',

headers:

{

'Content-Type': 'application/json',

'Authorization': 'bearer ' + localStorage.getItem('token')

},

body: JSON.stringify(booking)

}

const response = await fetch(base + `/ bookings / add /${restaurantId}`, requestOptions).then((response) => response.json()).then((data) => {return data});

return response;

}

Кодът представлява създаване на нова резервация. Функцията приема информацията за поръчката, превръща я в JSON формат и ги слага в тялото на заявката. Заявката още включва какъв тип е тя, допълнителна информация за самата заявка (header) – тип на данните в заявката и удостоверяването. След това използва функцията fetch, с която се осъществява връзка с контролерите от по-рано. Отговорът (response), които се връща се преобразува в JSON формат и след това информацията се връща на потребителят в правилния тип данни.

Моделите са идентични с тези от проекта Share и както там, така и тук се използват при въвеждането на нова информация и предаването й на съответната заявка и визуализирането на получената от HTTP GET заявките.

И в последната част представлява стилизацията на всички компоненти, като в нея може да се намерят всички css файлове.

## Организация и код на заявките към база от данни

В тази точка от документацията разглеждам как се съхраняват и извличат данните от базата. Преди да започна с тази част ще разгледам какво представлява Repository pattern. Repository pattern е design pattern, при който се създава клас, които отговаря пряко за връзката между програмния код и базата данни. Това означава, чрез директни заявки в кода се добавят, променят, взимат или изтриват записи в базата данни. Когато има много такива хранилища (repositories) се образува друг design pattern наречен Unit of work. Какво общо има това? В проекта се използва Entity framework, който ни предоставя DbSet, което представлява нашето хранилище (repository), а DbContext – Unit of work. Това, което правят е всичко свързано с управлението на базите данни без ние да знам как се случва това. DbSet има различни методи, които са свързани с добавянето, изтриването, промяната, извличането и др., които помагат за работата с базата.

Преди да разгледам конкретен метод е важно да се спомене едно нещо – в повечето услуги (services) има по едно свойство context от тип ApplicationDbContext, което се намира с слоя за достъп на данните. Този клас наследява IdentityDbContext<>, който е подобен на DbContext<> с малката разлика, че IdentityDbContext<> предоставя допълнителни таблици свързани с ролите и др., които помагат при упълномощаването и удостоверяването. ApplicationDbContext съдържа няколко DbSet<>.

public class ApplicationDbContext : IdentityDbContext<ApplicationUser, IdentityRole<Guid>, Guid>

{

private readonly EntityState[] auditableStates =

{

EntityState.Added,

EntityState.Modified,

};

private readonly ICurrentUser currentUser;

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options, ICurrentUser currentUser = default!)

: base(options)

{

this.currentUser = currentUser;

}

public DbSet<Booking> Bookings { get; set; }

public DbSet<Dish> Dishes { get; set; }

public DbSet<Order> Orders { get; set; }

public DbSet<Restaurant> Restaurants { get; set; }

public DbSet<OrdersDishes> OrdersDishes { get; set; }

............

}

За да придобием по-ясна представа ще разгледам следния метод, който предоставя ясен пример за работата на DbContext<> и DbSet<>:

public async Task<Guid> Add(BookingIM booking, Guid restaurantId, Guid? userId)

{

var user = await this.userManager.FindByIdAsync(userId.ToString());

if (user == null)

{

throw new ArgumentException("There is an empty dish");

}

var restaurant = await this.context.Restaurants.FindAsync(userId.ToString());

if (restaurant == null)

{

throw new ArgumentNullException("Restaurant doesn't exist or something get wrong")

}

if (booking == null)

{

throw new ArgumentNullException("Booking doesn't exist or something get wrong");

}

var newBooking = booking.Adapt<Booking>();

newBooking.UserId = userId;

newBooking.RestaurantId = restaurantId;

await this.context.Bookings.AddAsync(newBooking);

await this.context.SaveChangesAsync();

return newBooking.Id;

}

В първата част на метода проверявам дали потребителя и ресторанта съществуват и след това дали модела за резервации не е празен. След това информацията се разпределя (mapping) – информацията, в свойствата с идентични имена в класовете BookingIM и Booking се записват от обекта на първия клас в новия обект на другия. След като се разпределят, новият бива добавен в DbSet<Booking> в context. След това всички промени направени в context се запазват или с други думи Entityframework извършва промени в базата данни на базата на тези направени в context, извършва на заден план необходимите заявки.

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

За реализирането на уеб потребителския интерфейс е използвана една от най-популярните в наши дни библиотека на JavaScript – ReactJS. С помощта на библиотеката и различни нейни разширения е възможно създаването на лесен за използване и с приятен изглед потребителски интерфейс. Използваните разширения може да се видят в края на документация в точна номер осем – Използвани литературни източници и Уеб сайтове. В тази подточка също ще се видят и различните основни функционалност, които предоставя системата за своите потребители

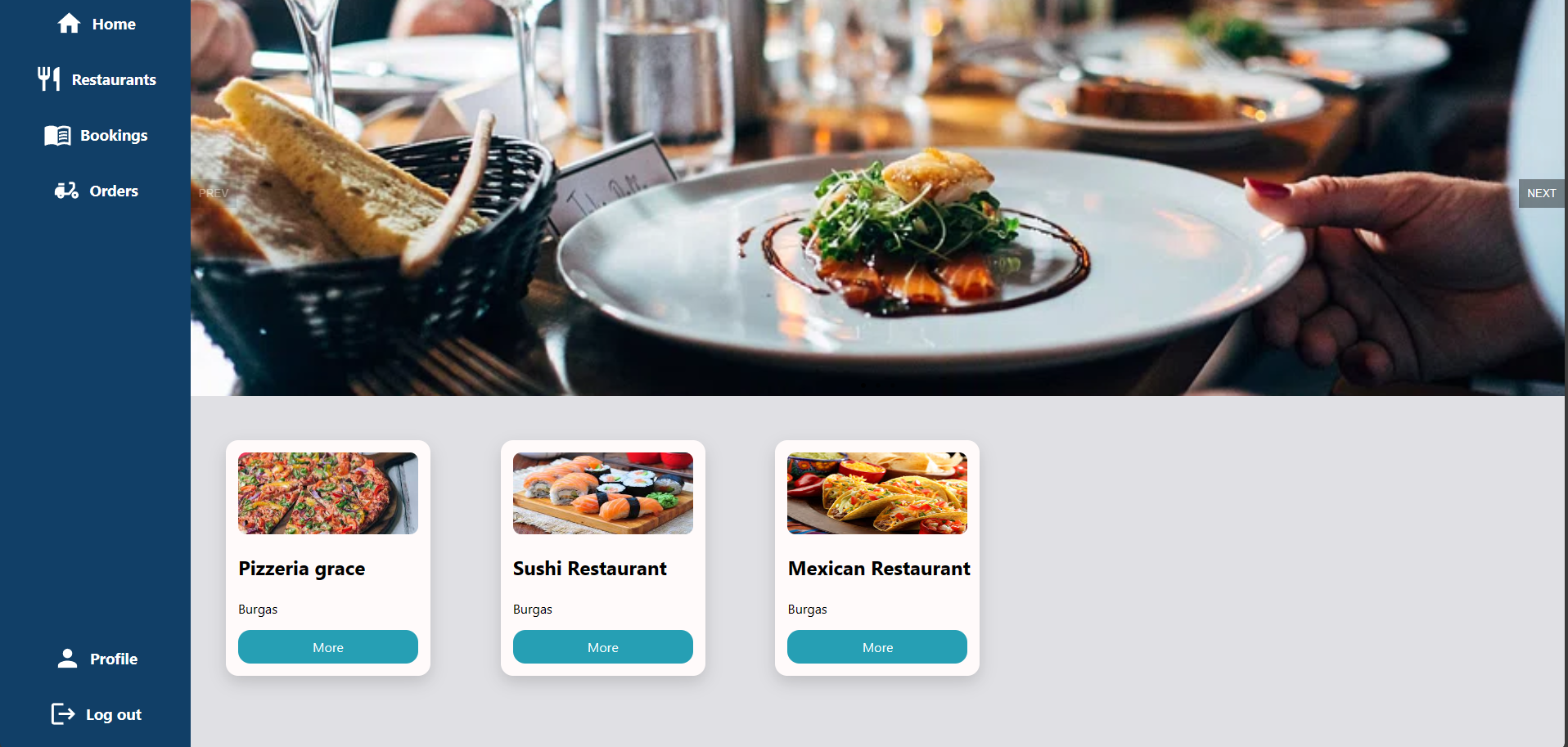
Както вече споменах има няколко различни роли – системен потребител, администратор, служител, мениджър и доставчик. Всяка от тях има различен изглед на уеб потребителски интерфейс. По надолу в тази подточка ще бъдат разгледани по отделно, като интерфейсът, които се отнася към служителите и мениджърите се препокрива в повечето, така че ще бъде разгледа като едно цяло, като разликите ще бъдат пояснени.

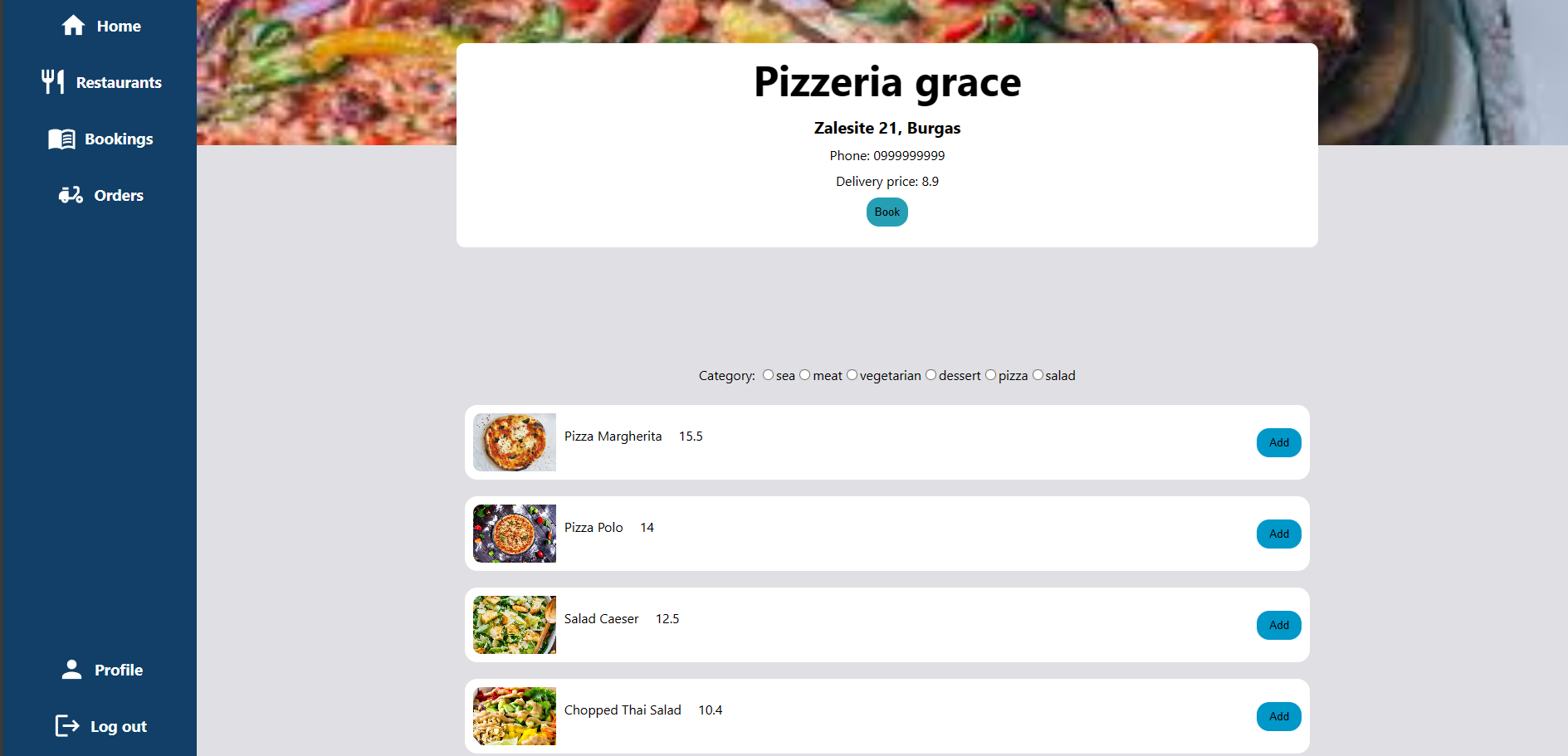
След като ASP.NET WEB API проектът бъде стартиран всички функционалности, които използва потребителския интерфейс могат да бъдат достъпни и няма да има никакви проблеми по време на работата с него.

A screenshot of a computer

Description automatically generatedСлед като се отвори началната страница се забелязват опциите за регистриране и за влизане в системата. Ако потребителят реши да се регистрира това, което се визуализира е една кратка форма, в която единствено трябва да въведе име, фамилия, имейл и парола. След като се направи регистрация, се връщате пак към началната страница. Ако вече има направен профил и решите да се впишете в системата трябва само да се въведе имейла и паролата. Това са две от общите неща, за всички потребители. Другото, което е общо е възможността да променят информация свързана с тях, като име, фамилия и адрес. Също така и възможността да се смени паролата. Ако потребителят вече не желае да е част от система е необходимо само да въведе текущия си имейл и профилът ще бъде премахнат.

Когато системният потребител влезе в системата му се показва първо сменящи се рекламни постове, а под тях няколко от ресторантите в града. С натискаме на бутона „More“ се отива до друга страница, в която се вижда повече информация за заведението, какви ястия предлагат и опцията за резервиране. При натискането бутонът „Book“ се показва формата, с която може да си резервирате маса. Единствено трябва да се въведат дата и часа, броя на хората и също, така да се избере дали предпочитате да сте вътре или извън заведението. Друго което може да се прави в тази страница е при избиране на бутонът „Add“ изкача нова форма, на която може да се види информация за самото ястие и едно поле, в което се въвежда броя, който желае клиентът.





Друго, което може да прави клиентът е да преглежда своите резервации и чрез натискане на хикс да бъде премахната.

Потребителят също може следи своите поръчки като може да ги филтрира както по ресторант или заведение, от което са направени, така и от дата до дата на поръчки, които са вече завършени.

След натискане на бутона „Повече“ потребителят бива отведен в нова страница, от която може да променя, боя на ястията в поръчки, които не са финализирани, да премахва изцяло дадено ястие и финализира поръчката си. След като го направи единственото, което остава е да чака някой да му я одобри и да бъде доставена.

A screenshot of a computer

Description automatically generated

И последното, което може да прави потребителят е да променя основната информация за себе си, паролата си и да изтрие напълно своят акаунт.

Следващата част от графичният интерфейс се отнася до служителите и мениджърите. Когато потребител с такава роля влезе в системата, първата страница, към която ще бъде отведен, ще е тази за поръчките. От там те ще могат да видят какви ястия има в поръчката или да я приемат, с което ще дадат възможност на доставчиците да я видят и да я вземат. Това става благодарение на натискане на бутоните „Изчаква за събиране“ от страна на служителя и “Приемане” от страна на доставчикът.

A screenshot of a computer

Description automatically generated

Друга опция, която може да изпълнява един служител е да добавя, променя вече създадено или да премахва ястие от предложенията на дадения ресторант. Тези действия стават благодарение на бутонът “Добави” най-отгоре на страницата и съответно бутоните “Промени” и “Премахни” от дясната част на ястието.

A screenshot of a computer

Description automatically generated

Последната функционалност, която е възможна за служителите е прегледът на резервациите.

A screenshot of a menu

Description automatically generatedПоследната роля, която остана е тази на администраторът. Функционалностите, които са му предоставен се отнасят до създаването и управляването на заведенията. Към това се отнасят промяната на информацията на дадено заведение, изтриването му или назначаването на мениджъри.

# Ефективност и бързодействие на решението

Ако трябва да се сравнява с останалите подобни решения и предоставеното в тази документация, това не е най-оптималното от гледната точка на бързодействие или с потребителско преживяване, но предоставя задоволително и не толко натрапчиво чувство. От гледна точка на ефективност решението се справя значително добре, с основната работа, за която е предназначено.

В какво се корени всичко това? От гледна точка на бързодействие на приложението има какво още да се желае. Един от основните проблеми за това е зареждането на снимките. Те се съхраняват като масив от байтове и за да се визуализират трябва да бъдат превърнати в base64 формат, което при наличието на множество снимки може да усложни цялостния проект.

По отношение на ефективността приложението изпълнява поставените изисквания и основните си задължения, но по отношение на потребителското преживяване има какво още да се постигне и да се добави.

Последно нещо, което трябва да се спомене за ефективността се отнася за времето за изпълнение на методите. Всеки един от тях се изпълнява за различно време. В повечето методи се използват FindByIdAsync, Find и Where. FindByIdAsync се използване е предоставен от userManager. Този метод си служи с различни структури от данни като hash таблици и др. и заради това в повечето случай нотацията му е O(1). Не винаги е така.Това зависи от броят на потребителите, но винаги е константна стойност. Find и Where са предоставени от LINQ. В повечето случай и двете имат нотации приближаващи се към O(log n). В зависимост колко е комплексна заявката Where може да достигне до O(n).

# Тестване

В тази част разглеждам различните тестове направени за системата. Тестовете са интеграционен (integration), защото повечето методи са свързани по някакъв начин с използване на базата данни. Представените тестове са свързани с услугите на ресторантите и резервациите.

Тестовете за ресторантите включват три теста за съответните методи: Add(), Delete(), GetRestaurants().

* AddRestaurant\_ShouldReturnRestaurantId() – този тест отговаря за тестването на метода Add(), като създава нов обект от тип RestaurantIM, задават му се основните неща за ресторанта, след това се подава на метода. След като сме взели идентификатора чрез друг метод GetRestaurant взимам ресторанта и проверявам дали имената съвпадат.
* GetRestaurants\_ShouldReturnRestaurantsInCity – задачата на този тест е да провери дали методът GetRestaurants() взима всички заведения в града на потребителя. Първо се създават две примерни заведения. След това извикваме метода и очакваме, броя на заведенията в града да са толкова колкото са направили.
* Delete\_ShouldReturnTrue – тестът се използва, за да провери дали ресторантите се изтриват правилно чрез метода Delete(). Създавам един нов ресторант и използвам върнатия идентификатор. Използвам метода. Проверявам дали върнатият резултат е true, ако е правил още една проверка, дали ако се опитаме да вземем даденото заведение дали ще върне специфична грешка.

Следващите тестове са за следните методи на услугите на резервациите – Add(), Delete(), GetAllByUserId().

* AddBooking\_ShouldReturnNewBookingId – създава се един нов ресторант, които да може да си направил резервация. След това се прави и самата резервация. Използвам метода Add(), който връща идентификатора. След като го използвам за да вземем тази резервация, проверявам дали връща резервация или null.
* Delete\_ShouldReturnTrue – тестът проверява дали метода Delete() изтрива резервацията. Първо се създава един нов ресторант и резервация към него. Върнатия идентификатор се подава на метода за изтриване. Проверяваме дали върнатият резултат е true, ако е правил още една проверка, дали ако се опитаме да вземем даденото заведение дали ще върне специфична грешка.
* GetAllByUserId\_ShouldReturnAllUserBookings – тестът се използва за проверка на метода GetAllByUserId(), който извлича от базата данни всички резервации на дадения потребител. Първо се създава един нов ресторант и три резервации към него. След като това бъде направено се извиква метода и се проверява дали броя им съответства на очаквания. В зависимост дали се стартират всички тестове или само този, броят на извлечените резервации е различен.

# Заключение и възможно бъдещо развитие

Какво заключение може да се извади от показаното до сега? Проблемът, които решава това приложение започва все повече да се засилва с нарастването на желанието у хората да се развиват и нежеланите и отнемащи значително голяма част от тяхното време задачи и задължения да бъдат оставяни на заден план. И такива приложения като представеното в документацията има и все повече ще се появяват. Представеното решение е до голяма степен задоволително като за начален етап, но има още много което може да се добави или да се променя за напред в бъдещето.

Пример за нещо, което може да се промени за в бъдеще е потребителския интерфейс. Друго нещо, което може да се добави са нови функционалности свързани със самите резервации и поръчките, по-лесен начин за съхраняване на снимките и други данни, които могат да забавят зареждането на различните страници, разширяване на избора на ресторантите.

# Използвани литературни източници и Уеб сайтове

1. Уеб сайт на appmysite, адрес <https://www.appmysite.com/blog/online-food-ordering-statistics/>

*Material UI*. (н.д.). Извлечено от Material UI: https://mui.com/

*Microsoft .NET Documentation*. (н.д.). Извлечено от Microsoft .NET Documentation: https://dotnet.microsoft.com/en-us/

*Microsoft .Net EntityFramework*. (н.д.). Извлечено от Microsoft .Net EntityFramework: https://learn.microsoft.com/en-us/ef/

*Microsoft Asp.Net Documentation*. (н.д.). Извлечено от Microsoft Asp.Net Documentation: https://dotnet.microsoft.com/en-us/apps/aspnet

*React*. (н.д.). Извлечено от React: https://react.dev/