**ДИПЛОМЕН ПРОЕКТ**

**ЗА ДЪРЖАВЕН ЗРЕЛОСТЕН ИЗПИТ**

по професия код 481030 „Приложен програмист“

специалност код 4810301 Приложно програмиране“

ТЕМА: „ЕЛЕКТРОННА СИСТЕМА ЗА УПРАВЛЕНИЕ НА ВЕРИГА ОТ АПТЕКИ“

Автор:

Ивайло Младенов Абаджиев, клас XII В

Ръководител:

Данаила Стаматова

БургасСЪДЪРЖАНИЕ

[1 Увод 2](#_Toc163302675)

[2 Цели и обхват на софтуерното приложение 3](#_Toc163302676)

[3 Анализ на решението 6](#_Toc163302677)

[3.1 Потребителски изисквания и работен процес 6](#_Toc163302678)

[3.2 Примерен потребителски интерфейс 8](#_Toc163302679)

[3.3 Диаграми на анализа 14](#_Toc163302680)

[3.4 Модел на съдържанието / данните 24](#_Toc163302681)

[4 Дизайн 27](#_Toc163302682)

[4.1 Реализация на архитектурата на приложението 27](#_Toc163302683)

[4.2 Описание на слоевете, предназначението им, библиотеки и методи включени в съответния слой. 29](#_Toc163302684)

[4.3 Организация и код на заявките към база от данни 35](#_Toc163302685)

[4.4 Наличие на потребителски интерфейс (конзолен, графичен, уеб) 41](#_Toc163302686)

[5 Ефективност и бързодействие на решението 43](#_Toc163302687)

[6 Тестване 44](#_Toc163302688)

[7 Заключение и възможно бъдещо развитие 48](#_Toc163302689)

[8 Използвани литературни източници и Уеб сайтове 49](#_Toc163302690)

[9 Приложения 49](#_Toc163302691)

# Увод

В днешно време, с увеличаващия се ръст на заболеваемостта сред хората и наличието на множество различни вируси и болести, аптеките са изключително посещавани места. Особено след Covid-19, вирусите станаха по-често срещани и, за съжаление, хората боледуват по-често, което подчертава значението на аптеките за обществото. Това демонстрира колко важно е те да функционират ефективно и ефикасно. За да постигнат това, аптеките се нуждаят от качествен софтуер, който да подпомага управлението и работата на персонала им.

Настоящият проект представлява система за управление на верига от аптеки, както и на складовете, свързани с тях. Всеки човек, който си е създал профил в системата има право да добавя както аптеки, така и складове. Управителят на аптеката може да избира с кой склад от наличните в системата да работи, като подава заявка за работа с него. В самия проект има много улеснения за целия персонал на аптеката, за собственика на аптеката, както и за управителя на склада.

Системата използва “ASP.NET Web API” за „back-end“, както и “React” (javascript базирана библиотека) за „front-end“. Проектът работи на основата на „HTTP“ заявки, тоест клиентът ги изпраща към сървъра и сървъра връща отговор. Използването на такъв “framework” като този за създаването на клиента позволява направата на един изчистен, функционален и гъвкав дизайн, който да се хареса на потребителите.

Чрез тази система, аптеките ще могат да се посветят изцяло на своята работа, без да се налага да използват стар и нефункционален софтуер, защото тази система предоставя както много полезни функционалности, така и интуитивен и стилен графичен интерфейс. Възможността за управлението на аптеките и складовете към тях в една система позволява по-лесна и по-ефективна работа на управителите и служителите, както и по-бързата комуникация между склада и аптеките, които работят с него.

Благодарение на средата за моделиране на диаграми (LucidChart) ще е възможно да се представят някои от основните части на проекта под формата на различни диаграми. Например това какви класове се използват в системата, какви таблици има в базата данни, какви връзки има между тях и какви атрибути и типове имат всяка една от тях. Може да се представи чрез диаграми какви дейности се извършват в системата, ролите в системата и какви права дава всяка роля.

# Цели и обхват на софтуерното приложение

Целта на проекта е да се създаде електронна система за управление на аптеки и складове за лекарства, която да подпомага дейността им.

Системата значително улеснява работата на управителите на складовете, на управителите на аптеките и на фармацевтите заради интуитивния и лесен за използване графичен интерфейс и много функционалности, с които системата превъзхожда конкуренцията.

Целевата група са всички собственици на аптеки, както и тези на складове, които планират да работят с аптеки, защото системата е разработена така, че комуникацията и работата между складовете и аптеките да бъде възможно най-улеснена и бърза.

Когато се влезе в системата, на потребителя се добавя роля „базов потребител“, която му предоставя опциите да преглежда профила си, където може да променя данните си като имейл, адрес, име и фамилия, както и своята парола. Също има правото да добавя и премахва складове, както и аптеки. Следователно, когато създаде склад, на него му се добавя роля „управител на склад“, а когато създаде аптека – „собственик на аптека“. Тези две роли дават достъп до много полезни функционалности на системата, които улесняват работата.

Ролята „управител на склад“ дава достъп до функционалности, свързани с управляването на склада, като добавяне на лекарства, премахването им. Има опция за промяна на информацията на дадено лекарство, като може да се променят цената му, количеството му, както и може да се добавя снимка, която се съхранява с помощта на облачните услуги на „Microsoft Azure“и по-специфично „Azure Blob Storage“. Може да се следи кога е добавено лекарството, кога последно е променяна информацията за него, дали е в срок на годност, колко е наличността му. Управителят може да следи с кои аптеки работи, да приема заявки за работа с нови аптеки, подадени към склада, както и да преглежда историята на зареждания и презареждания, които склада е извършвал в аптеките, с които работи. Когато някое от лекарствата в някоя от аптеките с които склада работи се изчерпи, тоест е с нулево количество, аптеката и склада, асоцииран с нея, получават известие, че даденото лекарство е изчерпано и се нуждае от презареждане.

Ролята „собственик на аптека“ дава достъп до функционалности, свързани с управлението на аптеката. Чрез тази роля се получава достъп до две менюта: главна страница, където може да се управляват лекарствата в аптеката и която не може да бъде използвана преди аптеката да е свързана с някой склад, както и страница за настройките, която служи като административен панел. В нея собственикът може да променя името на аптеката, както и адреса и. Има правото да добавя и премахва служители, като добавянето става, като собственикът въведе името и фамилията, имейла и телефонния номер на служителя, както и нова парола. По този начин се създава профил и след това на посочения имейл се изпраща известие на дадения работник, че е назначен и му се праща паролата, с която после може да влезе в системата като служител. Също на служителя му се дава роля „фармацевт“, която му предоставя много функционалности. Изпращането на имейл се осъществява отново чрез облачните услуги на „Microsoft Azure“ и по-специфично „Azure Communication Services“. При премахването на служител (фармацевт), профилът на фармацевта просто се премахва от системата и става невалиден. Последната опция е добавянето на склад. Собственикът на аптеката има право да подаде заявка за работа с един от складовете, налични в системата и в последствие може да смени склада с някой друг. Заявката се изпраща до склада и когато управителят на склада приеме заявката за работа се дава възможност на аптеката да започне да работи. Също тази роля дава достъп до всички функционалности, които са свързани с аптеката, включително и тези, които служителите могат да извършват.

Ролята „фармацевт“ дава достъп на служителите на аптеката към първоначалната страница, където могат да управляват работата на аптеката, тоест всичко свързано с лекарства. Могат да се заявяват лекарства от склада, с който аптеката работи, да се презарежда някое лекарство с определено количество, да се премахват лекарства, както и да се променя снимката на дадено лекарство. Имат достъп до опция, чрез която може да се види снимката на лекарството, да се промени, както и да се прочете за състава и къде се употребява избраното лекарство. Фармацевтите имат право да продават лекарства, като първо се избират кои лекарства да бъдат продадени, след това се избира количество на всяко едно от лекарствата и след това, когато се натисне бутона за продаване, се генерира фактура, която съдържа името на аптеката, отговорното лице (собственикът на аптеката), датата, кои лекарства са продадени, количеството и финалната цена. Фармацевтите имат право да споделят тази фактура по имейл, като трябва да въведат валиден имейл адрес и как искат да бъде името на файла.

Целта е да се създаде система, която превъзхожда конкуренцията със своята леснота на използване, функционалност, бързина и сигурност. Различните нива на достъп (роли) правят системата изключително сигурна. Подцел е привличането на клиенти, които работят в сферата на лекарствата (аптеки и складове), които да предпочетат този софтуер пред останалите. Ключови са функционалностите, както и интерфейса, защото това впечатлява потребителите. Това, което тази система предоставя пред конкуренцията е опцията за работа със складове в системата, многото полезни функции, интеграцията с Microsoft Azure, както и бързината и сигурността на приложението.

Системата може да намери приложение във всяка аптека. Понеже е уеб приложение, което може да бъде достъпвано онлайн от всеки, то може да бъде използвано както за работа, така и за лични интереси. Главното приложение е сред аптеките и складовете, които работят за аптеки и се занимават с лекарства. Популяризирането и може да бъде чрез директен маркетинг, като ползвателите на системата които в последствие да я препоръчат на други свой познати, които са в същата сфера на работа и така системата да започне да се използва от възможно повече аптеки, което да доведе до повече приходи, които биха се използвали нови функционалности. Също така приложението е лесно достъпно и не пречи да я достъпват може да се разглежда и от хора, които да не са в тази сферата, но могат да го популяризират.

# Анализ на решението

## Потребителски изисквания и работен процес

Всеки потребител си взаимодейства със системата посредством уебсайт.

Диаграмата показва работния процес и различните нива на достъп на системата за управление на вериги от аптеки, както и складове, с които в последствие може аптеките да работят.

Потребителите на ниво базов потребител, чиято роля се дава на съответния потребител в момента в който е направен профил в системата, има право да променя информацията за профила си като име, имейл адрес, телефонен номер, адрес, както и паролата. Също може да регистрира склад, както и аптека.

Потребителите на ниво управител на склад може да добавят и премахват лекарства в склада, да променят информацията за тях като цена, брой и снимка. Също могат да приемат заявки, подадени от аптеките за работа със съответния склад, да преглеждат историята на заявки за лекарства от аптеките и да премахват склада от системата.

Потребител на ниво фармацевт имат достъп само до аптеката, към която са добавени. В нея имат право да достъпват само първоначалната страница, където да управляват лекарствата, но не и настройките. Имат право да генерират заявки за зареждане и презареждане на лекарства, да премахват някое лекарство от системата, да го продават, както и да преглеждат допълнителна информация за лекарството като инструкции за използване и състав. Имат право да добавят снимки на лекарството, както и да виждат кога последно е добавено и променяно дадено лекарство. След като се продаде дадено лекарство се генерира фактура, която може да бъде споделена от фармацевта чрез посочване на имейл адрес и име на файла.

Потребителите на ниво управител на аптека имат достъп до две менюта: това, което фармацевтите виждат, което е за управлението на лекарствата в аптеката, както и към настройките, където имат право да променят информацията на аптеката като име и адрес. Могат да добавят и премахват работници, както и да заявяват работа с някой от складовете, които са регистрирани в системата. Също имат право да премахват аптеката от системата. След като работник бива добавен в системата, на имейла посочен от неговия работодател се праща паролата с която може да влезе в системата. При премахване на работник от системата, неговия профил бива изтрит и достъпът му до системата бива пректратен.

## Примерен потребителски интерфейс

A blue background with white text

Description automatically generated

Началната страница на проекта (Фигура 1).

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

Форма за влизане в системата. Форма за регистриране в системата.

(Фигура 2) (Фигура 3)

A screenshot of a computer screen

Description automatically generated

Меню, чрез което избираме дали да ни препрати към каталога с аптеките или със складовете (Фигура 4).

A screenshot of a computer

Description automatically generated

Каталог с всички складове, управлявани от съответния потребител (Фигура 5).

A screenshot of a computer

Description automatically generated

Каталог с всички аптеки, които са собственост на съответния потребител.

(Фигура 6)

A screenshot of a computer

Description automatically generated

Интерфейс на панела за управление на складовете. (Фигура 7)

A screenshot of a computer

Description automatically generated

Интерфейс на панела за управление на лекарства в аптеката.

(Фигура 8)

A screenshot of a computer screen

Description automatically generated

Първата опция от административния панел на аптеката, която дава възможност на собственика да променя информацията, свързана с нея.

(Фигура 9)

A screenshot of a computer

Description automatically generated

Втората опция от административния панел, където собственикът на аптеката може да добавя нови служители (фармацевти) и да ги упълномощава с ролята „фармацевт“.

(Фигура 10)

A screenshot of a computer

Description automatically generated

Третата опция от административния панел на аптеката, където собственикът може да праща заявка за работа към някой от складовете, налични в системата.

(Фигура 11)

A screenshot of a computer

Description automatically generated

Панел за известия. (Фигура 12)

A screen shot of a computer

Description automatically generated

Страница, в която може да се види и променя информацията за съответния потребител.

(Фигура 13)

## Диаграми на анализа

Диаграма 1

A diagram of a diagram

Description automatically generated with medium confidence

A diagram of a medical procedure

Description automatically generatedНа диаграмата на последователността можем да видим възможностите на потребител с ролята „базов потребител“ (“BaseUser”).

На диаграмата на последователността можем да видим възможностите на потребител с ролята фармацевт.

A diagram with arrows and text

Description automatically generated

На диаграмата, която е част 1 от това, което може да прави управителят на склада, можем да видим някои от възможностите, които системата предоставя.

Останалите възможности можем да видим на диаграмата отдолу.

A diagram of a process

Description automatically generated

A diagram of a process

Description automatically generated

Тази диаграма показва част от възможностите, които собственикът на аптеката може да извършва в системата.

A diagram of a process

Description automatically generatedНа диаграмата са описани останалите възможности, които системата предоставя на собственика.

A screenshot of a computer screen

Description automatically generated

На клас диаграмата са представени моделите за работа с базите данни, като имаме няколко основни класове, един “value object”, както и помощни абстрактни класове.

“ApplicationDbContext”: Това е класът, който съдържа всичко нужно за работата с базите данни. Той наследява “IdentityDbContext<ApplicationUser>”, което помага при аутентикацията и ауторизацията, понеже благодарение на това наследяване, при създаване на базата се добавят нови 6 таблици, които са свързани с потребителите. Тези таблици служат за работа с ролите, клеймовете, съхраняват информация за потребителя, която е обвързана с неговата идентичност.

Конфигурационните файлове, които завършват на “EntityConfiguration” са създадени, за да можем да конфигурираме таблиците в базата. Такива са например “ActivityEntityConfiguration”, “DepotEntityConfiguration”, “MedicineEntityConfiguration”. В тях използваме “Fluent API”, което помага при конфигурирането на връзките и като цяло на всичко свързано с таблиците.

“ApplicationUser”: Този клас съдържа информацията за потребителя. Наследява класът “IdentityUser”, като по този начин получава няколко полета (“properties”), които представляват потребителя като имейл, юзърнейм и други. В “ApplicationUser” са добавени още 4 полета, които не са свързани аутентикацията и те са име, фамилия, линк за снимката и отработени часове.

Абстрактният клас “BaseEntity” имплементира интерфейса “IEntity” и в него се съхранява уникалния идентификатор на потребителя. Всички класове го наследяват пряко или непряко, защото е задължително те да имат “Id”.

public interface IEntity

{

string Id { get; set; }

}

public abstract class BaseEntity : IEntity

{

public string Id { get; set; }

}

Абстрактният клас “AuditableEntity” имплементира интерфейса “ IAuditableEntity”, както и “ BaseEntity”, като по този начин класовете, които наследят “AuditableEntity” ще имат допълнителни полета, които да следят историята на записа. Понеже този клас наследява “BaseEntity”, другите класове могат да наследят само “AuditableEntity”.

public interface IAuditableEntity

{

string? CreatedBy { get; set; }

DateTime CreatedOn { get; set; }

string? UpdatedBy { get; set; }

DateTime UpdatedOn { get; set; }

}

public abstract class AuditableEntity : BaseEntity, IAuditableEntity

{

public string? CreatedBy { get; set; }

public DateTime CreatedOn { get; set; }

public string? UpdatedBy { get; set; }

public DateTime UpdatedOn { get; set; }

}

Класът “Pharmacy”: представлява аптеките в системата. В него се съдържа информация като името на аптеката, описанието и, адреса и, кой е собственикът (основателят) на аптеката, какви лекарства са добавени към аптеката и информация, свързана с фармацевтите, които работят в аптеките.

Класът “Depot”: представлява складовете в системата. В него се съдържа информация като името на склада, адреса му, какви лекарства са добавени към него, кой му е управителя. Един склад може да има много лекарства, но само един управител.

Класът “BasicMedicine”: съдържа информация за името на лекарството и описанието му.

Класът “Medicine”: представлява лекарствата в системата. Той е свързан с класа BasicMedicine, като така има поле, което съдържа информацията за лекарството, като име и описание, и допълнителна информация като наличност, цена, срок на годност, дали е изтекъл срокът на годност.

Класът “Notification”: съдържа информация за известията, пратени от сървъра към аптеката или склада. В него можем да открием информация като текста на известието, кога е пратено, дали е обвързано с аптека или склад, дали е заявка за работа със склада, дали е предупреждение за изчерпано количество или изтекъл срок на годност.

Класът “Address”: този клас представлява “value object”. В базата данни този клас не се създава като таблица, а вместо колона address се създават четири полета, които са следните: address\_number, address\_street, address\_city, address\_country.

## Модел на съдържанието / данните

ER диаграмата (Диаграма 1) на базата данни описва всички таблици в нея, колоните и техните типове. В базата данни се създават общо 14 таблици.

Базата данни е в трета нормална форма, като се имат предвид следните форми на нормализация:

* 1ва нормална форма: Няма повтарящи се колони или редове и всеки атрибут да съдържа само неделими единични стойности.
* 2ра нормална форма: Всеки неключов атрибут да бъде зависим от пълния първичен ключ.
* 3та нормална форма: Задължава първичния ключ да бъде единственият фактор, който да влияе на всички неключови свойства. Това изключва наличието на транзитивни зависимости.

Таблица “AspNetRoles”: тази таблица съдържа ролите на потребителите в приложението. Има първичен ключ (колоната Id) и не съществува функционална зависимост между колоните в таблицата, което показва, че тя е в трета нормална форма.

Таблицата “AspNetUserLogins”: тази таблица съхранява информация за потребителите в системата за управление на идентичности.

Таблица „AspNetUserRoles“: тази таблица има две колони “UserId” и “RoleId”, като всеки запис отговаря на връзката между потребител и роля в системата. Нарича се “bridge table” и чрез нея се осъществява връзка много към много.

Таблица „AspNetUsers“: тази таблица съхранява информацията за потребителите в системата. Има много колони, някои от тях автоматично генерирани, други добавяни, за да може да работи системата по начина, по който се очаква.

Допълнителните таблици, като “AspNetRoleClaims”, “AspNetUserClaims”, “AspNetUserTokens” също съдържат инфорамция, свързана с идентичността на потребителя.

Таблица “RefreshTokens” : Тази таблица е много важна за аутентикацията и ауторизацията на потребителите. В нея се съдържат самия „refresh token“ (токен за опресняване), кога изтича и към кой потребител принадлежи. Връзките са следните: Един запис от таблицата “RefreshTokens” може да бъде асоцииран само с един потребител и един потребител може да има само един “refresh token”.

Таблица “BasicMedicines”: Тази таблица съхранява информация за лекарството като името и описанието му. Връзките са следните: Едно базово лекарство може да бъде асоциирано с много записи от таблицата “Medicines”, но един запис може да бъде асоцииран само с едно базово лекарство (запис от таблицата “BasicMedicines”).

Таблица “Medicines”: Тази таблица съхранява допълнителна информация за лекарството. Има няколко колони: “Price”, “ExpirationDate”, “Quantity”, “BasicMedicineId”, което е връзка към таблицата “BasicMedicines”. По този начин се постига нормализация, защото базовите лекарства не се повтарят. Таблицата “Medicines” има връзка към таблицата “BasicMedicines” и така се съхранява само допълнителна информация като количеството, цената и други, но името и описанието не се повтарят, понеже са отделени в отделна таблица. Има и още две връзки (в зависимост дали се асоциира с аптека или склад): DepotId, което остава NULL ако лекарството е асоирано със аптека, или PharmacyId, което остава NULL ако лекарството е асоциирано със склад.

Таблица “Activities”: В тази таблица се съхранява информацията за активността на фармацевта, като колоните са следните: “FirstRequestMade”, “LastRequestMade”, “UserId”, “Id”, като благодарение на тази таблица се следи работата на фармацевта. Връзките са следните: Една активност може да принадлежи към един фармацевт, но един фармацевт може да има много активности.

Таблица „Notifications“: В тази таблица се съхраняват всички известия, които са пратени към аптека, склад или и двете. Колоните са следните: “Id”, “Text”, “SentOn”, “IsWarning”, “IsAssignRequest”, “DepotId”, “PharmacyId”. Вързките са следните: Едно известие може да принадлежи само на една аптека или един склад или и двете, но склад или аптека могат да имат много известия.

Таблица “Pharamcies”: В нея се съхранява информацията за аптеките, които са регистрирани в системата, като се пазят данните за името на аптеката (“Name”), описанието и (“Description”), адреса и (“Address”), служителите и (“Pharmacists”), собственикът и (“FounderId”), към кой склад принадлежи (“DepotId ”). Връзките са следните: Една аптека може да има много лекарство, но едно лекарство може да приндалежи само към една аптека. Една аптека може да има много фармацевти, но един фармацевт може да работи само в една аптека. Една аптека може да има един собственик, но един потребител може да бъде собственик на много аптеки.

Таблица „Depots“: В нея се съхранява информацията за складовете, които са регистрирани в системата, като се пазят данните за името на склада (“Name”), управителят му (“ManagerId”), адреса му (“Address”). Връзките са следните: Един склад може да има много лекарства, но едно лекраство може да принадлежи само към един склад. Един склад може да има един управител, но един потребител може да управлява много складове.

Благодарение на нормализацията на базата в трета нормална форма, няма излишни данни, както и е малка вероятността да се получат аномалий с данните. Системата също работи много по-добре, съхранението на данни е по-консистентно.

# Дизайн

За направата на проекта съм избрал софтуерната платформа на .NET – ASP.NET Web API от към сървърна част. От към клиент, който да комуникира със сървъра съм използвал React.

За съхранението и управлението на данните е използван софтуера “SSMS” (SQL Server Management Studio). Тази платформа позволява лесното менежиране на данните.

Комбинацията от тези платформи осигурява една стабилна и добре работеща система. Използван е статичен анализатор в проекта, което подобрява качеството на кода и осигурява ползването на добрите конвенции при писане на код.

## Реализация на архитектурата на приложението

Проектът е реализиран на базата на трислойна архитектура, като при сървърната част слоевете са следните:

* PMS.Data, който е слоя за данни и в него се намират моделите, които служат за работа с базата данни, конфигурационните файлове, които конфигурират тези модели, сийдинг ресурсите, папката с миграциите, както и ApplicationDbContext.
* PMS.Services – Съдържа всички интерфейси и техните имплементации.
* PMS.Shared – Съдържа всички модели, конфигурации, константи, които са общи между няколко слоя.
* PMS.Tests – това е проектът, който съдържа тестовете за системата. На практика не е главен слой от архитектурата, но е част от системата.
* PMS.WebHost, който представлява уеб сървъра. Служи за връзката между бизнес слоя и клиента. В него се намират контролерите, които се явяват като крайни точки, към които клиента прави заявки.

Client (React), който играе ролята на презентационен слой и си комуникира със сървъра чрез HTTP заявки.

На следната диаграма може да се види архитектурата и да се запознаете с това как работи тя и зависимостите между тях. Също на диаграмата може да се видят и външните ресурси на които зависи системата, в случая на Microsoft Azure, като се използват Azure Blob Storage и Azure Communication Services. Като техните използваемости са следните:

* Azure Blob Storage – съхранение на снимките на лекарства
* Azure Communication Services – изпрашане на имейли (при споделяне на фактура или при добавяне на нов служител)

A diagram of a software company

Description automatically generated

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

PMS.Data (Слой за данни): Този слой съдържа моделите, който служат за работата с базата данни, конфигурационните файлове, които конфигурират полеата и връзките, както и сийдинг ресурсите, които са файлове, съдържащи готова информация, която се вкарва в базата данни. В случая в този слой се съхранява BasicMedicines.json, в който се съхранява информацията за базовите лекарства, която в последствие се вмъква в таблицата в базата. На клас диаграмта по-нагоре в докуменатацията (точка 3.3) може да се придобие представа за класовете в слоя за данни.

Използвани пакети в слоя за данни са:

* "Microsoft.AspNetCore.Identity.EntityFrameworkCore", като този пакет ни предоставя “ IdentityDbContext” класа.
* "Microsoft.EntityFrameworkCore": използва се за работата с базата данни и ORM
* "Microsoft.EntityFrameworkCore.SqlServer": служи за работа с “Microsoft SQL Server”
* "Microsoft.EntityFrameworkCore.Tools"

PMS.Services (Бизнес слоя): Този слой съдържа бизнес логиката на приложението. В него има папки, в които се съдържат интерфейсите, имплементациите, един статичен клас за dependency injection. В този слой се намира още и конфигурационния файл за AutoMapper.

В този слой се намират няколко папки и файлове:

* “Contracts” папката: В нея се съдържат всички интерфейси в системата, които след това се имплементират от класове. Така постигаме един от “SOLID” принципите, а именно “Interface Segregation”.
* “Implementations” папката: В нея се съдържат всички имплементации на интерфейсите, както и два сървиса, които работят на заден план “Background Jobs”, които работят с помощта на Quartz, което е пакет за .NET.
* “Mapping” папката се съдържат конфигурационните файлове за “AutoMapper”, на който идеята му е да конвертира моделите за работа с базата данни към DTO (“Data Transfer Object”) моделите. Използването на такъв пакет улеснява логиката на приложението, понеже той извършва трансформацията вместо нас.
* “Enums” папката съдържа енумерациите, които се използват само при сървисите. В случая има един такъв и той е “BuildingType” с две възможни опции в него: “Pharmacy”, “Depot”.
* DependencyInjection.cs файлът служи за инжектирането на сървисите в системата. По този начин ние можем да използваме “Dependency Injection”, като инжектираме сървисите в конструктора.

Пример за енумерация:

public enum BuildingType

{

Pharmacy,

Depot,

}

Използването на “Dependency Injection” позволява на класа да бъде по-лесен за тестване и по-гъвкав. Може лесно да бъде конфигуриран да използва други имплементации на неговите зависимости без да модифицираме класа, в този случай контролера.

Пример за “Dependency Injection”:

public AuthController(IAuthService authService, ITokenService tokenService, IUserService userService, ICurrentUser currentUser, ILogger<AuthController> logger, IActivityService activityService)

{

this.authService = authService;

this.tokenService = tokenService;

this.currentUser = currentUser;

this.userService = userService;

this.logger = logger;

this.activityService = activityService;

}

Някои от използваните пакети в слоя за данни са:

* "AutoMapper"
* "Azure.Communication.Email": служи за изпращане на имейли
* "Azure.Storage.Blobs": използва се за съхраняване на файлове в “AzureBlobStorage”
* "Handlebars.Net": използва се за да може динамично да се вкарат в данни в HTML темплейт. В случая на проекта има създаден HTML код, който представлява фактура, в който с помощта на handlebars слагаме данни.
* "Quartz.Extensions.Hosting": използваме за създаването на сървиси, които да работят на заден план
* "NReco.PdfGenerator": на базата на HTML код се генерира PDF
* "Newtonsoft.Json": за работа с JSON обекти
* "Microsoft.IdentityModel.Tokens": Използва се за работа с “JWT (Json Web Tokens)”.
* "Essentials": Пакет, който предоставя обекти, репрезентиращи резултати. Например, в него има клас “MutationResult”, който връща Id-то на обекта, който е променян или добавен и резултата от операцията.

На диаграмата по-долу може да се видят всички сървиси в бизнес слоя, както и кои интерфейси имплементират:

A diagram of a website

Description automatically generated with medium confidence

Презентационният слой е написан на Javascript базираната библиотека React, която позволява по-гъвкав фронт-енд с повече възможности. Работи на основата на SPA (Single Page Application), тоест се зарежда една HTML страница и в последствие динамично се актуализира съдържанието на страницата в зависимост от това как потребителя използва приложението. Използването на този клиент позволява работата с безброй многото пакети, които могат да се изтеглят и да улеснят работата, както и да я подобрят. Изтеглянето на пакети се осъществява чрез npm (node package manager).

Допълнителни проекти:

* PMS.Tests: Този проект съдържа всички тестове. Тестовете са направени върху бизнес логиката.
* PMS.Shared: Този проект съдържа всички общи обекти – тези които се използват в повече от един слой.
* PMS.WebHost (Уеб Сървър): Този проект, който може и да се сметне като част от презентационния слой, съдържа контролерите, които репрезентират крайните точки (endpoints), към които клиента прави HTTP заявки. В този слой се намират конфигурациите за swagger, както и за oauth2, благодарение на която можем да работим с “JWT (Json Web Token)”. В този проект се намира и файлът “Program.cs”, който служи за конфигуриране на “back-end”. В него можем да конфигурираме ауторизацията, базата данни, “Swagger”. “Swagger” предоставя много възможности на потребителите като генерирана докуменатация, описваща нашия “API”, интерактивен интерфейс и визуализация, стандартизация, както и възможности за валидации чрез използването на схеми.

Примерен контролер:

[ApiController]

[Route("api/invoice")]

public class InvoiceController : ControllerBase

{

private readonly IInvoiceService invoiceService;

private readonly IEmailService emailService;

private readonly IActivityService activityService;

private readonly ICurrentUser currentUser;

public InvoiceController(IInvoiceService invoiceService, IEmailService emailService, IActivityService activityService, ICurrentUser currentUser)

{

this.invoiceService = invoiceService;

this.emailService = emailService;

this.activityService = activityService;

this.currentUser = currentUser;

}

[HttpPost("generate")]

[Authorize]

public async Task<IActionResult> GenerateInvoiceAsync([FromBody] InvoiceIM invoiceIM)

{

var invoice = await this.invoiceService.GenerateInvoiceAsync(invoiceIM);

await this.activityService.ChangeLastRequestAsync(this.currentUser.UserId);

return this.Ok(Convert.ToBase64String(invoice));

}

[HttpPost("share")]

[Authorize]

public async Task<IActionResult> ShareInvoiceAsync([FromBody] ShareIM shareIM)

{

var emailRequest = new IEmailService.SendEmailRequest(shareIM.Email, "New invoice", "Hello, \r\n Check out this invoice!", shareIM.FileName ?? null, shareIM.Base64File ?? null);

await this.emailService.SendEmailAsync(emailRequest);

await this.activityService.ChangeLastRequestAsync(this.currentUser.UserId);

return this.Ok();

}

}

На кода можем да видим какво представлява един контролер. Той наследява “ControllerBase”, което показва, че контролера е за “API”, а не е типичния MVC контролер. В него се намират нужните сървиси под формата на “private readonly” полета. В конструктора се инжектират тези сървиси, така че да могат да работят и в самите методи ги извикваме при нужда.

[ApiController] – показва, че контролера е за “API”

[Route("api/invoice")] – показва базовия път до крайната точка (“endpoint”)

[HttpPost("generate")] – показва, че правим “POST” заявка в “route” – /api/invoice/generate, като се има предвид базовия път.

[Authorize] – показва, че за да се направи заявка към “endpoint”-а е нужно потребителят да е влязъл в системата.

“API” следва “RESTful” принципите. “HTTP” заявките, които могат да се направят към сървъра са: GET, POST, PUT, DELETE.

## Организация и код на заявките към база от данни

За работа с базите данни се използва Microsoft Entity Framework от към фреймуърк за работа с бази данни и Microsoft SQL Server е базата данни с която работи проекта. Подходът който е използван при направата на този проект е Code First, тоест първо се създават моделите в програмната среда и след това се генерира базата данни на базата на тези модели и конфигурациите им. За да създадем базата данни, ние използваме миграции, които са автоматично-генериран програмен код, който съдържа заявки, които когато изпълним ще създадат базата и таблиците в нея с връзките както ние сме ги задали.

Класовете “IdentityUser”, “IdentityDbContext” генерират таблици в базата, които да служат за аутентикацията. В проекта, “ApplicationUser” наследява “IdentityUser”, а “ApplicationDbContext”, който клас е в основата на работата с базите данни, наследява “IdentityDbContext<ApplicationUser>”

На кода по-долу може да се види какво представлява ApplicationDbContext. В конструктора му се подават опциите и след това ICurrentUser, което взима уникалния идентификатор на потребителя, който прави заявката и го дава на базата.

Например DbSet<RefreshToken>: това е колекцията от токени за опресняване, които са записани в базата. Посредством тази колекция, ние можем да добавяме, премахваме и да извършваме всякакви “CRUD” заявки със записите в DbSet-а. Всички таблици в базата, без ApplicationUser и помощните му таблици, се добавят под формата на DbSet-ове, за да може да работим с тях и те работят на същата основа като гореописаният “DbSet<RefreshToken>”.

Има помощни методи, които допълват определени таблици и техните записи с допълнителна информация като кой е добавил записа, кой го е променил последно, уникалнция идентификатор на потребителя, който е извършил заявката, което помага за придобиване на по-добра представа за това, което се случва в системата.

public class ApplicationDbContext : IdentityDbContext<ApplicationUser>

{

private readonly ICurrentUser currentUser;

private readonly EntityState[] auditableStates =

{

EntityState.Added,

EntityState.Modified,

};

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options, ICurrentUser currentUser)

: base(options)

{

this.currentUser = currentUser;

}

public virtual DbSet<RefreshToken> RefreshTokens { get; set; } = default!;

public virtual DbSet<Depot> Depots { get; set; } = default!;

public virtual DbSet<BasicMedicine> BasicMedicines { get; set; } = default!;

public virtual DbSet<Pharmacy> Pharmacies { get; set; } = default!;

public virtual DbSet<Medicine> Medicines { get; set; } = default!;

public virtual DbSet<Notification> Notifications { get; set; } = default!;

public virtual DbSet<Activity> Activities { get; set; } = default!;

public override int SaveChanges(bool acceptAllChangesOnSuccess)

{

this.HandleAuditableEntities();

return base.SaveChanges(acceptAllChangesOnSuccess);

}

public override int SaveChanges()

{

this.HandleAuditableEntities();

return base.SaveChanges();

}

public override Task<int> SaveChangesAsync(CancellationToken cancellationToken = default)

{

this.HandleAuditableEntities();

return base.SaveChangesAsync(cancellationToken);

}

public override Task<int> SaveChangesAsync(bool acceptAllChangesOnSuccess, CancellationToken cancellationToken = default)

{

this.HandleAuditableEntities();

return base.SaveChangesAsync(acceptAllChangesOnSuccess, cancellationToken);

}

protected override void OnModelCreating(ModelBuilder builder)

{

builder.ApplyConfigurationsFromAssembly(this.GetType().Assembly);

base.OnModelCreating(builder);

}

private void HandleAuditableEntities()

{

var userId = this.currentUser?.UserId?.ToString();

var now = DateTime.UtcNow;

var auditableEntries = this.ChangeTracker

.Entries()

.Where(x => x.Entity is IAuditableEntity &&

this.auditableStates.Contains(x.State))

.ToList();

foreach (var entry in auditableEntries)

{

var entity = entry.Entity as IAuditableEntity;

entity.UpdatedOn = now;

entity.UpdatedBy = userId;

if (entry.State == EntityState.Added)

{

entity.CreatedOn = now;

entity.CreatedBy = userId;

}

}

}

}

Част от миграция можем да видим на следния фрагмент от код:

migrationBuilder.CreateTable(

name: "AspNetRoles",

columns: table => new

{

Id = table.Column<string>(type: "nvarchar(450)", nullable: false),

Name = table.Column<string>(type: "nvarchar(256)", maxLength: 256, nullable: true),

NormalizedName = table.Column<string>(type: "nvarchar(256)", maxLength: 256, nullable: true),

ConcurrencyStamp = table.Column<string>(type: "nvarchar(max)", nullable: true)

},

constraints: table =>

{

table.PrimaryKey("PK\_AspNetRoles", x => x.Id);

});

На фрагмента код се вижда описана AspNetRoles таблицата и това как тя да се създаде в базата данни. Имаме име, колони и на всяка колона се задават изискванията като тип и други.

Освен това, базата се конфигурира и таблиците в нея също чрез конфигурационните файлове. Такъв може да се види на кода отдолу.

public class MedicineEntityConfiguration : IEntityTypeConfiguration<Medicine>

{

public void Configure(EntityTypeBuilder<Medicine> builder)

{

builder.HasKey(ms => ms.Id);

builder

.HasOne(ms => ms.BasicMedicine)

.WithMany()

.OnDelete(DeleteBehavior.Restrict);

builder

.Property(ms => ms.Price)

.HasPrecision(6, 2);

}

}

Този файл конфигурира таблицата “Medicines”, като казва каква ще е връзката и (едно към много) и забранява на базовото лекарство (‘BasicMedicine”) да бъде изтривано когато се изтрие запис от таблицата (“Medicines”). Това става чрез “.OnDelete(DeleteBehavior.Restrict)”.

Можем да задваме интервал и точност на полета, които са от реален тип, както е показано на снимката (“.HasPrecision(6, 2)”) .

Основни методи за добавяне, променяне, изтриване и извличане на обекти в системата са следните:

|  |  |  |
| --- | --- | --- |
| Функция | Методи | Описание |
| Създаване | CreateUserAsync(RegisterIM registerIM);  CreateActivityForUserAsync(string userId);  CreateDepotAsync(DepotIM depotIM, string managerId);  CreateMedicineAsync(MedicineIM medicineIM,BuildingType buildingType, string buildingId);  CreatePharmacyAsync(PharmacyIM pharmacyIM, string founderId); | Това са основни методи за създаване на записи в базата данни. Те работят асинхронно и приемат входни модели (input models), които репрезентират главния модел, който работи с базата, но съдържат само нужните полета за добавянето на нов запис. |
| Променяне | UpdateUserInfoAsync(string id, UserUM userUM);  ChangePasswordAsync(string userId, string newPassword);  UpdateDepotAsync(DepotUM depotUM);  UpdateMedicineAsync(MedicineUM medicineUM);  UpdatePharmacyAsync(PharmacyUM pharmacyUM); | Това са основните методи за променяне на записи в базата, като тези методи приемат ъпдейт модели (update models), които съдържат само нужните полета, които могат да се промеянт. |
| Изтриване | DeleteDepotAsync(string id);  DeleteMedicineAsync(string id);  DeletePharmacyAsync(string id); | Това са основни методи за триене на записи от базата. Те приемат уникалния идентификатор на записа в базата. |
| Извличане | GetAllDepotsAsync(); GetDepotsByUserIdAsync(string userId);  GetDepotByIdAsync(string id);  GetAllMedicinesAsync(BuildingType buildingType, string id);  GetMedicineByIdAsync(string id); GetBasicMedicinesAsync();  GetPharmacyByIdAsync(string id);  GetPharmaciesAsync(string userId);  GetPharmacistsAsync(string pharmacyId);  GetPharmaiesByDepotId(string depotId); | Това са основните методи за извличане на данни от базата данни. Някои от тях извличат по определни критерии, други връщат всички записи. Има възможност за връщане на запис, който отговаря на даден уникален идентификатор или всички записи, които са асоциирани с друга таблица от базата. |

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

Потребителският интерфейс е уеб-базиран и е написан на “React”, което позволява направата на един гъвкав и стилен дизайн с много функционалности.

Като влезем в системата за първи път ни отшраща към началната страница (Фигура 1), където е визуализирано името на системата и още информация. Горе в дясно се намира бутон „Започни сега“, който като го натиснем ни отправя към формата за влизане в системата (Фигура 2). В случай, че нямаме профил, под формата има линк за отвеждане към формата за регистрация (Фигура 3).

Опциите, които базовият потребител може да прави след като се регистрира е да влиза в системата (Фигура 2), да преглежда профила си (Фигура 13), както и да променя информацията си, като име, имейл, адрес, телефонен номер, както и паролата и да излиза от системата (Фигура 14).

Когато влезем в системата, можем да видим нашата главна страница, на която има един бутон с опции и един бутон за избиране на опцията (Фигура 4). Опциите са две: “Аптеки ” и “Складове”. Която и от двете да изберем, след това биваме препратени към нова страница, която съдържа в зависимост от избора ни аптеките, които имаме или складовете, които имаме. За аптеки – (Фигура 6), а за складове - (Фигура 5).В случай, че нямаме никакви аптеки, а сме избрали опцията за аптека, ще ни изпише следното съобщение „Няма налични аптеки !“, съответно при складовете “Няма налични складове !”. Отгоре на страницата имаме два бутона, които са за добавяне на аптека или склад и за връщане назад. При избор на добавяне ни отвежда към форма за добавянето на избраната опция от нас.

Когато преглеждаме таблиците с лекарствата, независимо дали сме в склада или в аптеката, можем да видим, че има четири цвята за редовете в самите таблици:

* Червен: показва, че лекарството е изчерпано (количеството е нула)
* Жълт: показва, че лекарството е с изтекъл срок на годност
* Зелен: показва, че лекарството е в наличност и не е с изтекъл срок на годност
* Жълто-червен (Преливане от червено към жълто): показва, че лекарството е изчерпано и с изтекъл срок на годност.

Ако сме избрали опцията за аптеки и вече имаме създадена аптека, ние можем да влезем в нея или да я изтрием, като при изтриване се появява модал, който ни пита дали сме сигурни. Когато влезем в аптеката като собственик, ние имаме две опции (главна страница (Фигура 8) и настройки (Фигура 9)), като достъпът до главната страница не ни е позволен докато не работим с някоя аптека. Като влзезем в настройките, ние имаме три опции: Да променяме информацията за аптеката като име и адрес (Фигура 9), да добавяме или премахваме работници (Фигура 10) и да подаваме заявка за работа със определен склад (Фигура 11). В случай, че вече работим с някой склад, ние можем да извършваме операции с лекарствата. Като изберем главната страница (Фигура 8), виждаме една таблица с лекарствата и под нея има опции: “Заяви от склад”, “Продай ”, “Презареди ”, “Повече ”, “Изтрий ”. Която и от опциите да изберем ни се отваря модал, в който да извършим следната дейност. При “Заяви от склад” правим заявка за някое лекарство от склада, при “Продай” ни се отваря нов прозорец, като там трябва да изберем количеството на всяко лекарството и като приключим ни се генерира фактура. Опцията “Презареди” изисква просто да изберем лекарство, да въведем количеството и в случай, че склада има даденото лекарство с по-голямо или равно количество, то се взима от склада и се добавя в аптеката. Опцията „Повече“ ни позволява да видим лекарството, неговото описание, начин на използване и състав, като можем и да променим снимката на лекарството. Последната опция за “Изтрий” ни позволява да изберем едно или повече лекарства в системата и да ги изтрием.

В случай, че искаме да видим складовете, първоначално трябва да изберем склад от каталога с нашите складове и да влезем в него (Фигура 5). Първоначално ни препраща към менюто (Фигура 7) за лекарствата в склада, от където можем да добавяме, премахваме и актуализираме лекарство чрез следните функции: “Добави лекарство”, “Изтрий” и “Повече”. Имаме възможност да променяме снимката, цената и бройката.

Общо имаме 4 опции за управлението на склада:

* Да преглеждаме и управляваме лекарствата (добавяне, изтриване, промяна на информацията)
* Да виждаме с кои аптеки работим (“Клиенти” опцията)
* Да преглеждаме и приемаме заявките за работа (“Assign Requests” опцията)
* Да преглеждаме историята на презареждане и зареждане на лекарства към аптеките с които склада е работил или все още работи. (“Requests History”)

В случай, че сме влезли в системата като фармацевт, ние имаме право само да влизаме в аптеката, в която сме добавени. Нямаме право да създаваме нова или да я трием. Когато влезем, ние нямаме достъп до настройките, а само до главната страница с лекарствата.

При изчерпано количество на лекарство в някоя от аптеките в системата, се изпраща нотификация на аптеката и на склада който работи с нея (Фигура 12).

# Ефективност и бързодействие на решението

Използвани са добри практики и помощни средства, които подобряват ефективността и бързината на системата. Entity Framework подобрява работата с базата данни, като предоставя изключително много възможности, които подобряват ефективността. В сървъра са създадени два сървиса, които работят на заден план:

1. Да се проверява всеки ден в 09:00ч. за лекарства, на които им е изтекъл срока на годност.
2. Да се прегледжда всеки понеделник, точно в 00:00ч активността на фармацевтите и да се добавят работните часове, които всеки фармацевт е изработил през седмицата към техния профил, за да може да се следи работното време.

Благодарение на използването на токените за достъп и опресняване (AccessToken и RefreshToken), които се използват за аутентикация и ауторизация, сигурността е подобрена, работата с ролите е значително по-ефективна. Комуникацията между сървъра (ASP.NET Web API) и клиентът (React) е значително по сигурна и ефикасна, благодарение на тези токени.

Фактурите също се генерират от сървъра и се пращат в base64 формат на клиента, което също подобрява ефективността на системата, понеже клиента не се натоварва със сложна логика.

Клиентът (React) позволява направата на добър интерфейс, които е интуитивен и лесен за ползване.

# Тестване

За момента са тествани някои от сървисите, представляващи бизнес логиката на проекта. За тестовете е използван Xunit, както пакета Moq, който позволява по-лесната работа със сървисите. Също за тестовете се използва същата база, но в този случай работи в паметта (In Memory Database). За тестовете в папка са отделени фалшиви обекти, които репрезентират реални такива, за да може да се инжектират в сървисите. Такива са FakeCurrentUser, FakeRoleManager, FakeUserManager, които наследяват в същия ред: ICurrentUser, UserManager<ApplicationUser>, RoleManager<IdentityRole>.

Тестовите случаи са следните и са представени в таблицата долу:

|  |  |  |
| --- | --- | --- |
| Сървис, който се тества | Тестов метод | Описание |
| DepotService | CreateDepotAsync\_WithValidData\_  ShouldSucceed() | Чрез този метод се тества дали складът се създава успешно с подадени валидни данни. |
| DepotService | UpdateDepotAsync\_WithValidData\_  ShouldSucceed() | Чрез този метод се тества дали складът се променя успешно при валидни данни. |
| DepotService | DeleteDepotAsync\_ShouldSucceed() | Чрез този метод се тества дали складът се изтрива успешно при валиден уникален идентификатор. |
| DepotService | GetDepotById\_WithValidId\_  ShouldReturnDepot() | Чрез този метод се тества дали взимането на склад по уникален идентификатор работи успешно. |
| DepotService | GetDepots\_ShouldReturnDepots() | Чрез този метод се тества дали взимането на всички складове работи успешно. |
| PharmacyService | CreatePharmacyAsync\_WithValdData\_  ShouldSucceed() | Чрез този метод се тества дали създаването на аптека с валидни данни работи успешно. |
| PharmacyService | UpdatePharmacyAsync\_WidthValidData\_  ShouldSucceed() | Чрез този метод се тества дали аптеката се променя успешно при подаването на валидни данни. |
| PharmacyService | DeletePharmacyAsync\_WidthValidId\_  ShouldSucceed() | Чрез този метод се тества дали аптеката се изтрива успешно при валиден уникален идентификатор. |
| NotificationService | GetAssignRequestsAsync\_ShouldSucceed() | Чрез този метод се тества дали ще се получат всички заявки за работа подадени от аптеките към определн склад. |
| NotificationService | GetWarningsAsync\_ShouldSucceed() | Чрез този метод се тества дали ще се получат всички известия, които са предупреждения, свързани с липса на количество или изтекъл срок на годност, изпратени към аптеката. |
| NotificationService | GetNotificationsForPharmacyAsync  \_ShouldSucceed() | Чрез този метод се тества дали ще се получат всички нотификации, които са асоциирани само със с аптека. |
| NotificationService | GetNotificationsForDepotAsync  \_ShouldSucceed() | Чрез този метод се тества дали ще се получат всички нотификации, които са асоциирани само със склад. |

Проектът може да се тества в различни браузъри като Google, Firefox, Chrome, Microsoft Edge, Safari, Internet Explorer, за да се потвърди, че функционалността работи правилно и елементите на дизайна се поддържат във всички браузъри.

При извършване на тестовете за производителност е важно да се проверят времето за отговор на сървъра (Web API), времето за отговор на клиента (React), времето за зареждане на страниците, както и други параметри, които могат да доведат до забавянето на уебсайта.

Пример за тест:

[Fact]

public async Task GetAssignRequestsAsync\_ShouldSucceed()

{

var depot = new Depot

{

Id = "1",

Name = "test",

ManagerId = this.currentUser.UserId,

};

await this.context.Depots.AddAsync(depot);

var notification = new Notification

{

Id = Guid.NewGuid().ToString(),

IsAssignRequest = true,

Text = "www",

SentOn = DateOnly.FromDateTime(DateTime.Now),

DepotId = depot.Id,

};

await this.context.Notifications.AddAsync(notification);

await this.context.SaveChangesAsync();

var notifications = await this.notificationService.GetAssignRequestsAsync("1");

Assert.True(notifications.Count == 1);

}

При използването на “XUnit” като “framework” за създаване на тестове, за да обозначим, че метода е тест, използваме атрибута [Fact]. На самия код можем да видим тест за получаване на всички заявки за работа с определен склад. В случая към склада добавяме една заявка и след това с помощния клас “Assert” и един от методите “True” проверяваме дали броя на нотификациите (заявките за работа), които са асоциирани със склад със уникален идентификатор „1“ са една на брой. В случай, че са само една и условието е изпълнено, ще върне “true”, а в обратен случай “false”.

# Заключение и възможно бъдещо развитие

Реализираното решение предоставя един добър софтуер с много удобни функционалноти и изчистен дизайн. Спрямо проучванията за складовете и аптеките, които се занимават с лекарства, този проект изпълнява всичко нужно, дори и повече от очакваното.

Резултатите от работата върху проекта са разработена функционална платформа, която дава възможност на потребителите да създават складове и аптеки и да ги оправляват. Платформата предлага няколко нива на достъп, като управителят на склада има пълен достъп до всичко в склада, собственикът на аптеката съответно до всичко в аптеката, фармацевтите имат достъп само до всичко свързано с управлението на лекарствата (нямат достъп до настройките, които се явяват и административен панел). Разработеното приложение улеснява много работата на всяка една от ролите.

Предимства на използваните технологии могат да бъдат огромната възможност за интеграции, понеже .NET е платформа на “Microsoft”, която дава много предимства. Използването на “framework” като “React” също е предмиство, защото има добра поддръжка и често се пускат нови версии.

Алтернативи на използваните технологии могат да бъдат програмни ецизи като “Python”, “Java”. Предимства на тези алтернативи включват по-голяма гъвкавост в някои аспекти.

Подобни решения имат широко приложение в практиката, като предоставят на потребителите си много възможности.

За бъдето развитие може да се добавят повече функционалности, както и да се подобрят бързината и сигурността на програмата.

# Използвани литературни източници и Уеб сайтове

# Приложения