ПРОФЕСИОНАЛНА ГИМНАЗИЯ ПО КОМПЮТЪРНО ПРОГРАМИРАНЕ И ИНОВАЦИИ
бул. "Захари Стоянов", жк Меден рудник, 8009 Бургас, office@codingburgas.bg, codingburgas.bg

ДИПЛОМЕН ПРОЕКТ
ЗА ДЪРЖАВЕН ЗРЕЛОСТЕН ИЗПИТ

по професия код 481030 „Приложен програмист“
специалност код 4810301 Приложно програмиране“

ТЕМА: „Система за управление и контрол на работни процеси“

Автор:
Иван Николаев Михайлов, клас XII^{в}

Ръководител:
Даниела Пендашева

Бургас

С Ъ Д Ъ Р Ж А Н И Е

1 Увод

Управлението и контролът на отделните работни процеси е в основата на управленската философия на всяка една организационна единица, независимо от това дали става въпрос за обществена институция или за бизнес ориентирана организация.

Управлението е процес на упражняване на ръководни дейности в рамките на една конкретна система, с цел организиране на дейността на тази система – опазване на структурата й, разпределяне на отговорностите и синхронизация на участниците в нея, поддръжка и развитие на дейността й.

Контролът пък от своя страна е основна управленска дейност, чрез която се упражнява целенасочено въздействие за регулиране и оптимизиране на процесите във всяка една организация. Контролът се осъществява чрез планирани и приложени контролни дейности на всяко ниво и във всеки работен процес, които се въвеждат, за да се повиши ефективността от дейността на всеки процес в организацията и съответно да допринесе за постигане на нейните цели.

В сферата на публичния сектор системите за вътрешно управление и контрол са част от управленската отговорност, основен ангажимент на ръководителя на организацията със Закона за финансово управление и контрол в публичния сектор. На всеки ръководител на публична организация е възложено разработването и приемането на процедури и предприемане на контролни дейности не само във финансовата сфера, но във всеки един работен процес, за да се гарантира неговата ефикасност и ефективност.

Сходна е ситуацията и в бизнес организациите, при които се въвеждат Системи за управление на качеството, в съответствие с международните стандарти, като и при тях основната цел е подобряване и проследимост на процесите, които протичат в една организация.

Ето защо темата за разработване на система за управление и контрол на работните процеси, която е предмет на настоящия дипломен проект е от изключителна важност и актуалност за всяка една организация, независимо дали е с обществена насоченост или бизнес е ориентирана.

Необходимостта от ефективно управление на организациите и съответно на ресурсите, изисква да се разработят и приложат интегрирани решения за управление на работните задачи.

Редът и начинът за осъществяване на управление и контрол на работните задачи са решение на всяка една организация. В публичния сектор, който е изцяло с йерархична структура все още документооборотът и съответно поставянето и проследяването на работните задачи се извършва на хартиен носител. Това е и основното предизвикателство, с което се сблъскват публичните институции и част от бизнес компаниите, а именно количеството документи, които се генерират, обработват и съхраняват ежедневно.

В организациите с йерархична структура, управлението на работните процеси е свързано с ясно поставяне на задачите на всяко едно ниво, съответно проследяване на изпълнението като качество и като срочност на изпълнението.

Основните методи, които се прилагат за управление на работните процеси в организациите от публичния сектор, и в част от бизнес компаниите с по-големи структури, като се започне от поставянето на задачата и съответно се стигне до отчитане на изпълнението й са чрез разпределяне на документи и чрез провеждане на работни срещи, тип „оперативка“ със служители на ръководни длъжности на отделните йерархични нива.

Технологичното развитие, изискванията за въвеждане на електронното управление в публичния сектор и не на последно място Covid пандемията наложиха въвеждането на хибридна форма на работа – дистанционна работа и в реална работна среда. Това съответно изисква промяна в организацията на работа, адаптиране на работните процеси, разработване и въвеждане на нови технологични решения, адаптирани изцяло в електронна среда.

Предложената Система за управление и контрол на работни процеси представлява интегриран подход за ефективно управление на дейностите и контрол на процесите в една организация. Системата включва софтуерни компоненти, които работят с цел подобряване на производителността, оптимизиране на ресурсите и улесняване на взаимодействието между различните части на бизнеса.

Системата за управление и контрол на работни процеси предоставя възможност на работодателите да намалят документооборота, да сведат до минимум хартиените документи, както и да проследяват работните процеси в организацията изцяло в електронна среда.

В настоящия документ са описани поставените цели, предложените решения, реализации и анализи, чрез UML диаграми, тестове и разсъждения. Целта е да се

представи комплексен и интегриран подход на работните процеси, който да отговаря в максимална степен на потребностите на организациите.

2 Цели и обхват на софтуерното приложение

Основната цел на настоящия проект е да се предложи софтуерно приложение, представляващо електронна система за управление и контрол на работните процеси, чрез което ще се увеличи ефективността от управлението и осъществяването на контрол при изпълнение на работните процеси в една организационна структура.

Основната задача, която се стреми да реши софтуерното приложение, предмет на настоящия проект, е въвеждане на ефективни подходи за реализация на управленските процеси в дадена организация, като предложи електронна платформа за разпределение, управление и контрол на изпълнението на работни задачи.

Приложението се стреми да предложи интегрирано и ефективно решение, което да автоматизира и оптимизира обработката и управлението на работните процеси, което да помогне организациите да постигнат по-голяма ефективност и ефикасност в управлението си и в използването на ресурсите.

Въвеждането на приложението ще реши няколко основни проблема и предизвикателства, пред които е изправена всяка една организация. На първо място това е ограничаването на хартиения документооборот, което води до иразходване и на повече материални ресурси. На второ място приложението ще реши и друг съществен проблем, свързан със загубата на повече време от служителите на ръководни длъжности, на които им се налага да провеждат работни срещи, да разпределят задачи, да проследяват срокове и да контролират изпълнение.

Не на последно място, въвеждането на системата ще сведе до минимум опасността от неизпълнение на поети ангажименти, поради пропускане или забавяне на срокове или загуба на документи.

В системата са планирани няколко нива на достъп, които се определят според различните роли на потребителите:

- Ръководител
- Служител
- Администратор

Обхвата на приложението, с включените роли и съответно нива на достъп са разработени така, че да постигнат поставените подцели за ефективно внедряване и използване на системата, а именно:

- Да се използва многослойна архитектура за създаване на приложението, което за изработване на презентационния слой използва MVC(Модел, Изглед, Контролер) архитектурата на действие. Базите данни да се управляват от Entity Framework и да са с Microsoft SQL Server.
- Да се създаде обектно-релационен модел за връзка на базата данни със средата за програмиране. Да се подготвят отчети и заявки за извличане на данни необходими за визуализирането им.
- Администраторът да осигурява техническа поддръжка на системата, да добавя и изтрива потребители.
- Системата да поддържа възможност за упълномощаване и удостоверяване.
- Системата да има три нива на достъп – администратор, ръководител и служител. Всеки да вижда само определени части от приложението и да извършва действия само предназначени за тях.
- Системата да може да филтрира, сортира и обработва данните по различни критерии.
- Да бъде разработена функционалност, която позволява запис и работа с дати.
- Да бъде разработена възможност за отпечатване на документи.
- Да се спазят добрите практики и основни принципи за писане на качествен програмен код.

3 Анализ на решението

Системата предоставя бързо и ефикасно решение за управление и контрол на работните процеси в една организация.

3.1 Потребителски изисквания и работен процес

Входни данни:

Входните данни в системата представляват информация за задачи, работни процеси, потребители и групи.

Данните се получават от ръководителите или администраторите, които въвеждат информацията в системата или пренасят данните от външни източници.

Обработка и запазване в системата:

В системата данните се обработват и запазват в съответните бази данни или хранилища на данни.

Служителите могат да маркират задачите като изпълнени, като това обновява съответния запис в системата.

Изходни данни:

Изходните данни включват актуализираната информация за задачите и работните процеси, както и списъци с потребители и групи.

Тези данни могат да бъдат използвани за визуализация на статуса на проектите, за вземане на решения и за комуникация между потребителите на системата.

Роли на потребителите и функционалности:

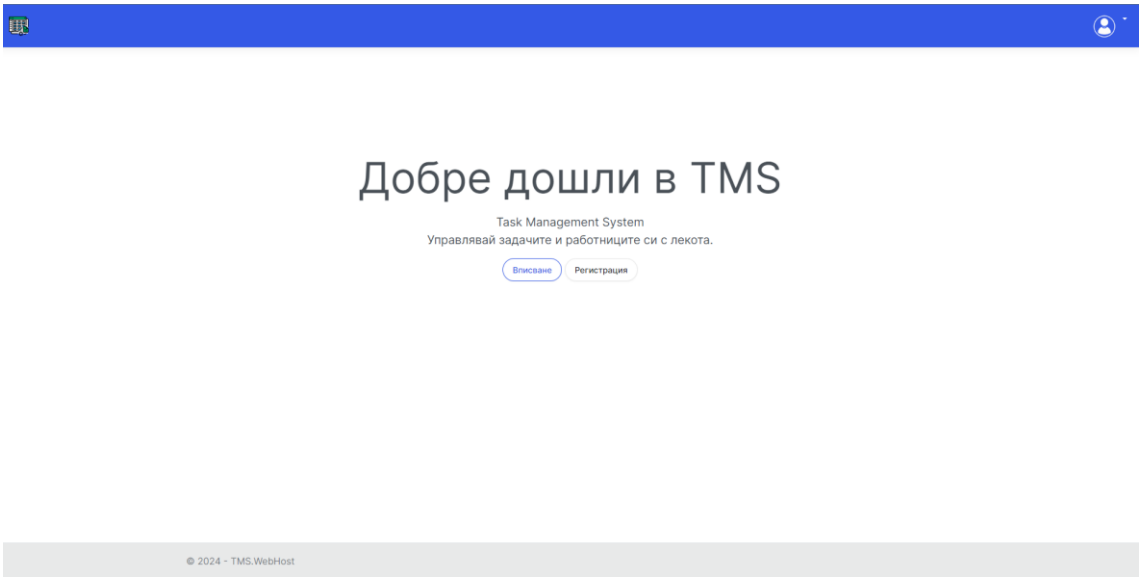
1. Служител, който има следните права и функционални възможности за достъп до системата:
   - Преглеждане на групи, в които е член.
   - Маркиране на задачи като изпълнени.
2. Ръководител, който има следните права и функционални възможности за достъп до системата:
   - Създаване на групи и добавяне на потребители към тях.
   - Обновяване на информацията на групите и потребителите.
   - Създаване, обновяване и преглеждане на информация за работните процеси.
   - Възлагане на задачи и работни процеси на групи и на отделни служители.
3. Администратор, който има следните права и функционални възможности за достъп до системата:
   - Обновяване и преглеждане на информацията за потребителите.

- Упълномощаване на потребители, като ги превръща в ръководители.

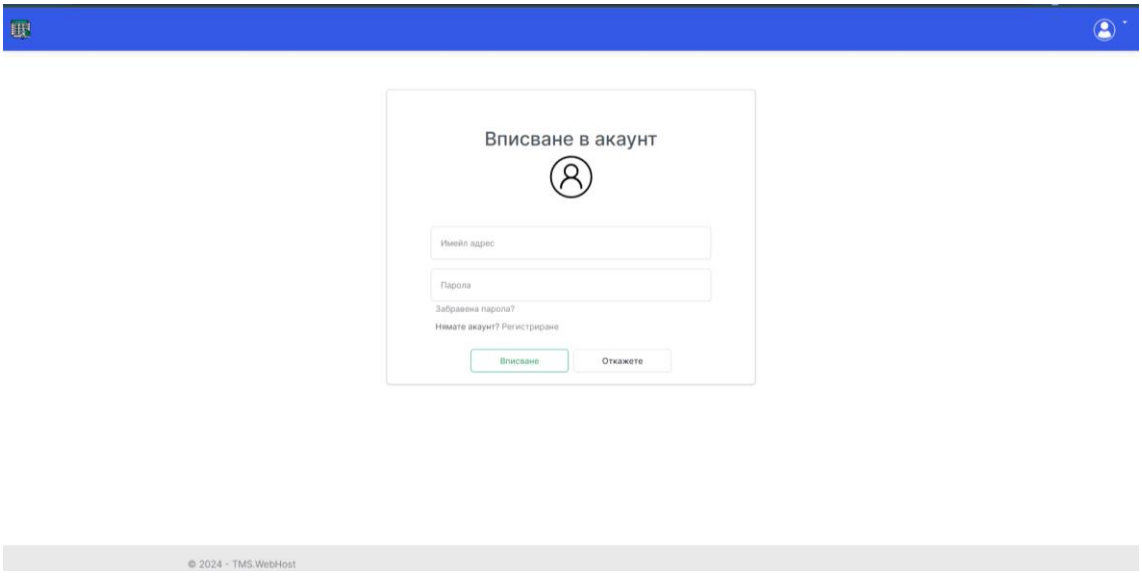
Използване на изходните данни:

- Визуализация на текущите работни процеси и статусите на задачите.
- Визуализация на всички налични работници.
- Оптимизация на работните процеси и постигане на по-голяма ефективност в организацията.

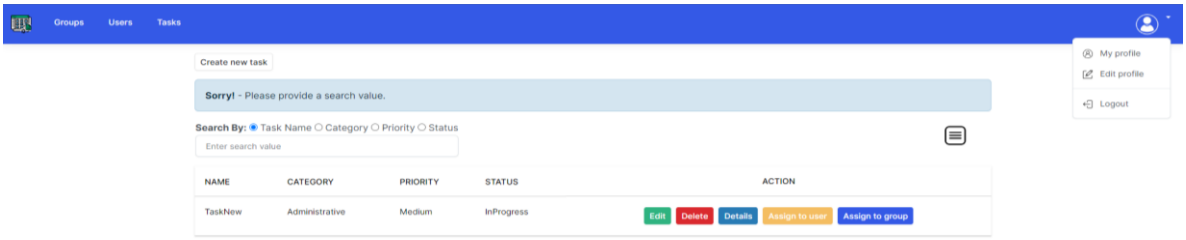
3.2 Примерен потребителски интерфейс



Фигура 1 Снимка на главната страница на приложението.



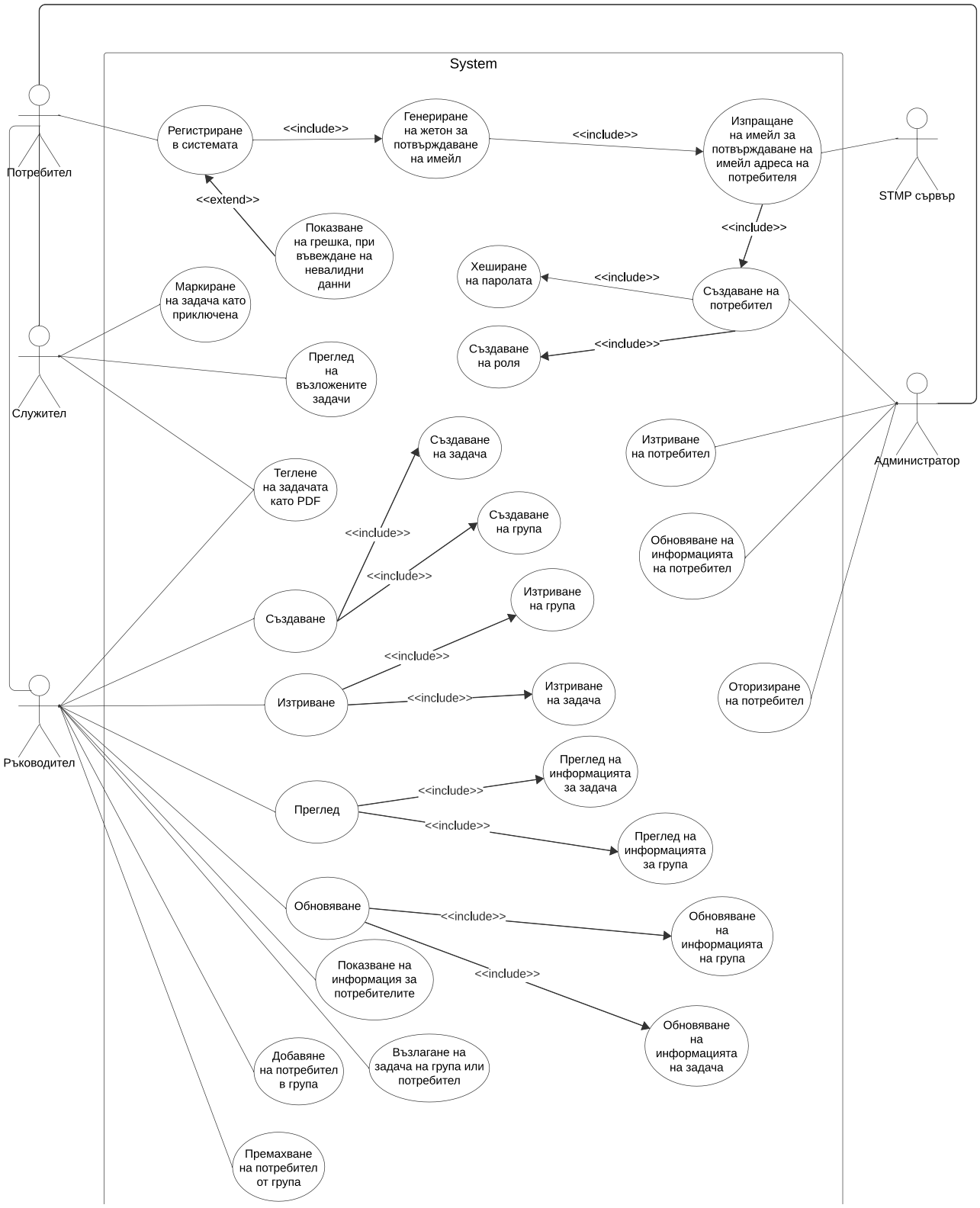
Фигура 2 Снимка на главната страницата за вписване на приложението.



Фигура 3 Снимка на страницата за операции със задачи.

3.3 Диаграми на анализа

3.3.1 Диаграма на случаи за употреба



Фигура 4 Диаграма на случаите за употреба.

Общото между всички потребители е, че имат възможността да се регистрират в системата.

След регистриране в системата, потребителят с роля администратор има право да създава други потребители, като всеки потребител получава една от трите възможни роли – Администратор, Служител или Ръководител.

Притежаването на различна роля, позволява на потребителя да извършва определен набор от операции, като по-долу са описани различните правомощия на всяка една от ролите.

1. **Потребител (Общи права за всички):**

- **Регистриране в системата:**
  - При правилно въведени данни: След което бива генериран жетон (token) за потвърждаване на въведения имейл. След това SMTP сървърът изпраща имейл, който потребителят трябва да отвори, за да потвърди въведената информация. Паролата на потребителя се пази под формата на символен низ, като преди това е хеширана, а ролята му се определя според това дали е първият регистриран в системата или не. Ако е – получава роля Администратор, ако не е – получава роля Служител.
  - При грешно въведени данни: Визуализира се поле с информация за грешката, след което потребителят е необходимо да въведе данните си повторно.

2. **Администратор:**

- Да създава потребители, на които се възлага различна роля. Паролата на потребителите бива хеширана, като тя бива пазена под формата на символен низ в базата данни.
- Да изтрива потребители.
- Да обновява информацията за потребителите.
- Да оторизира потребителите, като оторизирането се изразява в сменянето на ролята на потребител от Служител на Ръководител.

3. **Ръководител:**

- Да извършва операции за създаване на задачи:
  - Създаване на задача, като тя се характеризира чрез име, описание, начална дата, дата на приключване, категория, приоритет и статус.
  - Създаване на група, като тя се характеризира чрез име.
- Да извършва операции за изтриване:
  - Изтриване на задача.

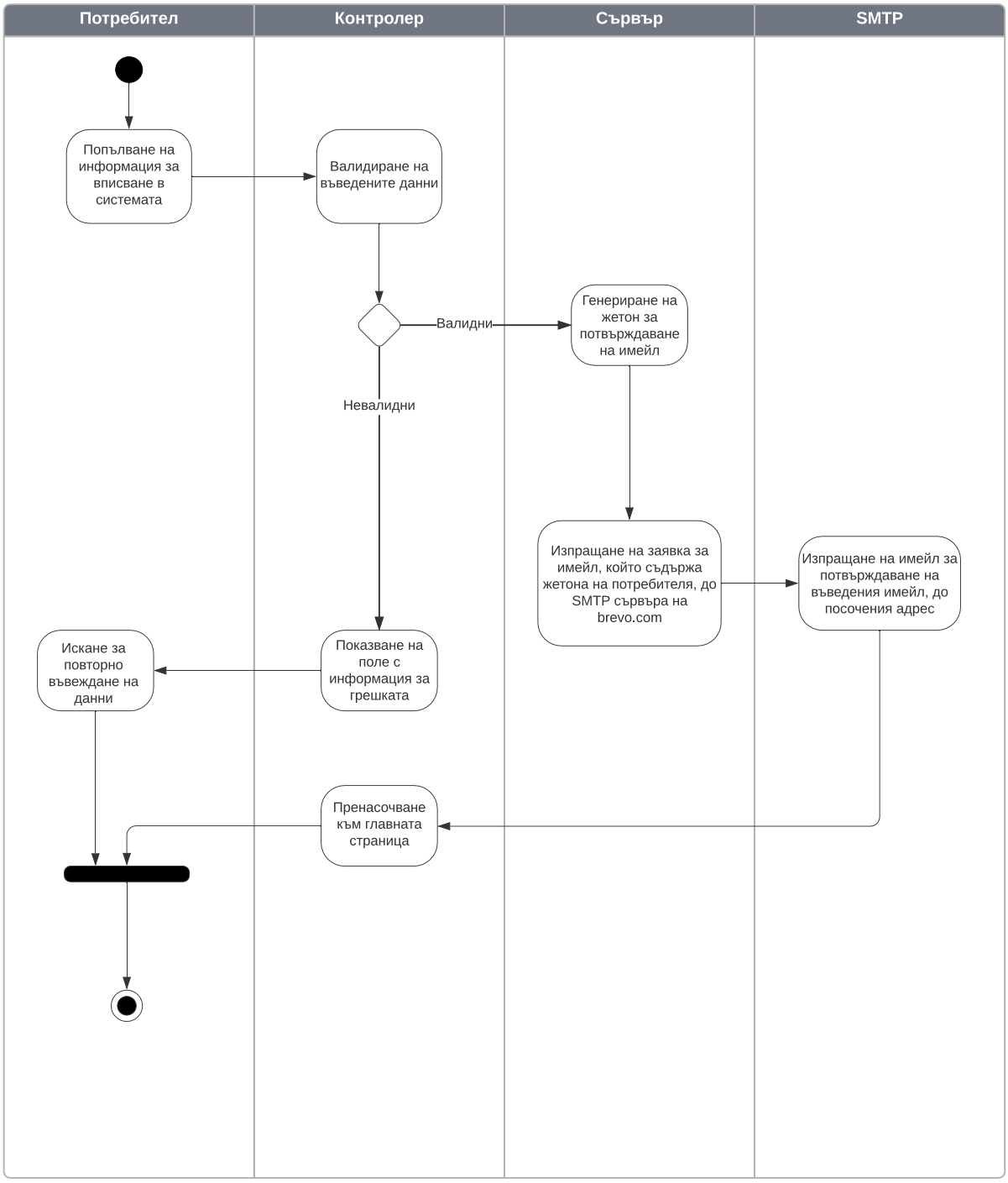
- Изтриване на група.

- Да извършва операции за обновяване на информация:
  - Обновяване на информация за задача.
  - Обновяване на информация за група.
- Да извършва операции за преглед на информация:
  - Преглед на информация за потребител.
  - Преглед на информация за задача.
  - Преглед на информация за група.
- Да възлага задачи на група или потребител.
- Да добавя потребители в група.
- Да премахва потребители от група.

4. Служител:

- Да преглежда информация за групите, в които членува.
- Да преглежда информация за възложените му задачи.
- Да маркира възложените му задачи като приключени.
- Да тегли информацията за възложената му задача в PDF формат.

3.3.2 Диаграма на дейността



Фигура 5 Диаграма на дейността.

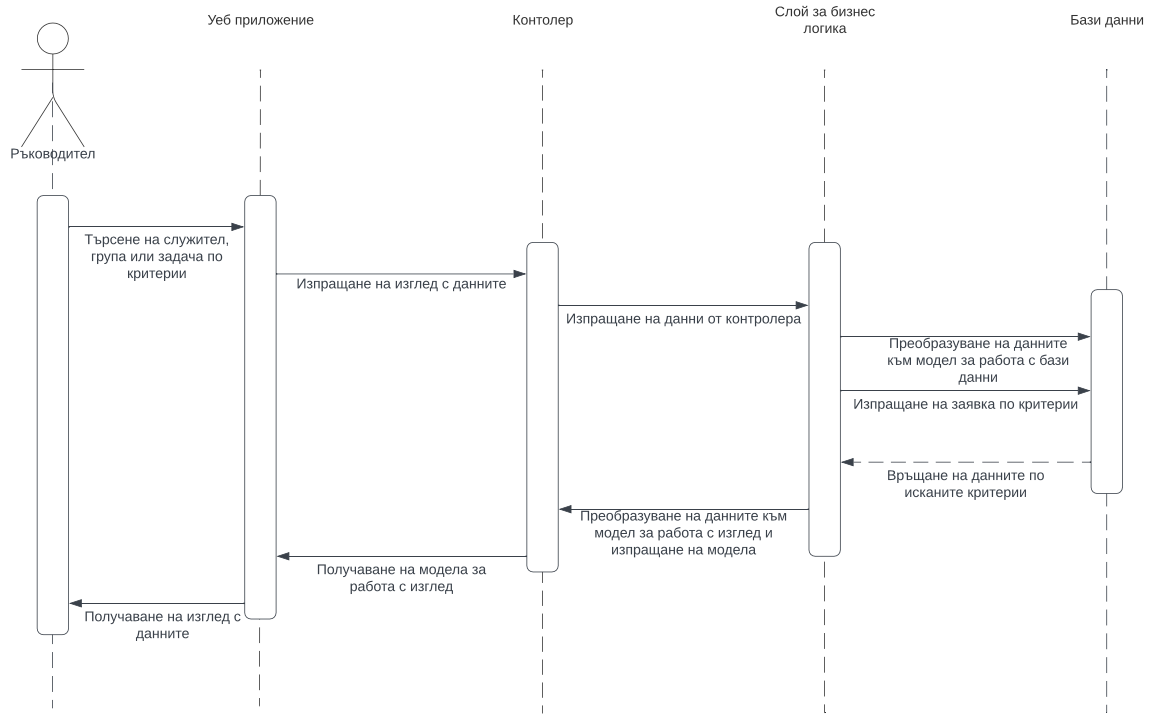
Диаграмата на дейността описва протичането на процеса регистриране. Регистрирането на един потребител се извършва, когато той въведе своите данни. Контролерът проверя валидността на тези данни.

- При случай на валидни данни, контролерът изпраща данните на сървъра, където бива генериран жетон (token) за потвърждаване на имейл.

SMTP клиентът изпраща имейл на вече регистрирания потребител, като използва генерирания жетон, за потвърждаване на имейл. При успешно изпращане, потребителят получава имейл, в който са му зададени указания за потвърждаване на въведения от него имейл адрес. Потребителят бива препратен към главната страница на приложението, където вече може да се впише с потвърдения си имейл адрес.

- При случай на невалидни данни или грешно въведени данни, се визуализира поле с информация за грешката, след което, от потребителят се изисква повторно въвеждане на данните.

3.3.3 Диаграма на последователността



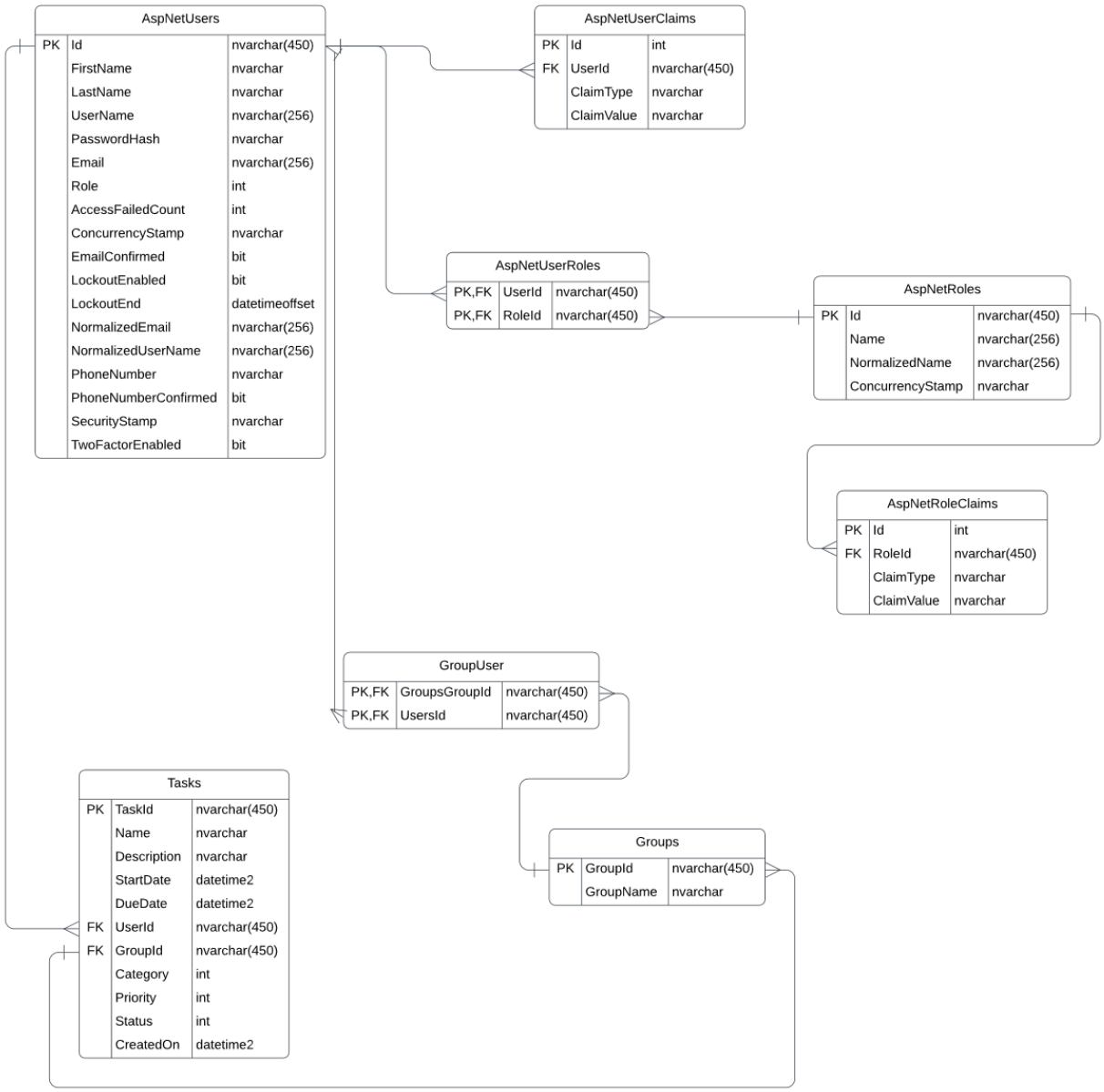
Фигура 6 Диаграма на последователността.

Диаграмата на последователността показва най-фундаменталната и най-често използвана операция в системата – работа с данни. Една обикновена заявка протича в следните стъпки:

- Потребителят, в конкретния случай с роля ръководител, използва системата за да достъпи определена информация относно служител, задача или група.
- Контролерът получава данните от изгледа и ги изпраща на слоя за бизнес логика.
- Данните биват преобразувани в модел за работа с бази данни (Domain model). След това към базата данни се изпраща заявка за извличане на данни, според въведените от потребителя критерии.
- Данните биват изпратени от базите данни към слоя за бизнес логика, където биват преобразувани към модел за работа с изглед и след това изпратени към контролера.
- Контролерът получава данните и ги изпраща към уеб приложението, където потребителя получава изглед с данните.

3.4 Модел на съдържанието / данните

3.4.1 Диаграма на отношение между моделите



Фигура 7 Диаграма на отношение между моделите.

3.4.2 Нормализация на базите данни

Базата данни е организирана колекция от структурирани информация или данни, обикновено съхранявани електронно в компютърна система. Базата данни се контролира от система за управление на бази данни (СУБД).

Данните в най-общите типове бази данни, които се използват днес се моделират в редове и колони в поредица от таблиците, за да се направи обработката и заявките за данни ефективни. След това данните могат да бъдат лесно достъпни, управлявани, променяни, актуализирани, контролирани и организирани. Повечето бази данни използват структурирания заявков език (SQL) за записване и заявки на данните. (Oracle, n.d.).

Нормализацията е процесът на организиране на данни в база данни. Тя включва създаването на таблиците и установяването на връзки между тях според правила, предназначени както да защитават данните, така и да направят базата данни по-гъвкава, като елиминират излишното дублиране и несъответствието на зависимостта.

Излишните данни заемат дисково пространство и създават проблеми при поддръжката. Ако данните, които съществуват на няколко места, трябва да бъдат променяни, те трябва да бъдат променени по точно същия начин на всички места.

Съществуват няколко правила за нормализация на базата данни. Всяко правило се нарича "нормална форма".

Ако се спазва първото правило, базата данни се счита за в "първа нормална форма". Ако се спазват първите три правила, базата данни се счита за в "трета нормална форма".

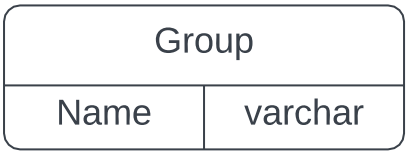
Въпреки че са възможни и други нива на нормализация, третата нормална форма се счита за най-високото ниво, необходимо за повечето приложения и поради тази причина, проектът е сведен до трета нормална форма. (Microsoft, Description of the database normalization basics, 2023).

Както и при много формални правила и спецификации, реалните сценарии не винаги позволяват перфектно спазване. Нормализацията изисква допълнителни таблици и ако бъдат нарушени някои от първите три правила на нормализация, е възможно да възникнат проблеми като наличие на излишни данни и несъответствие на връзките и зависимостите.

Първа нормална форма:

- Премахване на повтарящите се групи в една таблица.

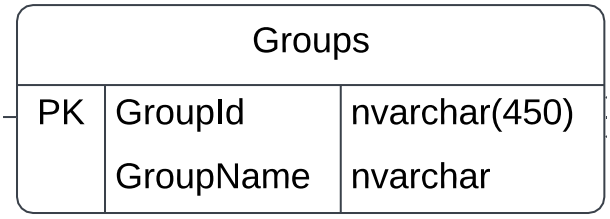
- Създаване на отделна таблица за всяка различна съвкупност от данни.
- Идентифициране всякаква различна съвкупност от данни с първичен ключ.



Фигура 8 Таблица в ненормализирана форма.

Представената извадка от данни визуализира таблица, която е ненормализирана. Няма колона, която може да идентифицира уникално името на групата, което позволява повтаряне на имената.

За да бъде приведена дадената таблица в първа нормална форма, трябва да бъде добавена колона Id, която да служи като уникален идентификатор на групите и като първичен ключ, който позволява свързването и връзката с други таблици. Ако таблицата беше приведена в първа нормална форма, тя щеше да изглежда така:

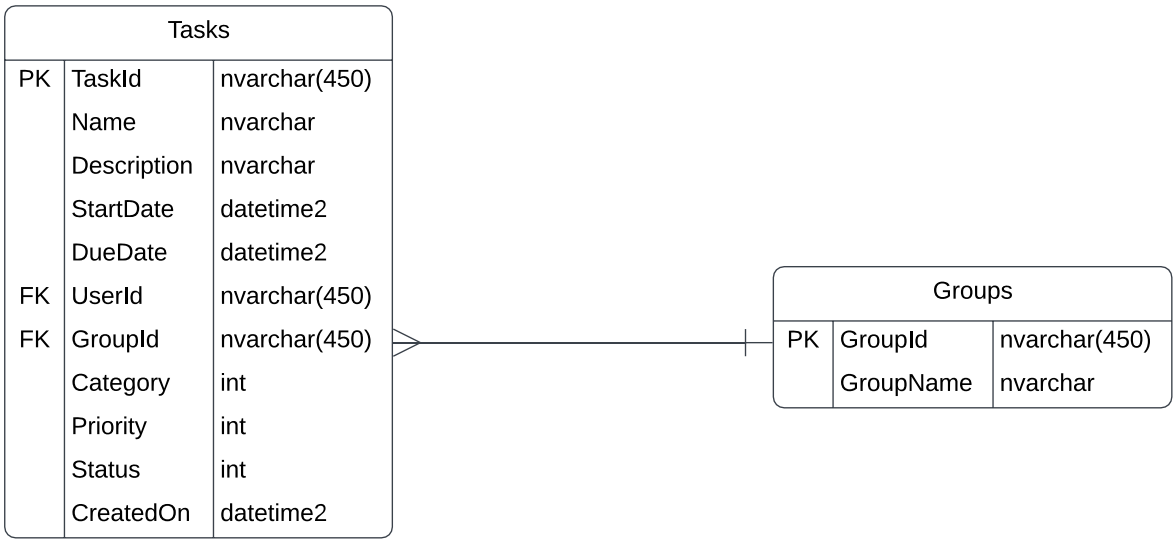


Фигура 9 Таблица в първа нормална форма.

Тук таблицата Groups, освен, че използва правилната конвенция за наименуване на таблици в базата данни, също прилага и първа нормална форма. Имаме уникален идентификатор GroupId, който е от тип nvarchar, тъй като се пази под формата Guid, с цел безопасност на данните. По този начин гарантираме уникалност.

Втора нормална форма:

- Нормализиране в първа нормална форма.
- Всички атрибути на таблицата да са функционално зависими от първичния ключ.

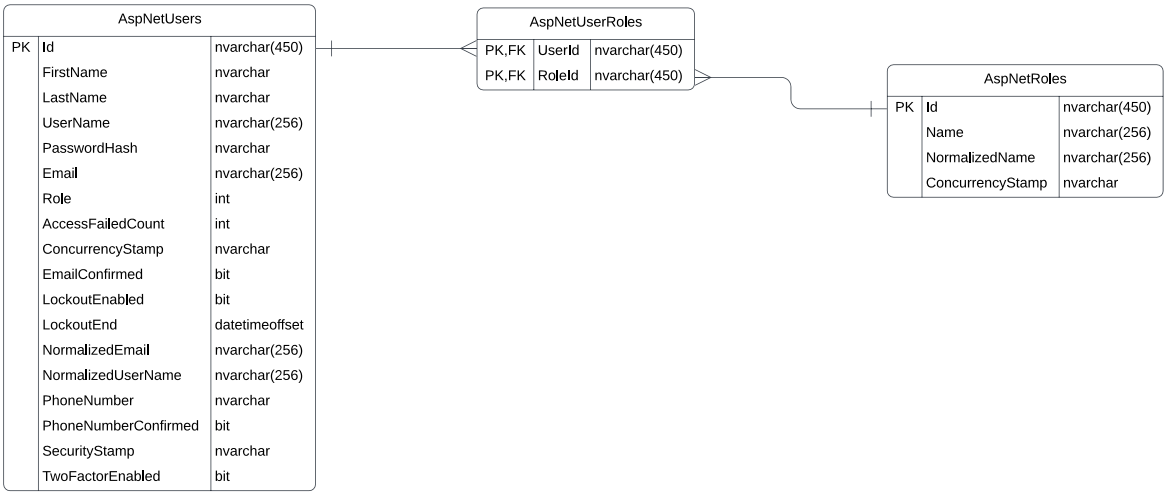


Фигура 10 Таблица във втора нормална форма.

- Посочените таблици от извадката са вече приведени в първа нормална форма, тъй като притежават уникални идентификатори.
- Таблицата преминава във втора нормална форма, когато анализираме полетата и стигнем до заключението, че всички колони на таблицата Groups зависят от първичния ключ.
- Привеждането във втора нормална форма гарантира, че данните са добре структурирани и организирани, като по този начин се избягват ненужни повторения и зависимости. Това улеснява управлението на данните и осигурява по-добра ефективност при извличането на информация от базата данни.

Трета нормална форма:

- Нормализиране на информацията във втората нормална форма. Промяната на запис в една колона да не се отрази на верността на информацията в друга колона.



Фигура 11 Таблица в трета нормална форма.

Като погледнем тези таблици, можем да стигнем до заключението, че те са приведени в трета нормална форма поради следните причини:

- Всички атрибути са атомарни: Всяко поле в таблиците съдържа само един вид информация. Например, в таблицата Users имаме First Name и Last Name, които представляват отделни атрибути за имената на потребителите.
- Всички нетривиални функционални зависимости са разгледани: В таблицата Users няма нелогични или излишни зависимости между атрибутите. Всяко поле зависи само от ключа на таблицата (UserId) и неговите части. Например, Email зависи само от UserId.
- Няма транзитивни функционални зависимости: Няма атрибути, които зависят от други ключови атрибути. В нашия пример, UserRoles таблицата свързва UserId с RoleId, без да добавя други неключови атрибути.

Тези таблици са в трета нормална форма, което гарантира ефективност и лесно управление на данните в базата данни. Те са добре структурирани и организирани, което улеснява извличането и манипулирането на информацията в приложението.

3.4.3 Описание на таблиците от базата данни

- **AspNetUsers:** Тази таблица е предоставена от UserIdentity и съдържа данни за потребителя на системата, който е идентифициран уникално, чрез Id, което се съхранява под формата на низ от символи (nvarchar).

В полето Id се пази идентификатор под формата на Guid и също така служи за първичен ключ на таблицата. Потребителите също притежават FirstName (първо име), LastName (фамилно име), UserName (потребителско име), E-mail , които се пазят под

формата на низ от символи, като е важно да се отбележи, че имейлът трябва да бъде уникален. Достъпът на потребителите се определя от възложената им роля, която се пази в базата данни под формата на число тип int (целочислен тип), като тази стойност е взета от класът за енумериране UserRoles. Също така потребителите притежават парола, която се бива хеширана чрез алгоритъм за хеширане. Паролата също се пази под формата на символен низ.

- **AspNetUserClaims**: Тази таблица също е предоставена от UserIdentity и тя съдържа потребителски претенции(claims) за всеки регистриран потребител.
- **AspNetRoles:** Тази таблица също е предоставена от UserIdentity и съдържа в себе си ролите на потребителите. Ролите са уникални, като тази уникалност се гарантира от идентификатора Id. Всяка роля се характеризира с име, което се пази под формата на символен низ.
- **AspNetUserRoles:** Това е таблица, която служи като преходна таблица (мост), като по този начин се осъществява връзка много към много между таблиците AspNetUsers и AspNetRoles.
- **AspNetRoleClaims:** Тази таблица съдържа претенции (claims) на ролите. Всяка роля притежава набор от претенции, които са уникални за ролята и служат за оторизиране на потребителите, които притежават дадената роля. Всяка претенция се характеризира от тип (ClaimType) и стойност (ClaimValue), които се пазят под формата на символни низове. Типът и стойността се използват за създаване на политики в приложението, които се използват за оторизиране на потребителите.
- **Groups:** Таблицата съдържа информация за групите в системата, като групите се характеризират с име, което се пази под формата на символен низ.
- **GroupUser:** Тази таблица се използва като преходна (мост) и се използва за връзка между таблицата с потребители-AspNetUsers и таблицата с групи-Groups. Връзката е от тип много към много, тъй като много потребители могат да са част от много групи.
- **Tasks:** Чрез тази таблица, в системата се пази информация за задачите, които са част от нея. Всяка задача се идентифицира с уникален идентификатор. Също така задачите се характеризират с име (Name) описание (Description), дата на започване (StartDate), дата на приключване (DueDate), категория (Category), приоритет (Priority), статус (Status) и дата на създаване (CreatedOn). Полетата Name и Description се пазят под формата на символни низове. Полетата StartDate, DueDate и CreatedOn се пазят под

формата на тип за дати-datetime2, който ни позволява да извършваме операции и да работим с дати. Полетата Category, Priority и Status се пазят под формата на число от целочислен тип (int), тъй като тяхната стойност се взима от стойностите на класовете за енумериране TaskCategory, TaskPriority и TaskStatus. Една задача може да бъде възложена на потребител или група, като това се осъществява чрез наличието на външните ключове UserId и GroupId. На един потребител или група могат да бъдат възложени много задачи и поради тази причина типът на връзките между Groups и Tasks и AspNetUsers и Tasks е едно към много.

4 Дизайн

Разработката на проекта е извършена в среда, базирана на .NET 7 платформата, която предлага широк спектър от инструменти и библиотеки за разработка на уеб приложения. Приложението е разделено на три слоя – **слой на услугите, слой за достъп до базата данни и презентационен слой** – уеб приложение изградено чрез ASP .NET Core. Използван е архитектурният шаблон MVC (Model-View-Controller), който разделя приложението на три основни компонента: моделите (Model), изгледите (View) и контролерите (Controller). Този подход улеснява поддръжката и разширяемостта на проекта, като разделя отговорностите между различните компоненти на приложението.

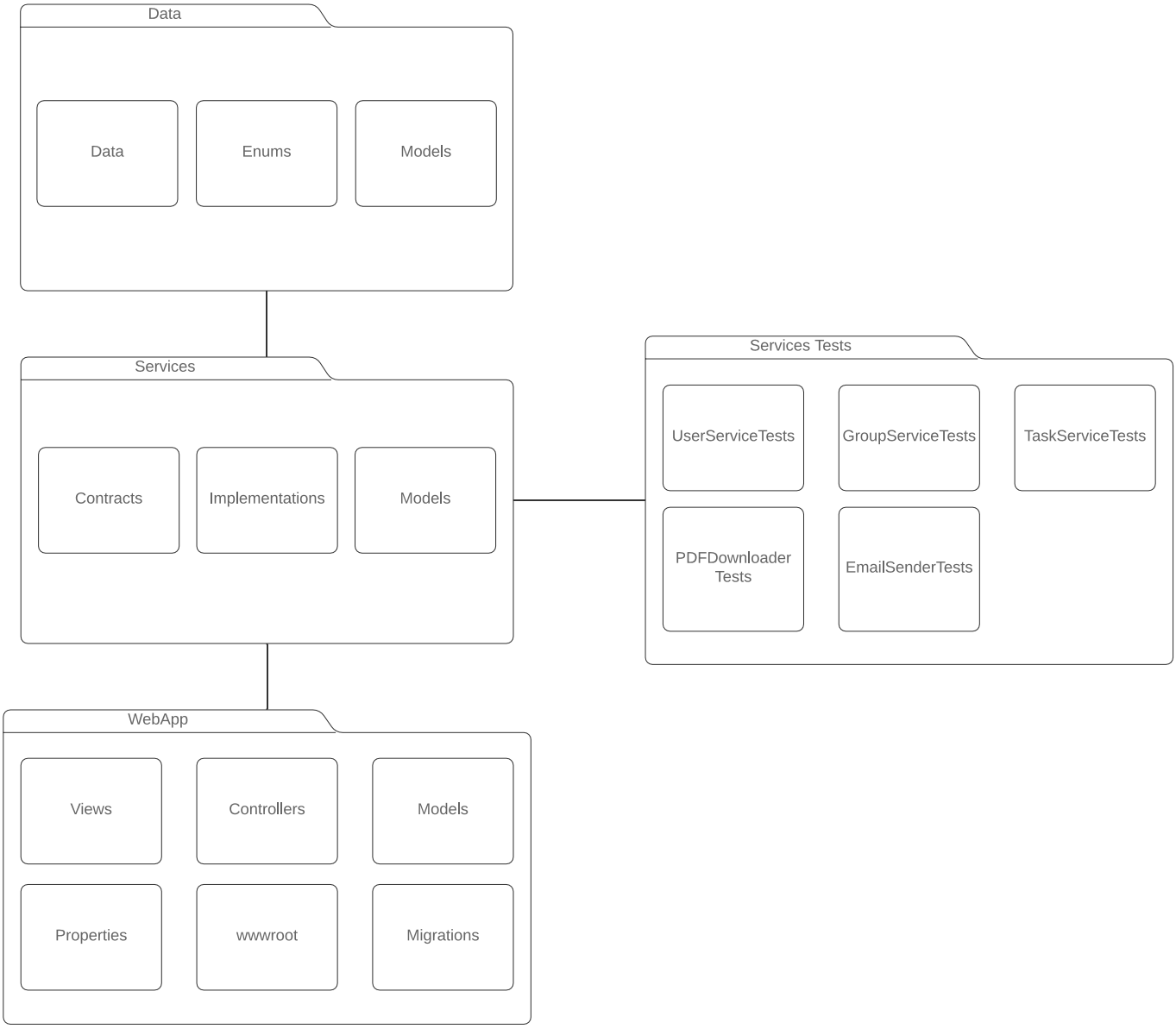
Приложението е разделено на четири проекта, които са отделени в различни проекти – клас библиотеки и уеб приложение. Един проект отговоря за тестването, един за бизнес логиката (услугите), един за достъпа до базите данни и един за уеб приложението.

- **.NET 7** платформата е избрана поради широкия обхват и гъвкавост при разработването на съвременни уеб приложения. Тя предлага широка гама от инструменти и библиотеки, които улесняват разработката, тестването и разпространението на софтуерни приложения.
- **Архитектурният шаблон MVC** е избран за разделяне на приложението на три основни компонента: моделите (Model), изгледите (View) и контролерите (Controller). Този подход улеснява поддръжката и разширяемостта на проекта, като разделя отговорностите между различните компоненти на приложението.
- **Entity Framework Core** е използван за съхранение на данни и работа с база данни. Избран е подходът Code First, който позволява дефиниране на моделите на данните чрез код и автоматично създаване на схемата в базата данни. Това улеснява разработката и поддръжката на базата данни.
- **AutoMapper** се използва за улесняване на преобразуването между моделите на данните и ViewModel-ите в презентационния слой. Този инструмент автоматизира процеса на преобразуване на данните между различните слоеве на приложението, което улеснява разработката и поддръжката на кода.
- **SMTP протоколът** се използва за изпращане на електронни мейли като част от функционалността на системата. Той предоставя възможност за автоматично изпращане на известия и съобщения към потребителите, което подобрява комуникацията и взаимодействието с тях.

- **LINQ (Language-Integrated Query)** се използва за извличане и манипулиране на данни в базата данни. Този инструмент предоставя удобен и ефективен начин за работа с данни, като осигурява декларативен синтаксис за извършване на заявки към данните.
- **Dependency Injection** се прилага за управление на зависимости и инверсия на управлението, което улеснява тестването и поддръжката на кода. Този подход позволява лесно внедряване на зависимости и подменяне на компоненти в приложението, което улеснява разработката и поддръжката на проекта.

Тези технологии и методи се комбинират за създаване на широко приложима, гъвкава и ефективна софтуерна платформа, която отговаря на изискванията на проекта и осигурява висока производителност и надеждност на приложението.

4.1 Реализация на архитектурата на приложението



Фигура 12 Диаграма на цялостната файлова и архитектурна структура на приложението.

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

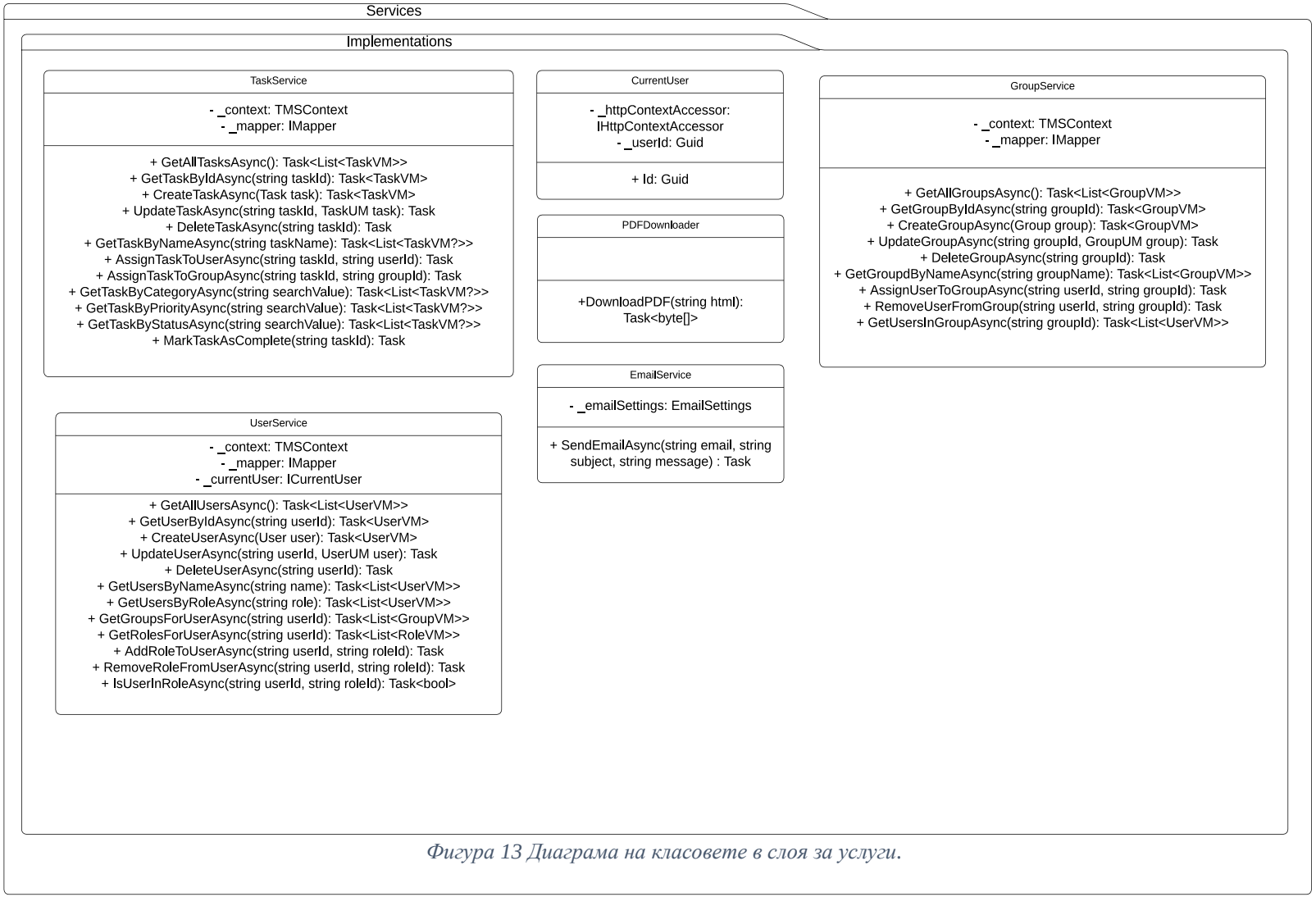
Приложението е разделено на няколко слоя:

4.2.1 Слой на услугите (бизнес логиката)

Слой за услуги, известен още като бизнес логика, представлява компонент в софтуерната архитектура, който съдържа логиката и правилата за обработка на данни и изпълнение на бизнес операции. Този слой отговаря за обработката на заявките от презентационния слой и манипулирането на данните, които влизат и излизат от системата. В него се съдържат функции за валидация, обновяване на данни и други бизнес-специфични операции.

Слоят за услуги помага за разделянето на отговорностите в приложението и осигурява ефективност, поддръжка и повторно използване на кода. Той е реализиран чрез клас библиотека и е разделен на няколко имеви (Namespaces) пространства или папки.

4.2.1.1 Имплементации(Implementations)



Фигура 13 Диаграма на класовете в слоя за услуги.

- **TaskService** (Услуга за Задачи): Класът TaskService представлява клас, отговарящ за управлението на операциите със задачи в системата. Той имплементира интерфейса ITaskService, който дефинира различни операции, свързани със задачите, които могат да се изпълняват чрез този сервиз.

Методите на TaskService позволяват създаване, четене, обновяване и изтриване на задачи, както и търсене на задачи по различни критерии като име, категория, приоритет и статус. Освен това, TaskService предоставя възможност за присвояване на задача на потребител или група, както и маркиране на задачата като завършена.

Класът използва TMSContext за взаимодействие с базата данни, IMapper за преобразуване на моделите на данни към моделите за изглед. Той работи с моделите TaskVM и TaskUM, които представляват съответно моделите за изглед и моделите за обновяване на данните за задачите.

- **UserService** (Услуга за Потребители): UserService е клас, отговарящ за управлението на операциите с потребителите в системата. Той имплементира интерфейса IUserService, който дефинира различни операции, свързани с потребителите, които могат да се изпълняват чрез този сервиз.

Методите на UserService позволяват създаване, четене, обновяване и изтриване на потребители, както и търсене на потребители по различни критерии като име, имейл, потребителско име и роля. Освен това, UserService предоставя възможност за извличане на списък от всички потребители и техните задачи или групи. Класът използва TMSContext за взаимодействие с базата данни, IMapper за преобразуване на моделите на данни към моделите за изглед и ICurrentUser за извличане на информация за текущия потребител. Той работи с моделите UserVM и UserUM, които представляват съответно моделите за изглед и моделите за обновяване на данните за потребителите.

- **CurrentUser** (Текущ Потребител): Класът CurrentUser, разположен в съответната папка, е отговорен за предоставянето на информация за текущия потребител, който използва системата. Този клас имплементира интерфейса ICurrentUser, който дефинира свойство Id, представляващо уникалния идентификатор на потребителя. При създаването на инстанция на класа CurrentUser, той приема IHttpContextAccessor като параметър в конструктора си. Този интерфейс осигурява достъп до текущия HTTP контекст, който се използва за извличане на информация за потребителя, който е вписан в момента. Свойството Id на класа CurrentUser предоставя метод за достъп до уникалния идентификатор на текущия потребител. Той използва

информацията от HTTP контекста, за да извлече уникалния идентификатор на потребителя от него. Ако такъв идентификатор бъде намерен и успешно бъде конвертиран, той се запазва в частно поле за по-късно използване. Ако идентификаторът вече е извлечен и конвертиран, методът просто го връща.

- **PDFDownloader** (Изтегляне на PDF): Класът PDFDownloader, намиращ се в съответната папка, представлява компонент от системата, отговорен за генерирането и изтеглянето на PDF файлове на база на предоставен HTML код. Чрез използването на библиотеката iTextSharp, този клас осигурява възможност за създаване на PDF документи от HTML съдържание.

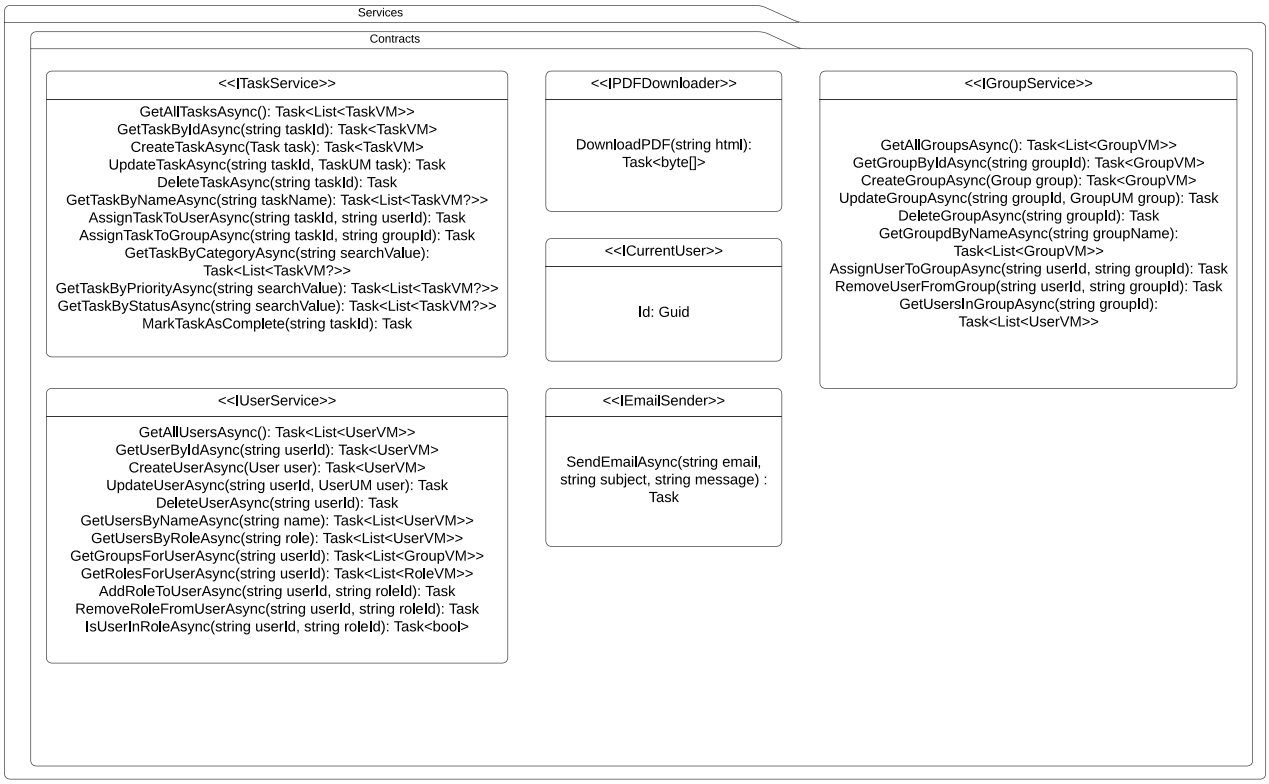
Методът DownloadPDF на класа приема HTML като входен параметър и връща байтов масив, съдържащ PDF представяне на съответното HTML съдържание. При извикване на метода, класът създава нов PDF документ със зададени параметри за размер и отстъпи. След това, извиква HTMLWorker за парсиране на предоставения HTML код и вграждане на съдържанието му в PDF документа. След като процесът завърши, генерираният PDF документ се конвертира в байтов масив и се връща на извикващия код. PDFDownloader предоставя възможност за лесно създаване на PDF файлове от HTML съдържание, което е от полза за различни аспекти на системата, като генериране на доклади, известия или други документи със специфичен формат.

- **EmailService** (Услуга за Имейли): Папката EmailService съдържа логиката за изпращане на имейли през системата, като се използва SMTP сървърът на Brevo и класът System.Net.Mail. Информацията за свързване към сървъра на Brevo е съхранена във файл, който е игнориран от системата за контрол на версиите. Класът, отговарящ за изпращането на имейли, имплементира интерфейса IEmailService и използва модела EmailSettings. EmailSettings моделът съдържа следните полета: SmtpServer (SMTP сървър), Port (порт), Login (потребителско име), SmtpKey (парола за SMTP), и EnableSsl (флаг за активиране на SSL връзка).

- **GroupService** (Услуга за Групи): Класът GroupService е отговорен за управлението на групи в системата. Той имплементира интерфейса IGroupService, който дефинира различни операции, свързани с групите, които могат да бъдат извършвани чрез този сервиз. Методите на GroupService позволяват създаване, четене, обновяване и изтриване на групи, както и добавяне и премахване на потребители от тях. Също така, той предоставя функционалност за извличане на списък от всички групи, както и за намиране на група по нейния уникален идентификатор. Освен това, GroupService

предоставя възможност за извличане на списък от потребители, които са членове на дадена група. Класът използва TMSContext за взаимодействие с базата данни и IMapper за преобразуване на моделите на данни към моделите за изглед. Той работи с моделите GroupVM и GroupUM, които представляват съответно моделите за изглед и за обновяване на данните за групите.

4.2.1.2 Интерфейси(Contracts)



Фигура 14 Диаграма на контрактите в слоя за услуги.

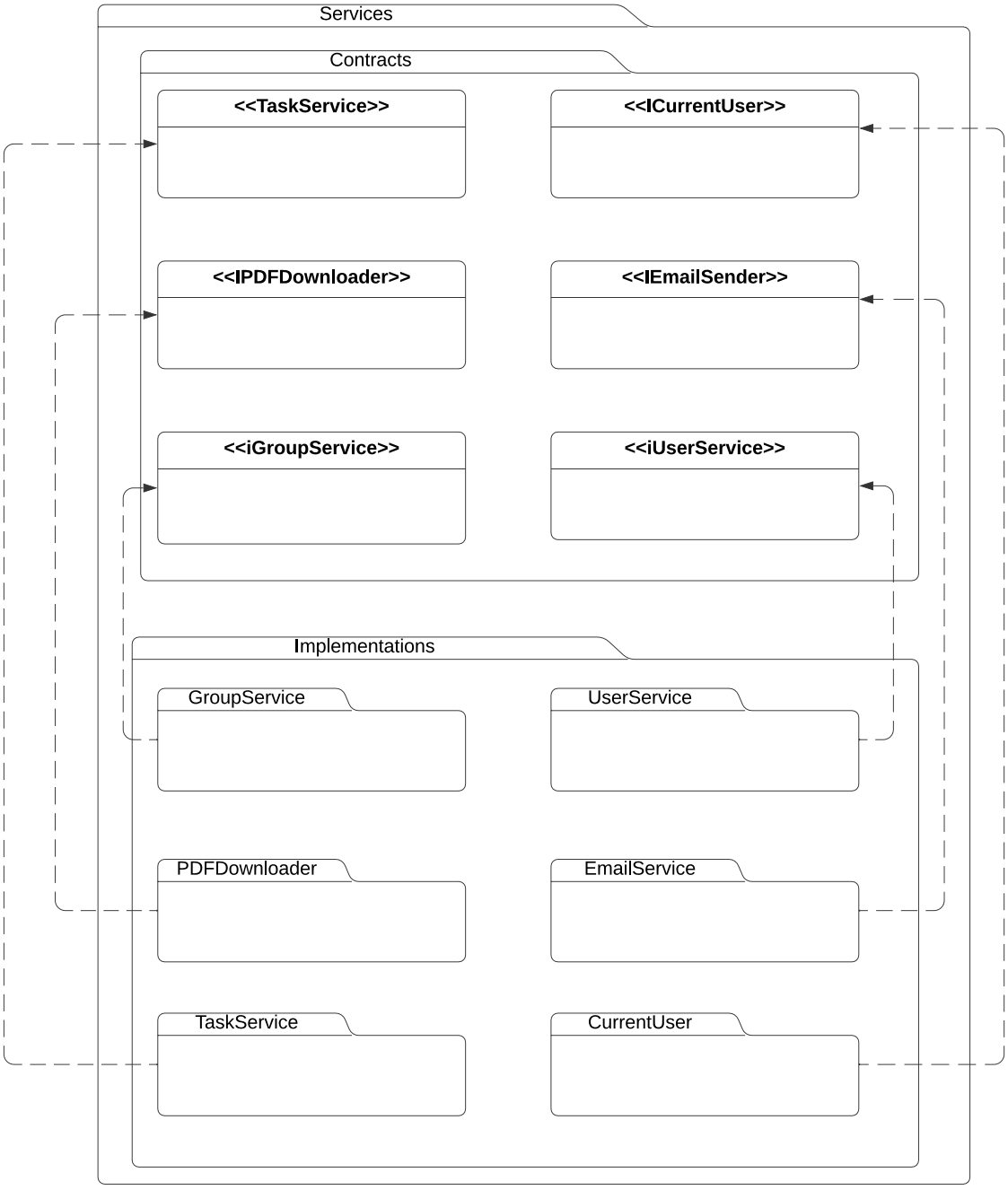
В дадения слой се използват интерфейси за дефиниране на контракти за взаимодействие между различните компоненти на системата. Ето интерфейсите и тяхното предназначение:

- **ICurrentUser:** Този интерфейс дефинира контракт за предоставяне на информация за текущия потребител в системата. В този случай, Id свойството връща уникален идентификатор на потребителя.
- **IEmailSender:** Този интерфейс дефинира контракт за изпращане на имейли в системата. Методът SendEmailAsync приема параметри за имейл адрес, тема и съобщение и изпраща имейл.

- **IGroupService:** Този интерфейс дефинира контракт за управление на групите в системата. Той предоставя методи за създаване, изтриване, обновяване и извличане на информация за групите, както и за управление на членството в тях.
- **IPDFDownloader:** Този интерфейс дефинира контракт за изтегляне на PDF файлове. Методът DownloadPDF приема HTML код и връща байтов масив, съдържащ PDF представяне на HTML кода.
- **ITaskService:** Този интерфейс дефинира контракт за управление на задачите в системата. Той предоставя методи за създаване, изтриване, обновяване и извличане на информация за задачите, както и за търсене на задачи по различни критерии.
- **IUserService:** Този интерфейс дефинира контракт за управление на потребителите в системата. Той предоставя методи за създаване, изтриване, обновяване и извличане на информация за потребителите, както и за търсене на потребители по различни критерии.

Тези интерфейси позволяват лесна замяна на конкретната реализация на услугите без промяна в кода на клиента и подпомагат в разделянето на кода и тестването.

4.2.1.3 Взаимовръзка между контракти и имплементации



Фигура 15 Диаграма на взаимодействие между имплементациите и интерфейсите (контрактите).

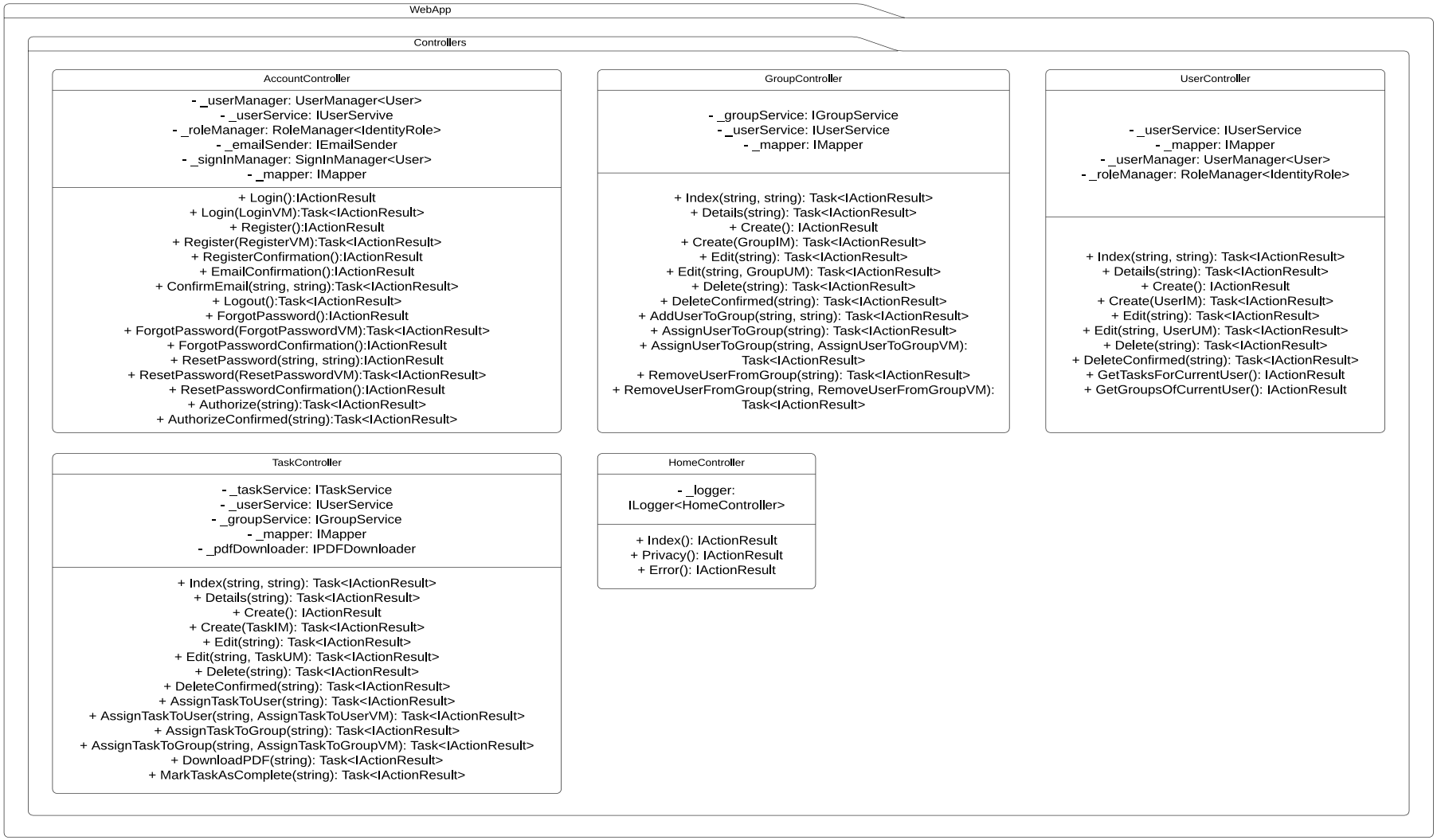
4.2.2 Слой за достъп до базата данни

Слоят за достъп до базата данни в проекта е реализиран чрез клас библиотека. В него се съхраняват моделите, с които работи приложението и класът за TMSContext, който наследява IdentityDbContext и бива използван за работа с рамката Entity Framework.

4.2.3 Уеб приложение

Уеб приложението представлява отделен проект тип ASP.NET Core MVC(Model-View-Controller). В този проект се съдържат изгледите, контролерите и моделите за изглед, които биват използвани от слоя за услуги.

4.2.3.1 Контролери (Controllers)



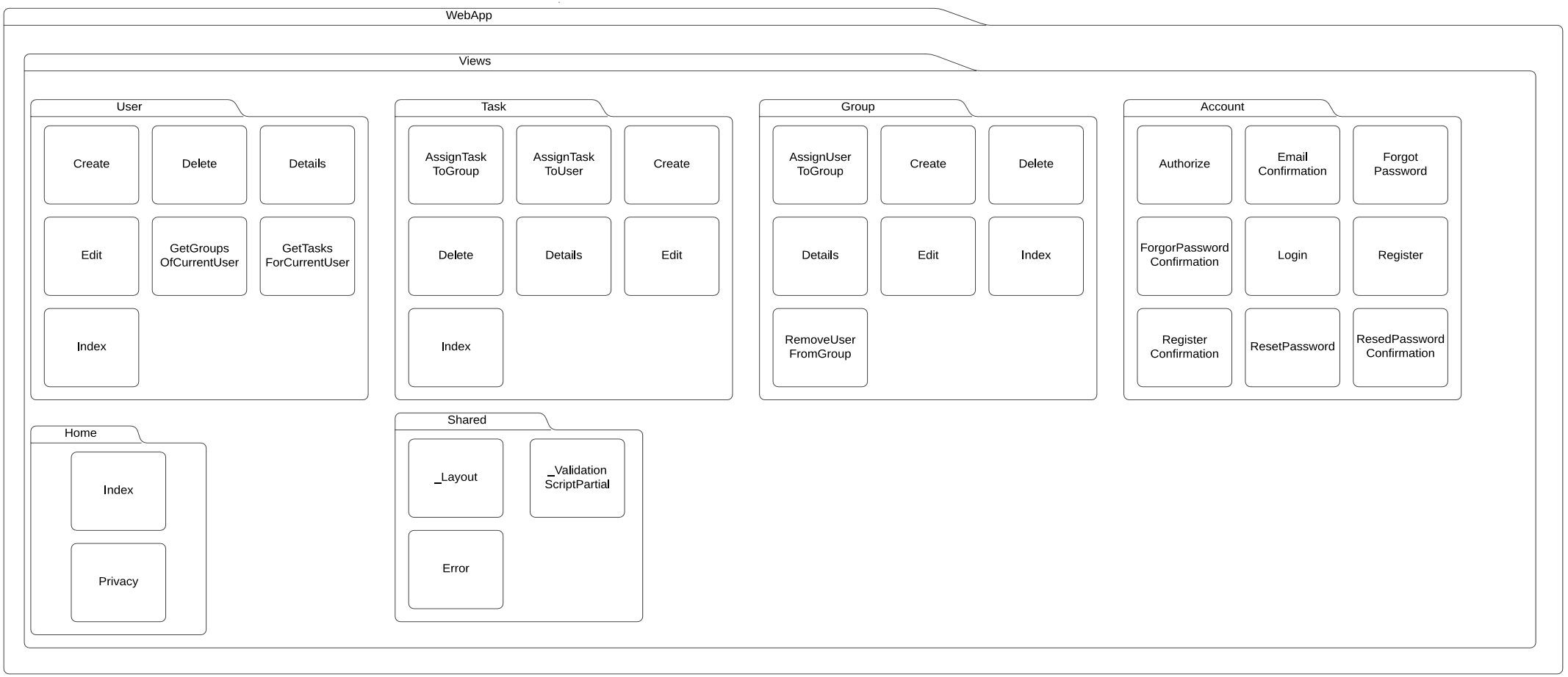
Фигура 16 Диаграма на контролерите.

В приложението със структура на ASP.NET Core MVC имаме няколко контролера, които управляват заявките и връзката между потребителския интерфейс и бизнес логиката на приложението. Да разгледаме поотделно всички контролери и тяхната функционалност: (Tutorialspoint, n.d.).

- **UserController:** Контролерът за потребителите управлява заявките, свързани с потребителите в системата, като създаване, изтриване, актуализиране и извличане на информация за потребителите.
- **TaskController:** Този контролер управлява операциите, свързани с управлението на задачи в системата. Той позволява създаване, редактиране, изтриване и извличане на задачи, както и възлагане на задачи на потребители или групи.
- **AccountController:** Контролерът за акаунти управлява заявките, свързани с акаунтите на потребителите, като вход, изход, потвърждение на имейл, промяна на парола и управление на профила.
- **GroupController:** Този контролер отговаря за операциите, свързани с управлението на групи в системата. Той позволява създаване, редактиране, изтриване и извличане на групи, както и управление на потребителите, които са членове на дадена група.
- **HomeController:** Контролерът за началната страница управлява заявките, свързани с началната страница на приложението, като визуализира основната информация, предоставена на потребителите, и предлага навигационни връзки към други части на приложението.

Контролерите служат като посредници между заявките на потребителите и слоя за услуги на приложението. Те приемат HTTP заявките, извличат необходимата информация от тях и я подават на съответните услуги или класове, които я обработват. Връщат резултатите от обработката на заявката обратно към клиента, като обикновено предават шаблони за генериране на HTML.

4.2.3.2 Изгледи (Views)



Фигура 17 Диаграма на изгледите.

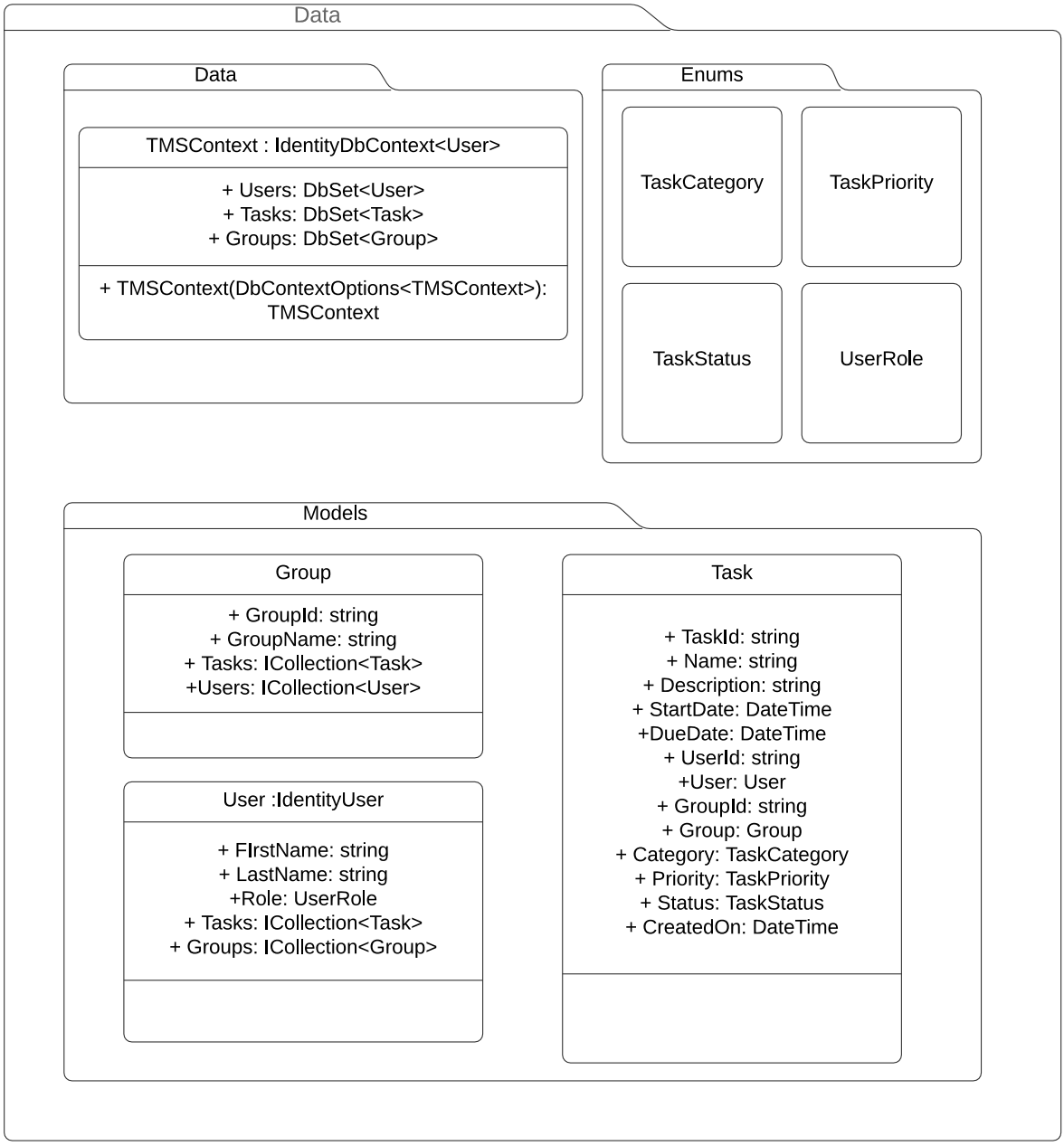
Изгледите в приложението представляват част от потребителския интерфейс, който се визуализира в браузъра на потребителите. Те съдържат HTML, CSS и JavaScript код, който определя визуалното представяне на информацията и интерактивността на уеб страниците. Всеки изглед съответства на един от контролерите в приложението и се използва за представяне на данните, които са получени от съответния контролер. (Tutorialspoint, n.d.).

Ето кратко описание на ролята и функционалността на изгледите в приложението:

- **User Views** (Потребителски изгледи): Тези изгледи се използват за визуализиране на информация, свързана с потребителите на системата, като профилни страници, форми за вход и регистрация, списъци с потребители и други.
- **Task Views** (Изгледи за задачи): Тези изгледи се използват за представяне на задачи и операции, свързани с тях, като списъци със задачи, форми за създаване, редактиране и изтриване на задачи, както и детайлни изгледи за отделни задачи.
- **Account Views** (Изгледи за акаунти): Тези изгледи предоставят интерфейс за управление на акаунтите на потребителите, включително форми за вход и изход, страница за редактиране на профила, потвърждение на имейл и други.
- **Group Views** (Изгледи за групи): Тези изгледи се използват за визуализиране на информация, свързана с групите в системата, като списъци с групи, страница за създаване и управление на групи и други.
- **Home Views** (Начални изгледи): Тези изгледи съдържат началната страница на уеб приложението и представят основната информация, навигационни елементи и връзки към други части на приложението.

Изгледите се използват за представяне на информацията по начин, който е удобен и разбираем за потребителите. Те са ключова част от уеб приложението, като осигуряват интерактивност и визуално привлекателно представяне на данните.

4.3 Организация и код на заявките към база от данни



Фигура 18 Диаграма на слоя за достъп до базите данни.

Слоят за достъп до базата данни (Data Layer) включва класове и енумерации, които управляват връзката и взаимодействието с базата данни. Този слой е отговорен за извличането, обновяването и съхранението на данни в базата данни, като това се случва чрез помощта на Entity Framework.

Entity Framework представлява обектно-релационен картограф, който позволява на .NET разработчиците да работят с релационни бази данни по бърз, лесен и удобен начин. (Microsoft, Entity Framework, 2022).

Ето описание на различните компоненти от този слой:

- **Клас TMSContext:** Класът TMSContext наследява IdentityDbContext<User> и представлява контекста за връзка с базата данни. Той съдържа свойства, които представят отделните таблиците в базата данни, като Users, Tasks и Groups. Този клас също така настройва връзката с базата данни чрез предоставянето на конструктор, който приема параметри от тип DbContextOptions<TMSContext>. (Microsoft, Introduction to Identity on ASP.NET Core, 2023).

- **Енумерации:** В папката Enums се намират различни енумерации, които съответстват на полета с пречислими стойности в моделите на данните. Включени са енумерации като TaskCategory, TaskPriority, TaskStatus и UserRole, които представят различни характеристики на задачи и потребители.

- **Модели на данни:** В папката Models се намират класове, които представят моделите на данни, с които приложението работи. Включени са модели като Task, User и Group, които имат съответстващи свойства за съхранение на данни като идентификационни номера, имена, описание, дати, роли и други. Тези модели са проектирани по начин, който отразява структурата и връзките между различните елементи в базата данни.

Тези компоненти в слоя за достъп до базата данни са организирани по начин, който улеснява управлението на данните и осигурява съвместимост със стандартите на архитектурата на приложението. Те са проектирани за ефективно извличане, обновяване и съхранение на данни, като същевременно осигуряват лесна манипулация и манипулация на данните в приложението.

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

Потребителският интерфейс, създаден с помощта на ASP .NET Core с MVC и използвайки Bootstrap, е съвременен, и лесен за навигация. Ето някои от характеристиките и компонентите на такъв интерфейс:

- **Адаптивен (responsive) дизайн:** Интерфейсът е създаден да бъде респонсивен, което означава, че се адаптира към различни размери на екраните, включително на мобилни устройства, таблети и десктоп компютри.

- **Използване на Bootstrap компоненти:** Bootstrap е широко използвана фронтенд рамка, която предлага готови компоненти за създаване на интерфейси.

- **Модерен дизайн и визуални ефекти:** Интерфейсът има модерен външен вид, като използва визуални ефекти като ховъри, транзиции и др.
- **Лесна навигация и използване:** Има ясна и интуитивна навигация, която прави лесно откриването на функционалността и достъпа до различни части от приложението.
- **Валидация на форми:** Формите за вход или въвеждане на данни използват вградената валидация на Bootstrap за проверка на данните, преди те да бъдат изпратени към сървъра.
- **Поддръжка на различни браузъри:** Интерфейсът е тестван и оптимизиран за работа в различни браузъри като Google Chrome, Mozilla Firefox, Microsoft Edge и др.

5 Ефективност и бързодействие на решението

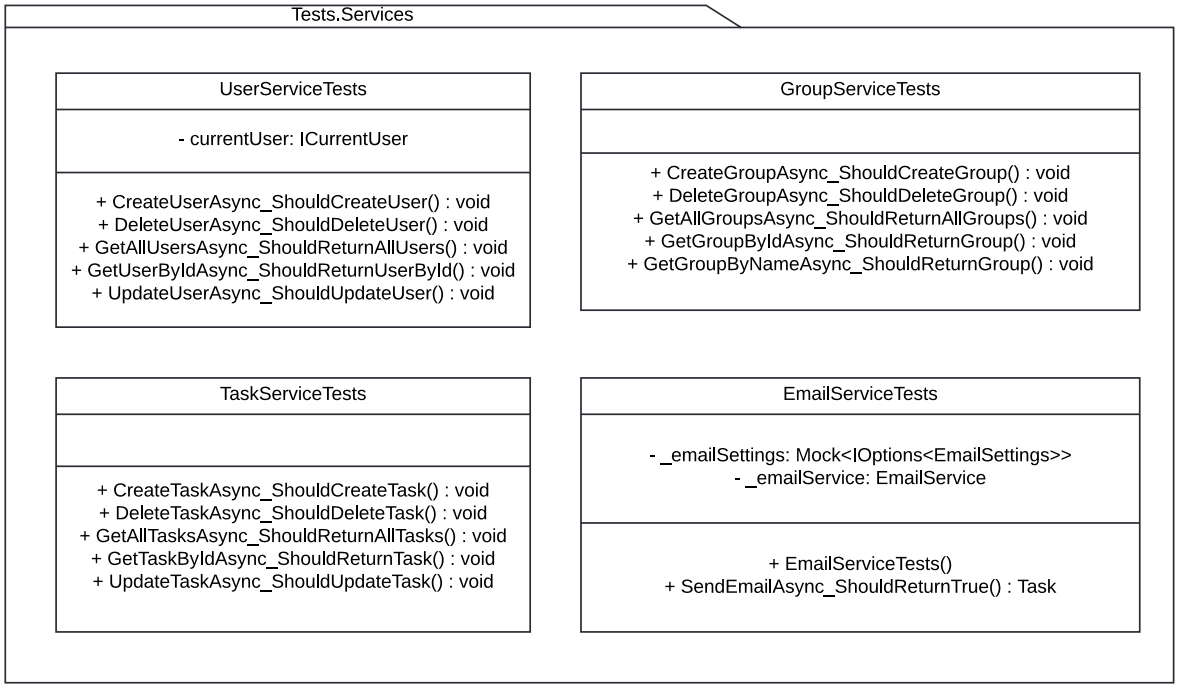
Бързодействието на едно приложение се определя от така наречената времева сложност (Time Complexity). Времевата сложност се отбелязва чрез O нотация, като по този начин се опива най-лошия вариант на времева сложност за един алгоритъм.

Ефективността и бързодействието на разработеното решение се основават на няколко ключови аспекта, които гарантират оптимално функциониране и ускоряване на работните процеси в организацията: (Banimahd, 2023).

- **Оптимизиране на търсенето и достъпа до информация.** Системата предоставя бърз и лесен достъп до процеси и информация, което улеснява работата на персонала и намалява времето, нужно за търсене и навигация в голямото количество данни.
- **Алгоритми за обработка на данни.** Разработените алгоритми са свързани основно с обработката на данните на потребителите. Това се реализира, чрез използване на методите, предоставени от LINQ(Language Integrated Queries) за да обслужи заявките на потребителите. Повечето методи от LINQ са със сложност от O(n) O(1) или с логаритмична сложност(O(n log n)).
  - Примери за методи със сложност O(n) са Remove(), Insert(), RemoveAt(), IndexOf() и Where(), чието време се увеличава линейно, според броя на елементите, с които работят.
  - Пример за метод със сложност O(1) е методът Add(), който просто добавя елемент в края на колекция. Неговата сложност не зависи от броя на елементите в колекцията.
  - Примери за методи със сложност O(n log n) са OrderBy(), GroupBy() и Join(). Тяхното време на изпълнение се увеличава логаритмично, според броя на елементите в колекцията.
- **Гъвкавост.** Приложението е гъвкаво, което означава, че може да се приспособи към нуждите на различните организации и да се разшири при необходимост, без да се губи ефективност.
- **Интуитивен потребителски интерфейс.** Потребителският интерфейс на приложението е интуитивен и семпъл. Това го прави лесен за използване, което намалява времето за обучение на потребителите и увеличава продуктивността.
- **Автоматизация на процесите.** Приложението предлага автоматизирани функционалности за обработка на документи, което намалява необходимостта от ръчно въвеждане на данни и повторение на задачи.

Чрез тези мерки и подходи, разработеното решение гарантира висока ефективност при управлението на работните процеси в организацията, което допринася за повишаване на ефективността и ефикасността по отношение на въведените контролни дейности в организацията. По този начин предложеното решение отговаря в максимална степен на потребностите както на обществените (публичните) организации, така и на бизнеса.

6 Тестване



Фигура 19 Организация на проекта за тестване на код.

В проекта е включен модул за тестване на методите от слоя за бизнес логика, който е реализиран с помощта на тестваща рамка xUnit. Този модул осигурява автоматизирано тестване на различните функционалности и бизнес правила, които са част от приложението. Конвенцията за имената на методите е: ИмеНаМетод_ТрябваДаИзвършиДействие(). (Microsoft, Unit testing C# in .NET using dotnet test and xUnit, 2024).

Тестовите случаи се създават с цел проверка на различните методи и функции от слоя за бизнес логика. В тестовите случаи се представят различни сценарии и входни данни, за да се провери коректността на поведението на системата при различни условия.

Тестовете се организират по възможно най-подходящия начин и структура, като се групират по функционалност и компоненти на приложението. Това помага за лесното навигиране и поддръжка на тестовите случаи.

Тестовете следват метода на трите А-та(Arrange, Act Assert) – подреждане, действане, твърдение и са организирани по следния начин:

- **UserServiceTests**: Отговаря за тестовете на методите за работа с потребители в системата.

- CreateUserAsync_ShouldCreateUser(): Тестов метод, който проверява дали методът CreateUserAsync от UserService успешно създава нов потребител в системата.
- DeleteUserAsync_ShouldDeleteUser(): Тестов метод, който проверява дали методът DeleteUserAsync от UserService успешно изтрива съществуващ потребител от системата.
- GetAllUsersAsync_ShouldReturnAllUsers(): Тестов метод, който проверява дали методът GetAllUsersAsync от UserService успешно връща списък от всички потребители в системата.
- GetUserByIdAsync_ShouldReturnUserById(): Тестов метод, който проверява дали методът GetUserByIdAsync от UserService успешно връща потребител по зададен идентификационен номер.
- UpdateUserAsync_ShouldUpdateUser(): Тестов метод, който проверява дали методът UpdateUserAsync от UserService успешно актуализира съществуващ потребител в системата.

- **GroupServiceTests**: Отговаря за тестовете на методите за работа с групи в системата:
  - CreateGroupAsync_ShouldCreateGroup(): Тестов метод, който проверява дали методът CreateGroupAsync от GroupService успешно създава нова група в системата.
  - DeleteGroupAsync_ShouldDeleteGroup(): Тестов метод, който проверява дали методът DeleteGroupAsync от GroupService успешно изтрива съществуваща група от системата.
  - GetAllGroupsAsync_ShouldReturnAllGroups(): Тестов метод, който проверява дали методът GetAllGroupsAsync от GroupService успешно връща списък от всички групи в системата.
  - GetGroupByIdAsync_ShouldReturnGroup(): Тестов метод, който проверява дали методът GetGroupByIdAsync от GroupService успешно връща група по зададено идентификационно номер.
  - GetGroupByNameAsync_ShouldReturnGroup(): Тестов метод, който проверява дали методът GetGroupByNameAsync от GroupService успешно връща група по зададено име.

- **TaskServiceTests**: Отговаря за тестовете на методите за работа със задачи в системата:
  - CreateTaskAsync_ShouldCreateTask(): Тестов метод, който проверява дали методът CreateTaskAsync от TaskService успешно създава нова задача в системата.
  - DeleteTaskAsync_ShouldDeleteTask(): Тестов метод, който проверява дали методът DeleteTaskAsync от TaskService успешно изтрива съществуваща задача от системата.
  - GetAllTasksAsync_ShouldReturnAllTasks():Тестов метод, който проверява дали методът GetAllTasksAsync от TaskService успешно връща списък от всички задачи в системата.
  - GetTaskByIdAsync_ShouldReturnTask(): Тестов метод, който проверява дали методът GetTaskByIdAsync от TaskService успешно връща задача по зададено идентификационно номер.
  - UpdateTaskAsync_ShouldUpdateTask(): Тестов метод, който проверява дали методът UpdateTaskAsync от TaskService успешно актуализира съществуваща задача в системата.
- **EmailServiceTests**: Отговаря за тестване на услугата за изпращане на имейл:
  - SendEmailAsync_ShouldReturnTrue(): Тестов метод, който проверява дали методът SendEmailAsync от EmailService успешно изпраща имейл и връща стойност true за успешно изпратен имейл.

7 Заключение и възможно бъдещо развитие

В заключение може да се обобщи, че проектът изпълнява всички поставени в заданието цели. Разработената система за управление и контрол на работните процеси представлява важен инструмент за подобряване на организацията на работа, нивото на контрол и съответно ефективността и продуктивността на организациите. Архитектурата на приложението е подпомогната от добрите практики, спазени при създаването на моделите, изгледите и всички класове и интерфейси.

Предимствата на използваните технологии, като например UML за моделиране на системата и съответните програмни езици и инструменти за имплементация, включват:

- **Стандартизация и яснота:** Използването на UML осигурява стандартизиран начин за визуализация на системните архитектури, което улеснява разбирането и комуникацията между разработчиците и заинтересованите страни.
- **Гъвкавост и разширяемост:** Програмните езици и инструменти, използвани за разработката на системата, позволяват гъвкаво и лесно разширяване на функционалността и адаптиране към специфичните нужди на организацията.
- **Бърза разработка и имплементация:** Използването на съвременни технологии и методи позволява бърза разработка и имплементация на системата, което е от решаващо значение в днешния бързо променящ се бизнес свят.

Недостатъците на използваните технологии включват възможни ограничения във функционалността или производителността при мащабиране на системата до по-големи обеми или при срещане на сложни сценарии.

Като алтернатива, е възможно да се използват други методологии за разработка на софтуер, като Agile или Lean, които се фокусират върху гъвкавостта и непрекъснатата доставка на функционалност. Също така, могат да се разгледат и алтернативни програмни езици и инструменти, които са насочени към специфични изисквания и нужди на проекта.

В практиката, подобни решения се използват широко в различни отрасли и сфери на бизнеса за оптимизиране на работните процеси и подобряване на оперативната работа. За бъдещото развитие на разработената система, бих препоръчал изследване и внедряване на нови технологии и методи, които да подпомагат внедряването на изкуствен интелект, машинно самообучение и автоматизация на рутинните задачи.

Също така, увеличаването на сигурността и защитата на данните са от съществено значение за бъдещото развитие на системата, особено във връзка със защитата на личните данни и съответствието с нормативните изисквания.

8 Използвани литературни източници и Уеб сайтове

1. Banimahd, M. (25 January 2023 г.). *Understanding Time Complexity of C# List and LINQ Methods*. Извлечено от Medium: https://masoudx.medium.com/understanding-time-complexity-of-c-list-and-linq-methods-6b5e36d66243

2. Microsoft. (22 July 2022 г.). *Entity Framework*. Извлечено от Microsoft: https://learn.microsoft.com/en-us/aspnet/entity-framework

3. Microsoft. (14 7 2023 г.). *Description of the database normalization basics*. Извлечено от Microsoft: https://learn.microsoft.com/en-us/office/troubleshoot/access/database-normalization-description

4. Microsoft. (25 August 2023 г.). *Introduction to Identity on ASP.NET Core*. Извлечено от Microsoft: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-8.0&tabs=visual-studio

5. Microsoft. (7 3 2024 г.). *Unit testing C# in .NET using dotnet test and xUnit*. Извлечено от Microsoft: https://learn.microsoft.com/en-us/dotnet/core/testing/unit-testing-with-dotnet-test

6. Oracle. (н.д.). *What is a Database*. Извлечено от Oracle: https://www.oracle.com/database/what-is-database/

7. Tutorialspoint. (н.д.). *MVC Framework - Introduction*. Извлечено от Tutorialspoint: https://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm

9 Приложения

10 Критерии и показатели за оценяване

Критерии и показатели за оценяване	Максимален брой точки за показателите	Максимален брой точки за критерия
1. Съответствие с изискванията за съдържание и структура на дипломния проект		20
1. 1. логическа последователност и структура на изложението, балансиране на отделните части	4	
1.2. задълбоченост и пълнота при формулиране на обекта, предмета, целта и задачите в разработването на темата	7	
1.3. използване на подходящи изследователски методи	4	
1.4. стил и оформяне на дипломната работа (терминология, стил на писане, текстообработка и оформяне на фигури и таблици)	5	
2. Съответствие между поставените цели на дипломния проект и получените резултати		20
2.1. изводите следват пряко от изложението, формулирани са ясно, решават поставените в началото на изследването цели и задачи и водят до убедителна защита на поставената теза	10	
2.2. оригиналност, значимост и актуалност на темата	6	
2.3. задълбоченост и обоснованост на предложенията и насоките	4	
3. Представяне на дипломния проект		20
3.1. представянето на разработката по темата е ясно и точно	5	
3.2. онагледяване на експозето с: а) презентация; б) графични материали; в) практически резултати; г) компютърна мултимедийна симулация и анимация	10	

3.3. умения за презентиране	5	
4. Отговори на зададените въпроси от рецензента и/или членовете на комисията за защита на дипломен проект		30
4.1. разбира същността на зададените въпроси и отговаря пълно, точно и убедително	10	
4.2. логически построени и точни отговори на зададените въпроси	10	
4.3. съдържателни и обосновани отговори на въпросите	10	
5. Използване на професионалната терминология, добър и ясен стил, обща езикова грамотност		10
5.1. Правилно използване на професионалната терминология	5	
5.2. Ясен изказ и обща езикова грамотност	5	
Общ брой точки:	Максимален бр. точки 100	Максимален бр. точки 100