



ДИПЛОМНА РАБОТА

ЗА ПРИДОБИВАНЕ НА ТРЕТА СТЕПЕН НА
ПРОФЕСИОНАЛНА КВАЛИФИКАЦИЯ

**Тема: Изграждане на комплекс от методи, средства и
подходи при разработването на приложение:
„Система за данъчни услуги“**

Ученик:
Иво Недев

Ръководител консултант:
Милен Василев

Професия: „Приложен програмист“
Специалност: „Приложно програмиране“

Велико Търново 2024г.

Съдържание

Увод.....	2
Теоретична част.....	4
Използвани технологии.....	4
Microsoft Visual Studio 2022.....	4
MVC (Model-View-Controller).....	5
HTML.....	6
CSS.....	7
JavaScript.....	8
C#.....	9
.NET.....	9
Microsoft SQL Server.....	11
SQL Server Management Studio.....	12
Github.....	13
Bootstrap.....	14
Razor.....	15
HtmlSanitizer.....	15
Практическа част.....	16
Конструкция на проекта.....	16
Startup.cs.....	16
ApplicationBuilderExtensions.cs.....	18
ApplicationDbContext.cs.....	19
Миграции.....	21
Изгледи (Views).....	22
Контролери.....	26
Сървиси (Services).....	28
Алгоритми.....	30
Заклучение.....	32
Използвана литература.....	32
Приложение.....	33
Използване на проекта.....	33

Увод

Дипломната работа представя ревизия на уеб-базираното приложение „Система за данъчни услуги“. Като показва целта на проекта, използваните технологии при разработката му и подходите, чрез които е създадено.

Целта на проекта е да улеснява клиентите и служителите в дадена община, като разпределя клиенти по всички гишета, които предлагат желаната от тях услуга. Така се предотвратяват опашките и напреженията в обществената сградата. Всеки клиент може да направи заявка за всяка предоставяна услуга и проекта автоматично ще му изчисли на кое гише и в колко часа да бъде. Не е нужна външна намеса за функционалността на проекта като обработката на данни може да бъде извършена изцяло от уеб интерфейса. Така потребителите могат да създават, редактират и изтриват информация в базата данни.

Подходите, използвани при създаването на уеб-базираното приложение "Система за данъчни услуги", включват:

- **Обектно ориентирано програмиране (ООП):** Използването на C# позволява изграждането на приложение, което се базира на принципите на ООП. Това включва използването на класове, обекти, наследяване, полиморфизъм и инкапсулация за създаване на модулна и лесно разширяема архитектура.
- **Интегрирана среда за разработка (IDE):** Visual Studio е избраната среда за разработка за създаването на приложението. Тя предоставя широк спектър от инструменти за разработка, включително удобен интерфейс за програмиране, интегриран отстраняване на грешки и автоматично довършване на кода.
- **Уеб-базирана архитектура:** Приложението е базирано на уеб технологии като HTML, CSS, JavaScript и Razor, което го прави достъпно от различни устройства с достъп до интернет. Използването на уеб интерфейс позволява на потребителите да използват приложението без необходимостта от инсталиране на допълнителен софтуер. HTML се използва за създаване на структурата на уеб страниците. С помощта му се дефинират различни елементи и техните взаимоотношения, което определя как информацията се представя на потребителите. CSS е приложен за стилизиране на уеб страниците и придаване на техния външен вид. Чрез CSS се определят различни аспекти на дизайна, като цветове, шрифтове, размери, разположение на елементите и други, което допринася за усещането за естетика и удобство за потребителите. С JavaScript е създадена една от валидацията на входните данни в проекта. Razor е технология, която позволява съчетаването на код на C# с HTML в едно уеб приложение, което е особено полезно при създаването на динамично генерирани уеб страници. Тази технология се използва в ASP.NET, което позволява лесното интегриране на бизнес логиката на приложението с уеб интерфейса.
- **Автоматизирано разпределение на ресурсите:** Чрез изчисления и алгоритми, вградени в приложението, се осигурява автоматично разпределение на клиентите към различните гишета в зависимост от тяхната заявка и текущото натоварване. Това помага за оптимизиране на обслужването и намаляване на опашките.

- Сигурност и защита на данните: Приложението включва мерки за защита на данните, като например криптиране на комуникацията чрез HTTPS и мерки за управление на достъпа до данните. Също така е използван и HTML Sanitizer, който е инструмент с голямо значение за осигуряване на сигурността на уеб приложението, като предотвратява възможни атаки като XSS (Cross-Site Scripting). HTML Sanitizer се използва за филтриране и премахване на потенциално опасен HTML и JavaScript код от входните данни, преди те да бъдат обработени и показани на потребителите. Това помага за предотвратяване на злонамерени атаки и защитава уеб приложението и данните на потребителите от външни заплахи.
- SQL Management Studio. Това софтуерно приложение предоставя мощни инструменти за управление на бази данни, включително създаване на таблици, извършване на заявки и оптимизиране на базата данни за ефективно съхранение и достъп до информацията. Използването му допринася за създаването на стабилна и надеждна база данни, която е основен компонент от функционалността и ефективността на приложението за данъчни услуги.
- GitHub предоставя редица полезни функции и възможности за управление на кода и сътрудничество в софтуерните проекти. Някои от ключовите функции включват:
 - Хостинг на хранилища за код: GitHub позволява на потребителите да създават хранилища за своите проекти, където могат да съхраняват, управляват и споделят своя код.
 - Версионизиране на кода: С GitHub може да се правят версии на кода, което позволява на разработчиците да следят промените, да сътрудничат и да се връщат към предишни версии на своите проекти.
 - Управление на задачи и проблеми: GitHub предоставя инструменти за управление на задачи и проблеми, което улеснява сътрудничеството и проследяването на проблемите в различните версии на софтуерните проекти.

Тези подходи и технологии създават функционално и удобно за използване уеб приложение за управление на данъчни услуги, което подобрява ефективността и удобството както за клиентите, така и за служителите на общинската администрация.

Теоретична част

Използвани технологии

Microsoft Visual Studio 2022

Microsoft Visual Studio е мощна интегрирана среда за разработка (IDE), специално създадена за създаването на софтуерни приложения за Windows и платформата .NET Framework. Тя представлява универсален инструментариум, който позволява разработката както на конзолни, така и на графични потребителски интерфейси (GUI) приложения. Обхватът ѝ включва както традиционните Windows Forms, така и по-модерната Windows Presentation Foundation (WPF). Освен това, Visual Studio разширява възможностите си, за да обхване и уеб ориентираните проекти, включително уеб сайтове, уеб приложения и уеб услуги, работещи на различни платформи на Microsoft - Windows, Windows Mobile, Windows CE, .NET Framework, .NET Compact Framework и Microsoft Silverlight.

В рамките на мощния си *каркас*, Visual Studio предлага богата гама от функции и възможности, които дават сила на разработчиците през целия цикъл на разработката на софтуера. От началото на създаването на кода до неговото изпълнение, дебъгване и крайното разпространение, Visual Studio оркестрира всяка стъпка с изящество. Нейните възможности са очевидни и във фината настройка и дебъгване както на високо, така и на ниско ниво, гарантирайки задълбочен анализ и доработване на приложенията.

В сърцето на функционалностите на Visual Studio лежи нейната мощ в разработка на код, компилиране, изпълнение и дебъгване. Въпреки това, нейният обхват отива далеч извън просто манипулиране на кода. IDE-то се гордее с множество инструменти и помощни програми за тестване на приложения, дизайн на потребителски интерфейси, включващи форми, диалози, уеб страници и визуални контроли, моделиране на данни и класове, изпълнение на тестове, пакетиране на приложения и много други функции.

Освен това, Visual Studio е изключително разширяема платформа, позволяваща интеграцията на плъгини, които разширяват нейните възможности, за да отговарят на разнообразни нужди в разработката. Без значение дали става въпрос за подобряване на поддръжката на системи за управление на изходния код като Subversion и Visual SourceSafe или въвеждане на специализирани инструменти като редактори и визуални дизайнери за езици със специфични домейни, Visual Studio позволява персонализация на почти всеки аспект от процеса на разработка. Допълнително, то се интегрира безупречно с колаборативните инструменти на Microsoft, като Team Foundation Server и Team Explorer, създавайки условия за по-ефективно сътрудничество и управление на проекти.

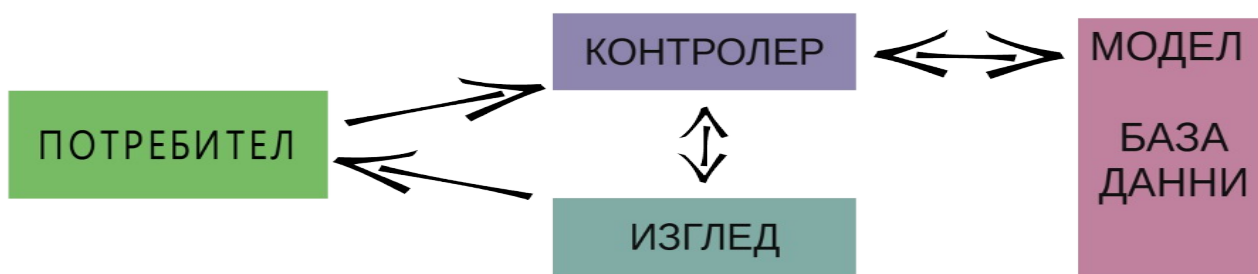


MVC (Model-View-Controller)

Архитектурният шаблон MVC е проектиран с цел да раздели функционалността на софтуерните приложения на три основни компонента, което улеснява разработката, поддръжката и тестването на кода. Ето по-подробно какво представляват всяка от частите на MVC:

- Модел (Model)
 - Моделът представлява сърцевината на приложението и включва бизнес логиката и данните, които приложението обработва и управлява.
 - Това е мястото, където се извършват операции като създаване, четене, обновяване и изтриване на данни (CRUD операции).
 - В модела се съхраняват данни като потребителски профили, продукти, резервации и други, както и бизнес правила и валидации.
 - Моделът е независим от потребителския интерфейс и контролера, което го прави лесен за преизползване и тестване.
- Изглед (View)
 - Изгледът е отговорен за представянето на данните от модела на потребителя в подходящ и разбираем формат.
 - Това може да бъде визуалното представяне на данните, като HTML шаблон, XML документ, PDF файл и други формати за изображение или текст.
 - Изгледът не съдържа логика за обработка на данни, а само представя информацията, която му е предоставена от модела.
 - Предимството на изгледа е, че може да бъде гъвкаво променян или заменян без да се наруши бизнес логиката в модела или контролера.
- Контролер (Controller)
 - Контролерът е посредник между модела и изгледа и се използва за обработка на заявките от потребителите.
 - Той приема заявките, извлича необходимата информация от модела и избира подходящия изглед за представяне на тази информация.
 - Контролерът съдържа логиката за обработка на заявките, включително валидацията на входните данни и пътят на заявките към съответните ресурси.

Архитектурният шаблон MVC осигурява ясно разграничение между различните компоненти на приложението, което води до по-добра поддръжка, разширяемост и преизползваемост на кода. Той се използва широко в уеб разработката, като предоставя гъвкав и ефективен начин за създаване на уеб приложения.



HTML

HTML (съкращение от английски: HyperText Markup Language, в превод „език за маркиране на хипертекст“) е основният маркиращ език за описание и дизайн на уеб страници. HTML е стандарт в интернет, а неговите стандарти се определят от международния консорциум W3C. Текущата версия на стандарта е HTML 5.0 (от 28 октомври 2014г.), а предходната стабилна версия е HTML 4.1.

Описанието на документа става чрез специални елементи, наречени HTML елементи или техните маркери, които се състоят от тагове и съответстващите етикети (HTML tags) и ъглови скоби (като например елемента `<html>`). HTML елементите са основната градивна единица на кода, който изграждат уеб страниците. Чрез тях се форматира, графично оформя текста и неговите отделните части в рамките на една уеб страница, като например заглавия, цитати, текстови раздели, хипертекстови препратки. Най-често HTML елементите са групирани по двойки `<h1>` и `</h1>`.

В повечето случаи HTML кодът е написан в текстови редактори, с файлов формат .html и се качва и хоства на сървъри, които са онлайн в интернет или са част от WWW мрежата. Тези .html файлове съдържат програмно на таговете на HTML и текстово съдържание със маркери и коментари – също инструкции за брауъра, за това какъв точно тип е .html страницата, а също за това как да се показва текстът, особено що се отнася до езиковите характеристики. За да се илюстрира как се включва текст в HTML код: `<маркер>` Някакъв текст. `</край на маркера>`. уеб брауърите са програмирани от своя страна така, в повечето случаи, макар че някои брауъри могат да имат съответно проблеми на версията, за да могат да прочетат HTML документите и да ги покажат на екрана като уеб страници. Брауърите не показват самите HTML тагове, освен ако не се отиде в менюто за да се направи това, така че те „интерпретират“ съдържанието на страницата като код и текст за да могат след работа на процесора да покажат желаното уеб съдържание.

Основното предимство на HTML е, че уеб страниците, които са го включват в кода си, могат да се разглеждат чрез показването им от брауъра на екрана на повечето устройства. Уеб страницата може да има дизайн, който дори изглежда с добър дизайн с помощта на CSS или „правилно оформен“, както върху монитора на персоналния компютър, но също и върху миниатюрния дисплей на пейджър или дисплея на мобилен телефон.

HTML може да прикрепя скриптове писани на езици като JavaScript, който е помощен за HTML, и това променя поведението на дадена уеб страница. Cascading Style Sheets (CSS) се използват, като това се прави за да се определя изгледа и оформлението на текста и други включени в страницата изображения и илюстриращи материали. World Wide Web Consortium

(W3C) поддържа както HTML, така и CSS, и насърчава използването на CSS в HTML страниците още от 1997. Това допринася за разделяне съдържанието и структурата на уеб страниците от тяхното визуално представяне.



CSS

CSS (Cascading Style Sheets) е език за стилове, използван за задаване на представянето и стилизирането на документ, написан на маркиращ език като HTML или XML (включително XML диалекти като SVG, MathML или XHTML). CSS е основна технология на Световната мрежа, заедно с HTML и JavaScript.

CSS е проектиран да позволява разделянето на съдържанието и представянето, включително изграждането на макет, цветовете и шрифтовете. Това разделяне може да подобри достъпността на съдържанието, да осигури по-голяма гъвкавост и контрол при специфицирането на характеристики за представяне, да позволи на множество уеб страници да споделят форматиране, като се специфицира съответният CSS в отделен .css файл, което намалява сложността и повторението в структурното съдържание; и да позволи на .css файла да бъде кеширан за подобряване на скоростта на зареждане на страниците, които споделят файла и неговото форматиране.

Разделянето на форматирането и съдържанието също прави възможно представянето на същата маркировка на страница в различни стилове за различни методи на визуализация, като на екран или на печат. CSS също има правила за алтернативно форматиране, ако съдържанието да бъде достъпно от мобилно устройство.

Името "cascading" произлиза от определената приоритетна схема, за да се определи коя декларация се прилага, ако повече от една декларация на дадено свойство отговарят на определен елемент правейки тази каскадна приоритетна схема е предсказуема.



JavaScript

JavaScript, често съкращаван като JS, е програмен език и основна технология на уеба, заедно с HTML и CSS. Приблизително 99% от уеб сайтовете използват JavaScript на клиентската страна за управление на поведението на уеб страниците.

Уеб браузърите разполагат със специализирани двигатели за изпълнение на JavaScript кода. Тези двигатели се използват и в различни сървърни среди и приложения. Един от най-популярните среди за изпълнение на JavaScript извън браузъра е Node.js.

JavaScript е език от високо ниво, който често се компилира в реално време и се придържа към стандарта ECMAScript. Той разполага с динамично типизиране, прототипно-ориентирано обектно-ориентиране и функции от първи клас. JavaScript е език, който поддържа различни стилове на програмиране, включително събитие-ориентиран, функционален и императивен.

JavaScript разполага с широк набор от приложни програмни интерфейси (API), които позволяват работа с текст, дати, регулярни изрази, стандартни структури от данни и модела на обектите на документа (DOM). Той също така предоставя възможности за манипулиране на графични елементи и интеракция с потребителите.

Въпреки че името на JavaScript е подобно на Java, двата езика са съвсем различни и различават се значително по дизайн и функционалност. Въпреки това JavaScript е един от най-широко използваните програмни езици в света, който играе ключова роля в уеб разработката и интерактивността на уеб страниците.

JavaScript и DOM предоставят потенциала за злонамерени автори да доставят скриптове, които да се изпълняват на клиентски компютър чрез уеба. Авторите на браузъри намаляват този риск, като използват две ограничения. Първо, скриптовете се изпълняват в пясъчник, в който те могат да извършват само действия, свързани с уеба, а не общи задачи като създаване на файлове. Второ, скриптовете са ограничени от политиката на един и същ източник: скриптове от един уеб сайт нямат достъп до информация като потребителски имена, пароли или бисквитки, изпратени към друг сайт. Най-честите проблеми със сигурността, свързани с JavaScript, са нарушения на политиката за същия източник или на пясъчника.

Има подмножества на общия JavaScript - AD safe, Secure ECMAScript (SES) - които предоставят по-високи нива на сигурност, особено за код, създаден от трети страни (като реклами). Closure Toolkit е друг проект за безопасно вграждане и изолация на JavaScript и HTML от трети страни.

JavaScript



C#

C# (C Sharp) е обектно ориентиран език за програмиране, разработен от Microsoft като ключов компонент на софтуерната платформа .NET. При създаването на C# основната цел беше да се създаде прост, модерен и мощен език, който да отговаря на нуждите на съвременната софтуерна индустрия.

Основата за разработката на C# включва влиянието на езици като C++, Java, Delphi, VB.NET и C. Той е проектиран с цел да балансира мощността на C++ с възможността за бързо и лесно разработване, като по този начин комбинира висока производителност с удобство за програмиране.

В програмите на C# се използват класове, които съдържат методи и програмна логика. Файловете с разширение .cs съдържат дефинициите на тези класове и други типове данни. След компилация от компилатора на C#, тези файлове се превръщат в изпълним код, като се създават асемблита – файлове със същото име, но с различни разширения (.exe или .dll).

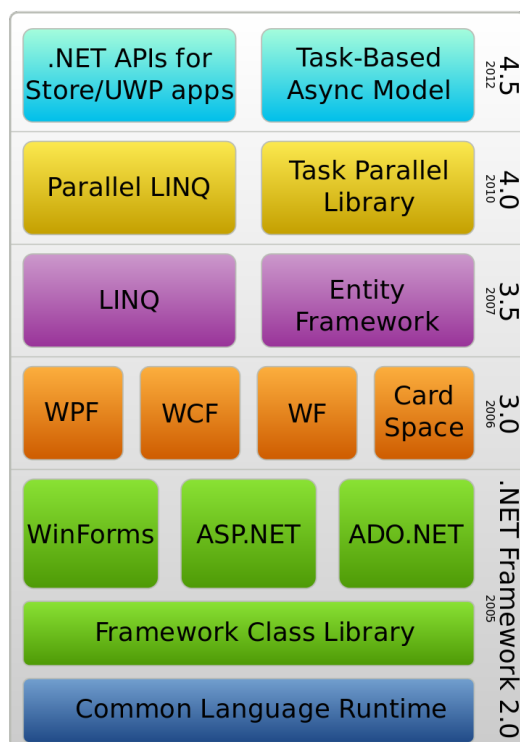
Важно е да се отбележи, че C# е интегриран с широк набор от инструменти и технологии, което го прави мощен инструмент за разработка на разнообразни приложения – от уеб базирани до десктоп и мобилни приложения.



.NET

.NET е комплексна софтуерна платформа, разработена от Microsoft, която предоставя мощни инструменти и средства за създаване, разгръщане и управление на разнообразни приложения. Тя включва голям набор от технологии и компоненти, които се използват в различни области на софтуерната разработка. Основните компоненти на .NET включват:

- *Common Language Runtime (CLR)*: CLR е виртуална машина, която изпълнява код, написан на езиците за програмиране на .NET, като C#, Visual Basic.NET, F# и други. Тя осигурява управление на паметта, обработка на изключения, управление на ресурсите и много други функционалности, които гарантират сигурност и надеждност на изпълнявания код.
- *Framework Class Library (FCL)*: FCL е колекция от готови за използване библиотеки и компоненти, които предоставят различни функционалности за разработка на приложения. Те включват работа с мрежи, файлова система, бази данни, графични интерфейси, уеб разработка и много други.
- *Езици за програмиране*: .NET поддържа различни езици за програмиране, включително C#, Visual Basic.NET, F# и други. Тези езици предоставят разнообразни възможности за създаване на приложения и позволяват на разработчиците да изберат най-подходящия за тях език според нуждите на проекта.
- *ASP.NET*: ASP.NET е платформа за създаване на уеб приложения и услуги, които се изпълняват в средата на .NET. Тя предоставя инструменти за създаване на динамични уеб страници, обработка на заявки от клиенти и много други. Проектът „Система за данъчни услуги“ е разработен на ASP.NET
- *.NET Core*: .NET Core е модерно разширение на .NET платформата, което е преносимо и отворено с отворен код. Той поддържа разработка на приложения за различни операционни системи, включително Windows, Linux и macOS, и предлага по-голяма гъвкавост и производителност в сравнение със старите версии на .NET.



Microsoft SQL Server

Microsoft SQL Server е изключително разнообразна и мощна релационна база данни, която предлага обширен набор от възможности и функционалности за съхранение, управление и анализ на данни. Това са някои от основните аспекти и характеристики, които правят SQL Server една от водещите бази данни в индустрията:

- **Многоплатформеност и възможност за развитие:** SQL Server е наличен за различни операционни системи, включително Windows и Linux, като предоставя възможности за изграждане на системи с висока наличност и гъвкавост в управлението на ресурсите. Той се скалира от малки до много големи инсталации, позволявайки на организациите да се адаптират към различни обеми от данни и заявки.
- **Широк спектър от инструменти и услуги:** SQL Server предоставя разнообразни инструменти и услуги за разработка, управление и анализ на данни. Това включва SQL Server Management Studio (SSMS) за администрация и управление на бази данни, SQL Server Integration Services (SSIS) за ETL процеси, SQL Server Reporting Services (SSRS) за създаване на отчети и SQL Server Analysis Services (SSAS) за анализ на данни.
- **Вградена сигурност и защита на данните:** SQL Server предлага високо ниво на сигурност и защита на данните чрез различни механизми, включително ролево базирано управление на достъпа, криптиране на данни, аудит на дейността и мониторинг на сигурността. Това позволява на организациите да се съобразяват със законодателството за защита на данните и да предотвратяват нарушения на сигурността.
- **Висока производителност и оптимизация на заявките:** SQL Server е оптимизиран за работа с големи обеми от данни и интензивни заявки. Той предлага механизми за оптимизация на заявките, индексирание на данните и кеширане на резултатите, което осигурява бърз и ефективен достъп до информацията.
- **Интеграция с други платформи и приложения:** SQL Server се интегрира лесно с други платформи и приложения, включително различни работнически инструменти, облачни услуги като Microsoft Azure, бизнес приложения и много други. Това позволява на организациите да създадат комплексни и свързани системи за управление на данни.



Microsoft®
SQL Server®

SQL Server Management Studio

SQL Server Management Studio (SSMS) е интегрирана среда за разработка и управление на бази данни, създадена от Microsoft, която предоставя богат набор от инструменти и възможности за администриране на SQL Server и свързани бази данни. Със своя графичен интерфейс и мощни функционалности, SSMS е незаменим инструмент за администратори, разработчици и анализатори на данни. Ето по-подробно за някои от ключовите възможности и функционалности на SQL Server Management Studio:

- **Управление на бази данни и обекти:** SSMS позволява на потребителите да управляват бази данни и техните обекти, като създават, променят и изтриват таблиците, индексите, изгледите, процедурите и тригерите. Те могат лесно да преглеждат структурата на базите данни и да извършват различни операции за управление върху тях.
- **Разработка и изпълнение на SQL заявки:** С помощта на SSMS потребителите могат да създават и изпълняват SQL заявки към базите данни. Това включва извличане, обновяване, изтриване и вмъкване на данни, както и изпълнение на сложни заявки за анализ и обработка на информацията.
- **Мониторинг на производителността и диагностика:** SSMS предоставя инструменти за мониторинг на производителността на SQL Server и базите данни. Потребителите могат да проследяват използването на ресурси, активността на сървъра и да диагностицират проблеми, които могат да възникнат в работата на системата.
- **Интегриране с други инструменти и услуги:** SSMS се интегрира с различни инструменти и услуги на Microsoft, като Azure Portal, Visual Studio и Azure Data Studio. Това позволява на потребителите да работят с различни услуги и ресурси в облака, свързани с SQL Server.
- **Сигурност и управление на потребители:** Платформата предоставя инструменти за управление на сигурността на базите данни, включително управление на потребителските роли, правата за достъп и одит на дейността. Това е от съществено значение за осигуряване на защита на данните и спазване на регулаторните изисквания.



Microsoft®
SQL Server
Managemer

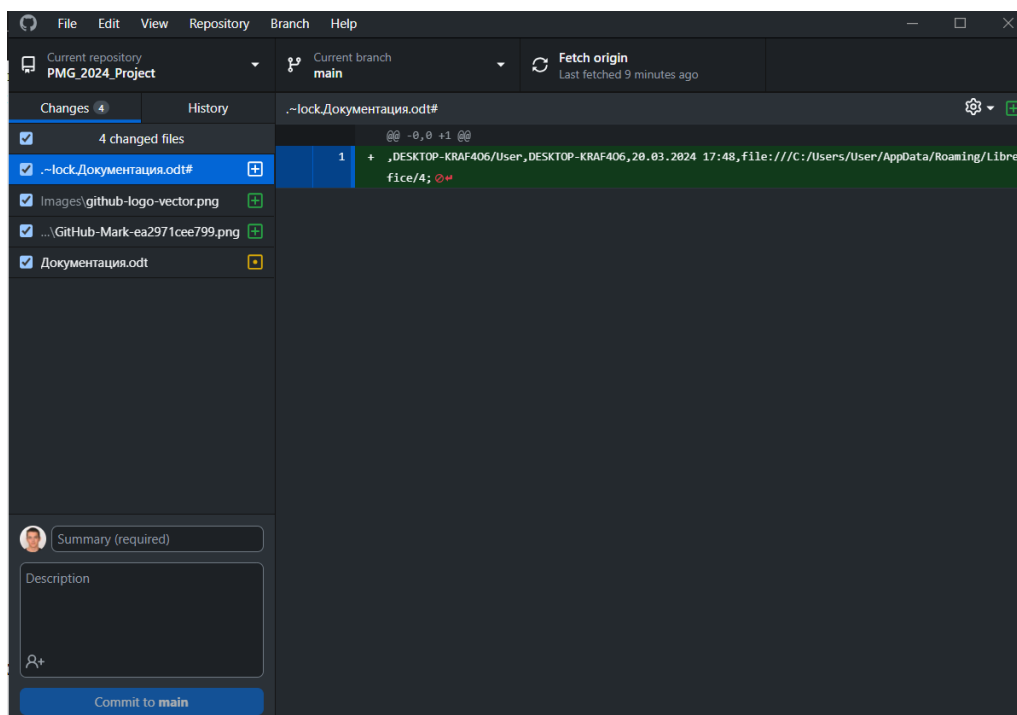
Github

GitHub е уеб-базирана платформа за хостинг на софтуерни проекти, която предлага инструменти за управление на изходния код, сътрудничество и контрол на версиите. Това е основният хъб за разработка и сътрудничество на програмен код, където програмисти от целия свят могат да работят заедно по проекти, да споделят код, да преглеждат промените и да управляват задачите си.

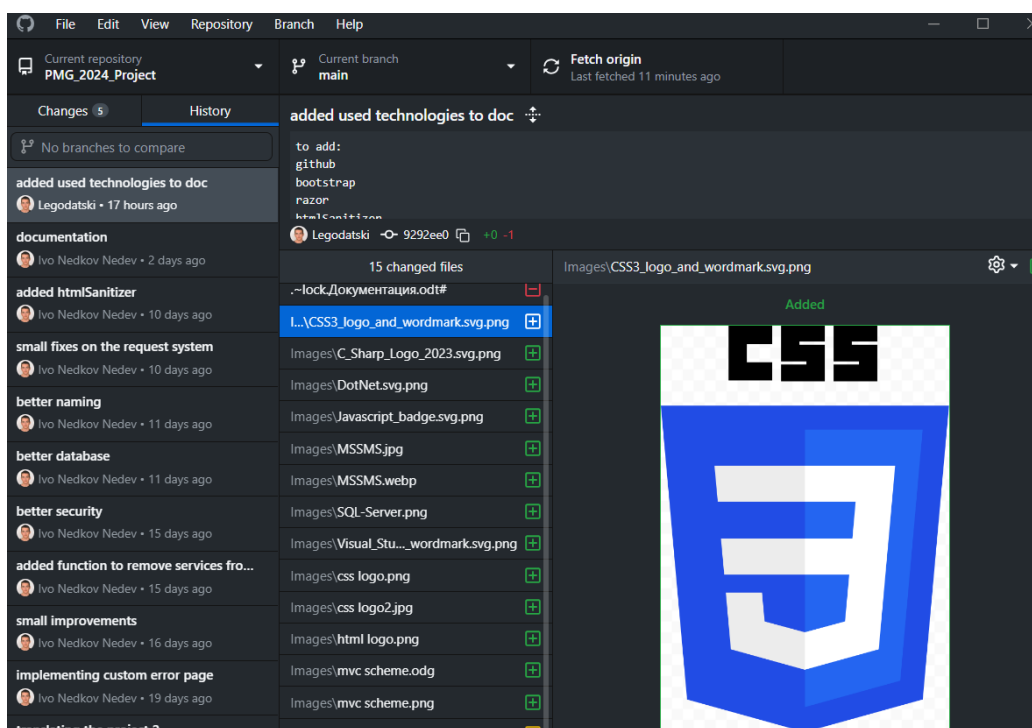
GitHub позволява на потребителите да създават и публикуват репозиторията за своите проекти, които могат да бъдат достъпни за публично или частно използване. Той предоставя инструменти за управление на промените в кода, така че програмистите могат да работят ефективно в екип и да следят различните версии на кода.

GitHub също така поддържа различни функции за сътрудничество, като например преглед на кода, проблеми и задачи, коментари и обсъждания, които правят комуникацията между участниците в проекта по-лесна и ефективна.

GitHub Desktop, от друга страна, е десктоп приложение, което предоставя графичен потребителски интерфейс за работа с GitHub. То позволява на потребителите да управляват репозиторията си, да преглеждат историята на промените, да правят нови клонове и да синхронизират промените си с централното репозитори на GitHub. GitHub Desktop е интуитивен и лесен за използване инструмент, който прави работата с GitHub още по-удобна и достъпна за всички потребители.



Интерфейсът на GitHub Desktop за добавяне промени в проекта „Система за данъчни услуги“. Като можеш да видим всички променени файлове, които ще бъдат комитнати.



Всички промени по приложението за сега, кога са направени, от кога са направени. [В сайта на GitHub също могат да бъдат видени комитите.](#)

Bootstrap

Bootstrap е отворен източник фронтенд рамка за уеб разработка, създадена от екипа на Twitter. Тя предоставя набор от инструменти за бързо и лесно създаване на модерни и отзивчиви уеб приложения и уеб сайтове. Ето някои от основните характеристики и функционалности на Bootstrap:

- Bootstrap включва голям набор от готови компоненти за изграждане на уеб интерфейси, включително бутони, форми, навигационни менюта, модални прозорци и много други. Тези компоненти са лесни за използване и могат да бъдат персонализирани спрямо конкретните нужди на проекта.
- Bootstrap е известен със своя отзивчив дизайн, който позволява на уеб сайтовете да се адаптират автоматично към различни устройства и екрани, включително настолни компютри, таблети и мобилни телефони. Това прави Bootstrap идеален избор за създаване на мобилно-приятни уеб приложения.
- Bootstrap е модулен и лесен за употреба, като позволява на разработчиците да използват само тези компоненти, които са им необходими за техния проект. Това го прави подходящ за начинаещи и опитни уеб разработчици.
- Bootstrap осигурява съвместимост с широк набор от браузъри, включително последните версии на Google Chrome, Mozilla Firefox, Microsoft Edge, Safari и други

популярни браузъри. Това гарантира, че уеб сайтовете, създадени с Bootstrap, ще работят коректно на всички поддържани платформи.

- Bootstrap предлага богати теми и шаблони, които могат да бъдат използвани за бързо стартиране на проекти. Тези теми включват различни стилове и компоненти, които могат да бъдат персонализирани спрямо уникалните изисквания на даден проект.



Razor

Razor е програмен модел и синтаксис за създаване на динамични уеб страници и уеб приложения в рамките на ASP.NET. Той е основен инструмент за създаване на страници и шаблони в ASP.NET MVC, ASP.NET Web Pages и ASP.NET Core. Разработен от Microsoft, Razor предоставя чист и ефективен начин за комбиниране на HTML и C# или VB.NET код в едно, което прави създаването на динамично генерирани уеб страници лесно и интуитивно.

HtmlSanitizer

HTMLSanitizer е библиотека или инструмент, който се използва за сигурно и надеждно почистване и обработка на HTML код. Той е предназначен да предпази уеб приложенията от потенциални атаки и уязвимости, които могат да възникнат от злонамерен HTML код, който се въвежда от потребителите.

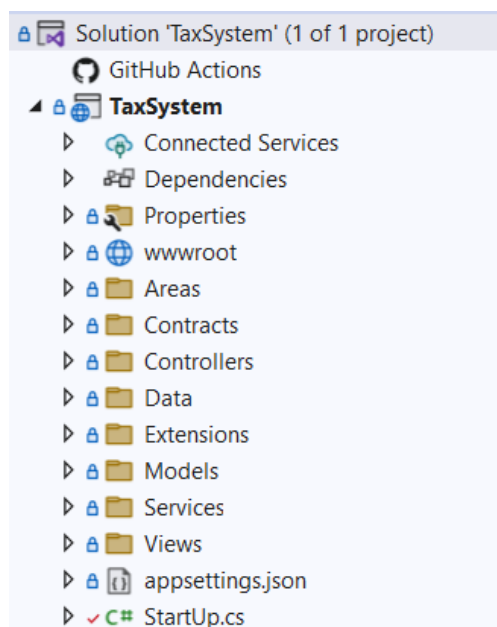
Основната функция на HTMLSanitizer е да премахва или изолира потенциално опасни елементи или атрибути от HTML кода, като например скриптове, вградени обекти, стилове, JavaScript събития и други. Това помага да се предотврати изпълнението на нежелани скриптове или код в контекста на уеб приложението и да се осигури защита на потребителите от атаки като XSS (Cross-Site Scripting).

HTMLSanitizer често предоставя и допълнителни възможности за настройка и конфигурация, които позволяват на разработчиците да дефинират правила и политики за разрешаване или забрана на определени видове HTML елементи или атрибути в зависимост от нуждите на тяхното приложение.

Практическа част

Конструкция на проекта

За да се спазят ООП принципите на програмирането, данните в проекта са разделени на папки, като всяка папка съхранява файлове с определена функция. Сега ще бъдат разгледани приложенията, имплементацията и архитектурата на файловете.



Архитектурата на проекта

Startup.cs

Startup.cs е файлът, който ще се зареди, когато пуснем (build-нем) проекта. Той отговаря за множество настройки като изискванията за потребителска парола, адресите, по които ще работят линковете в сайта и много други.

В него се пояснява и връзката с базата данни, като се използва и appsettings.json. За да осъществи връзка на проекта му трябва и `ConnectionString`, в който се съдържа информация за базата.

Също така, за да се създаде база данни трябва да използваме `Package Manager Console` и да изпълним командата `UpdateDatabase`.

```

var connectionString = builder.Configuration.GetConnectionString("DefaultConnection") ??
    throw new InvalidOperationException("Connection string 'DefaultConnection' not found.");
builder.Services.AddDbContext<ApplicationDbContext>(options =>
    options.UseSqlServer(connectionString));
builder.Services.AddDatabaseDeveloperPageExceptionFilter();

```

Намира връзката с име "DefaultConnection" и използва определената база или хвърля грешка ако не я намери.

```

{
  "ConnectionStrings": {
    "DefaultConnection":
      "Server=DESKTOP-AHPOV9V\\SQLEXPRESS;Database=TaxSystem;Trusted_Connection=True;TrustServerCertificate=True;"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*"
}

```

appsettings.json

В Startup.cs също се настройват и сървисите, които ще бъдат използвани в проекта. Като има три вида сървиси – Scoped, Singleton и Transient. Разликата между тях е времето на живот обектите.

- Scoped
 - При регистрация на услуга като "scoped", контейнерът за инжектиране на зависимости (DI контейнерът) създава единствена инстанция на услугата за всеки HTTP request.
 - Това означава, че в рамките на един HTTP request, всички компоненти, които са заявили тази услуга, ще получат една и съща инстанция.
 - След като HTTP request завърши, контейнерът освобождава тези обекти и те се изтриват.
- Transient
 - При регистрация на услуга като "transient", контейнерът за инжектиране на зависимости (DI контейнерът) създава нова инстанция на услугата всеки път, когато е поискана.
 - Това означава, че всеки път, когато компонент заяви тази услуга, ще получи нова инстанция на нея.
 - Няма споделяне на състояние между различните компоненти, които използват тази услуга.

- Singleton
 - При регистрация на услуга като "singleton", контейнерът за инжектиране на зависимости (DI контейнерът) създава само една инстанция на услугата за целия живот на приложението.
 - Това означава, че всяко извикване на услугата в приложението ще използва същата инстанция на нея.
 - Тази инстанция се съхранява и се използва от всички компоненти, които изискват тази услуга.

```
builder.Services.AddScoped<IAdminService, AdminService>();
builder.Services.AddScoped<IAmenityService, AmenityService>();
builder.Services.AddScoped<IRequestService, RequestService>();
builder.Services.AddScoped<IDeskService, DeskService>();
```

Добавяне на сървисите в проекта, като използваме Scoped сървиси, защото са най-подходящи за функционалността на проекта.

ApplicationBuilderExtensions.cs

В проекта има автоматично генерирани роли и профили. Те се създават чрез метода `SeedRoles` в класа – `ApplicationBuilderExtensions`, които се намират в папката `Extensions`. Ролите са `User`, `Worker` и `Admin`. Всяка от тях дава на различните потребители уникални за тях специални права и възможности за ползване на проекта. Също така се създава и администраторски профил, на който всичките му параметри са точно определени с първото стартиране на проекта. Този клас се извиква в `Startup.cs` с `app.SeedRoles()`;

```
if (!await roleManager.RoleExistsAsync("Admin"))
{
    IdentityRole adminRole = new IdentityRole("Admin");
    await roleManager.CreateAsync(adminRole);
}

if (!await roleManager.RoleExistsAsync("User"))
{
    IdentityRole userRole = new IdentityRole("User");
    await roleManager.CreateAsync(userRole);
}

if (!await roleManager.RoleExistsAsync("Worker"))
{
    IdentityRole userRole = new IdentityRole("Worker");
    await roleManager.CreateAsync(userRole);
}
```

Създаване на ролите

```

string adminId = "35bbae7d-b2a0-472a-8137-e8df5f4ac614";
ApplicationUser admin = await userManager.FindByIdAsync(adminId);

if (admin == null)
{
    await userManager.CreateAsync(new ApplicationUser()
    {
        Id = adminId,
        FirstName = "Admin",
        PhoneNumber = "+359885778908",
        LastName = "Adminov",
        Email = "admin1@gmail.com",
        UserName = "Admin"
    }, "admin123");

    admin = await userManager.FindByEmailAsync("admin1@gmail.com");
}

await userManager.AddToRoleAsync(admin, "Admin");

```

Създаване на администраторския профил. Като първо се проверява дали вече не е създаден.

ApplicationDbContext.cs

ApplicationDbContext е класът, който отговаря за връзката между приложението и базата данни. Чрез него се извършват CRUD операциите върху базата данни. Допълнително той определя кои от класове, създадени от нас в приложението, ще бъдат направени в базата данни, какъв да бъде техния дизайн и какви да са връзките между всяка от таблиците.

7 references

```
public DbSet<Desk> Desks { get; set; }
```

10 references

```
public DbSet<Request> Requests { get; set; }
```

17 references

```
public DbSet<Amenity> Services { get; set; }
```

6 references

```
public DbSet<DeskAmenity> DeskService { get; set; }
```

По структурата на класовете Desk, Request, Amenity и DeskAmenity ще бъдат направени таблици в базата данни.

За определянето на колоните на таблиците в базата данни съм повече използвам Data Annotations в самите класове, но ApplicationDbContext може да използва Fluent API да свърши работа. Аз намирам DataAnnotations за по-лесни за работа и за четене, но Fluent API

ни предлага възможността да създаване по-сложен дизайн на таблиците в базата данни като сложни ключове и други. Например [Required] означава, че задължителното попълване на конкретния атрибут.

```

20 references
public class Amenity
{
    1 reference
    public Amenity()
    {
        Desks = new List<DeskAmenity>();
    }

    [Key]
    6 references
    public int Id { get; set; }

    [Required(ErrorMessage = GlobalConstants.RequiredErrorMsg)]
    [Display(Name = "Название")]
    20 references
    public string Name { get; set; }

    [Required(ErrorMessage = GlobalConstants.RequiredErrorMsg)]
    [Display(Name = "Описание")]
    10 references
    public string Description { get; set; }

    [Required(ErrorMessage = GlobalConstants.RequiredErrorMsg)]
    [Display(Name = "Необходимо време за извършване в минути")]
    12 references
    public string RequiredMinutes { get; set; }

    [Required]
    7 references
    public ICollection<DeskAmenity> Desks { get; set; }
}

```

Използване на DataAnnotations да определят параметрите на различните полета на таблицата Services.

```

0 references
protected override void OnModelCreating(ModelBuilder builder)
{
    base.OnModelCreating(builder);

    builder.Entity<Request>()
        .HasOne(x => x.Desk)
        .WithMany(x => x.Requests)
        .OnDelete(DeleteBehavior.Restrict);

    builder.Entity<DeskAmenity>()
        .HasKey(x => new { x.ServiceId, x.DeskId });

    builder.Entity<DeskAmenity>()
        .HasOne(x => x.Desk)
        .WithMany(x => x.Amenities)
        .HasForeignKey(x => x.DeskId);

    AlterUser(builder);
}

```

Използване на Fluent API, за конфигуриране на по-сложните параметри на определени таблици.

Също така ApplicationDbContext.cs ни дава възможността да променя автоматично генерираните таблици. В проекта съм добавил две нови колони към таблицата AspNetUsers чрез класа ApplicationUser. Като е важно да се уточни кой клас ще отговаря за профилите в ApplicationDbContext.cs и Startup.cs.

```
41 references
public class ApplicationUser : IdentityUser
{
    [Required]
    9 references
    public string FirstName { get; set; }

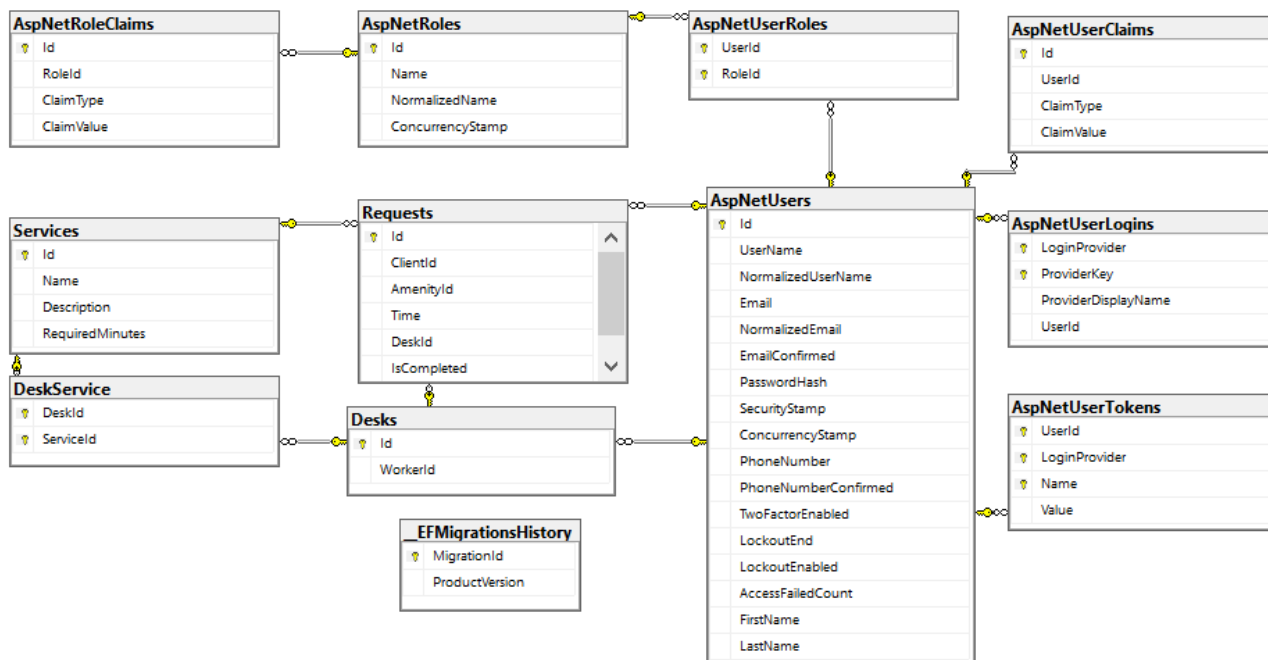
    [Required]
    10 references
    public string LastName { get; set; }
}
```

Класът ApplicationUser

Миграции

За да промени дизайна на базата данни от приложението, се използват миграции. Те съдържат код, който ще бъде преведен на SQL, за да може да бъде изпълнен от базата данни. Те се намират в Data\Migrations. Като информацията за изпълнените миграции се пази в отделна таблица в базата данни.

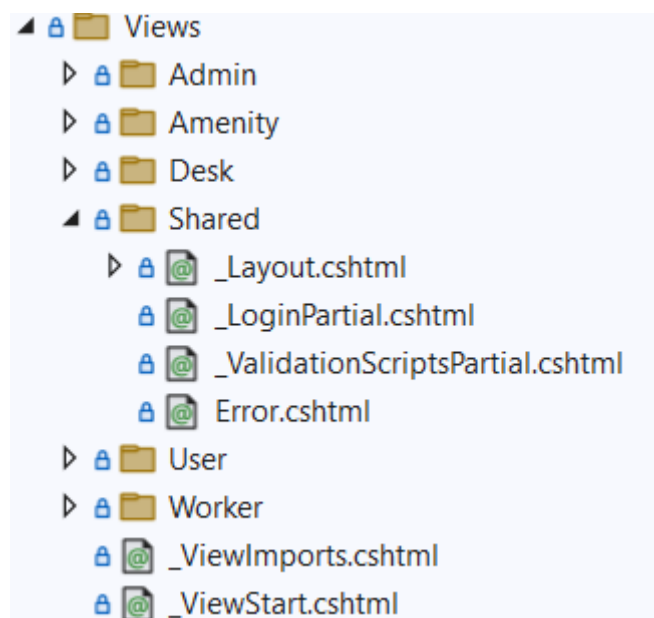
```
migrationBuilder.CreateTable(
    name: "Requests",
    columns: table => new
    {
        Id = table.Column<int>(type: "int", nullable: false)
            .Annotation("SqlServer:Identity", "1, 1"),
        ClientId = table.Column<string>(type: "nvarchar(450)", nullable: false),
        AmenityId = table.Column<int>(type: "int", nullable: false),
        Time = table.Column<string>(type: "nvarchar(max)", nullable: false),
        DeskId = table.Column<int>(type: "int", nullable: false),
        IsCompleted = table.Column<bool>(type: "bit", nullable: false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_Requests", x => x.Id);
        table.ForeignKey(
            name: "FK_Requests_AspNetUsers_ClientId",
            column: x => x.ClientId,
            principalTable: "AspNetUsers",
            principalColumn: "Id",
            onDelete: ReferentialAction.Cascade);
        table.ForeignKey(
            name: "FK_Requests_Desks_DeskId",
            column: x => x.DeskId,
            principalTable: "Desks",
            principalColumn: "Id",
            onDelete: ReferentialAction.Restrict);
        table.ForeignKey(
            name: "FK_Requests_Services_AmenityId",
            column: x => x.AmenityId,
            principalTable: "Services",
            principalColumn: "Id",
            onDelete: ReferentialAction.Cascade);
    });
```



Всички таблици в проекта, параметрите им и връзките между тях

Изгледи (Views)

В архитектурата MVC, представянето е интерфейсът, чрез който потребителят взаимодейства с приложението. Основната му задача е да представя данните и информацията на крайния потребител. Представянето не съдържа бизнес логика и се ограничава само до представяне на информацията. В проекта всички изгледи се намират в папката Views и са разделени в подпапки спрямо кой контролер ще ги използва. Като е важно имената на подпапките да съвпадат с имената на контролерите, които ги използват. Също така има и подпапка Shared, където се намират изгледи които ще бъдат използвани от няколко контролера като Error.cshtml и _Layout.cshtml.



Допълнително, в папката Views се намират и файловете с високо значение за всички изгледи `_ViewImports.cshtml` и `_ViewStart.cshtml`.

`_ViewImports.cshtml` обединява всички `using` тагове за всички изгледи в един файл. Така се предвратява дублирането на код и се спазват ООП принципите.

`_ViewStart` дефинира кой ще е главният изглед в проекта, спрямо който ще бъдат изградени другите изгледи.

```
@using TaxSystem
@using TaxSystem.Models
@using TaxSystem.Models.User
@using TaxSystem.Models.Amenity
@using TaxSystem.Models.DeskModels
@using TaxSystem.Models.Requests
@using TaxSystem.Data
```

Съдържание на `_ViewImports.cshtml`

```
@{
    Layout = "_Layout";
}
```

Съдържание на `_ViewStart.cshtml`

За да може потребителят да подаде информация на приложението със използвал формуляри, които изпращат информация към контролерите.

```
<form id="account" method="post">
  <hr />
  <div asp-validation-summary="ModelOnly" class="text-danger"></div>
  <div class="text-center mb-3">
    <label asp-for="@Model.Email" class="form-label"></label>
    <input asp-for="@Model.Email" class="form-control" aria-required="true" />
    <span asp-validation-for="@Model.Email" class="text-danger"></span>
  </div>
  <div class="text-center mb-3">
    <label asp-for="@Model.Password" class="form-label"></label>
    <input asp-for="@Model.Password" type="password" class="form-control" aria-required="true" />
    <span asp-validation-for="@Model.Password" class="text-danger"></span>
  </div>
  <div class="text-center">
    <button id="login-submit" type="submit" class="btn btn-primary">Влез</button>
  </div>
</form>
```


Показаният формуляр се използва, за да потребителите да влязат в акаунтите си. Те ще трябва да въведат имейл и паролата си. За тяхно улеснение въведените данни се проверяват още преди да са изпратени към базата данни с `_ValidationScriptsPartial`.

```
@section Scripts {  
    <partial name="_ValidationScriptsPartial" />  
}
```

Използването на `_ValidationScriptsPartial` в изгледите

Обмена на информация между контролерите и изгледите се осъществява чрез допълнителни класове, които се не записват в базата данни. Те имат за цел да показват на потребителите само нужната за тях информация и да предпазват базата данни. Отново чрез `DataAnnotations` се управляват пропъртитата на тези класове. Също така се използват за валидацията на данните. Като във всяка страница трябва да се поясни кой модел ще използва. Всички модели се намират в папката `Models`.

5 references

```
public class LoginModel
```

```
{
```

```
    [Required(ErrorMessage = GlobalConstants.RequiredErrorMsg)]
```

```
    [EmailAddress]
```

```
    [Display(Name = "Имейл")]
```

5 references

```
    public string Email { get; set; }
```

```
    [Required(ErrorMessage = GlobalConstants.RequiredErrorMsg)]
```

```
    [DataType(DataType.Password)]
```

```
    [Display(Name = "Парола")]
```

4 references

```
    public string Password { get; set; }
```

```
}
```

Моделът, по който се дава информация за влизането на потребител в профила си.

```
@model AllUsersQueryModel
```

Използване на класа `AllUsersQueryModel` в страницата `AllUsers.cshtml`

Също така изгледите се използват да визуализират информация на потребителя. Като това отново се осъществява чрез модели. Най-често съм използвал таблици за показване на данните.

Название	Описание	Необходимо време	Бюро	Първи имена на работниците	Фамилии на работниците
Предел на данъци;	Служител преглежда вашите данъци като ви предоставя информация за тях	25;			
Проверка за спазване за застрояване;	Проверка за спазване определената линия на застрояване, заснемане и нанасяне на мрежи и съоръжения на техническата инфраструктура	40;			
Разрешение за реклама;	Издаване на Разрешение за поставяне на рекламно информационен носител/ РИН/ върху частна или държавна собственост	30;			
Разрешение за специално ползване на пътя;	Издаване на разрешение за специално ползване на пътя чрез изграждане на търговски крайпътен обект и пътни връзки към него	50;			

Таблица за наличните услуги

Моделите могат едновременно да визуализират данни и да получават информация от потребителя. Добър пример за това са всички търсачки в приложение, където с един модел „взимаме“ по коя дума да търсим данни и приемаме исканите данни от базата данни.

```

5 references
public class AllAmenitiesQueryModel
{
    0 references
    public AllAmenitiesQueryModel()
    {
        Services = new HashSet<AmenityViewModel>();
    }
    3 references
    public string SearchTerm { get; set; } = null;
    4 references
    public IEnumerable<AmenityViewModel> Services { get; set; }
}

```

Моделът за извличане на потърсените услуги от потребителя

Допълните чрез изгледите можем да контролираме кой потребител има достъп кой данни и функции. За тази функционалност използваме Razor.

```

@if (User.IsInRole("Admin"))
{
    <th>
        <a asp-action="Edit" asp-route-id="@s.Id">Обработи</a>
    </th>
    <th>
        <a asp-action="Delete" asp-route-id="@s.Id" method="post">Изтрий</a>
    </th>
}

```

Само администраторът ще може да достъпи бутоните да промени или изтрие дадена услуга.

Контролери (Controllers)

В архитектурата MVC (Model-View-Controller), контролерите са компонентите, отговорни за управлението на приложението и обработката на входящите заявки от потребителите. Основната им задача е да свържат моделите и изгледите, като при това извършват бизнес логика и решават какво изглед да бъде показан на потребителя.

Контролерите приемат HTTP заявки (или друг вид заявки, в зависимост от контекста) и извличат необходимата информация от моделите, които обикновено представляват данните на приложението. След това контролерите прилагат бизнес логиката, която може да включва валидация, обработка на данни и вземане на решения.

След като бизнес логиката е изпълнена, контролерите решават кой изглед (или шаблон) да се покаже на потребителя, за да се представят резултатите от заявката. Това може да включва рендиране на HTML страници, връщане на JSON или други формати за данни, въз основа на изискванията на клиента.

Следователно, контролерите играят ролята на посредник между моделите и изгледите, което гарантира разделение на отговорностите и улеснява поддръжката и разширяването на приложението. Те обикновено са асоциирани с определени URL адреси или пътища в приложението и обработват заявките, насочвайки ги към съответните изгледи или действия.

Контролерите отговарят за преноса и валидацията на данни между изгледите и базата данни. Те също изпълняват бизнес логиката на приложението. Важно е да не се оставя валидацията само в изгледите за по-силна защита на приложението.

Също така контролерите разпределят кой потребител ще има достъп до какви функционалности на приложението. Това съм го постигнал като съм авторизирал целия контролер или само определени негови методи. Като не е задължително да конкретизираме дадени роли за авторизация.

```
[Authorize(Roles = "Admin")]  
1 reference  
public class AdminController : Controller  
{
```

Авторизацията на контролера Admin така че потребители с ролята Admin да могат използват негови функции.

Най-често валидацията на входните данни се извършва чрез ModelState.IsValid, където отново се проверяват параметрите като Required и MinLength на информацията въведена от потребителя. Също так в контролерите съм използвал и HTMLSanitizer за допълнителна валидация.

```

string email = sanitizer.Sanitize(model.Email);
string password = sanitizer.Sanitize(model.Password);

if (!ModelState.IsValid ||
    email == null ||
    password == null)
{
    return View(model);
}

```

Проверка на данните, когато потребител иска да си влезе в профила.

Контролерите са разделени на отделни методи като всеки метод отговаря за различна функция. Много често всеки метод има за цел да зареди даден изглед. Има много видове методи по тяхната цел, но в това приложение съм използвал само видове Post и Get.

- GET методи
 - Служат за взимане на информация от базата данни и показване на тази информация на потребителя.
 - Параметрите се предават чрез URL адреса и се виждат ясно, когато се извиква определен изглед.
 - Използват се по подразбиране, когато потребителят просто посещава определена страница или изпраща GET заявка.

3 references

```

public async Task<IActionResult> AllUsers([FromQuery] AllUsersQueryModel query)
{
    var queryResult = _adminService.GetAllUsers(
        query.SearchTerm,
        query.RoleName);

    query.Users = await queryResult;

    return View(query);
}

```

GET метод за взимане на всички потребители от базата данни.

- POST методи
 - Служат за качване на информация към базата данни и обработка на данни, които потребителят въвежда във формуляри или други средства за въвеждане на данни.

- Параметрите се предават чрез HTTP тялото на заявката и не се показват явно в URL адреса.
- Полезни са за операции като създаване, редактиране или изтриване на данни.

```
[Authorize(Roles = "Admin")]
[HttpPost]
0 references
public async Task<IActionResult> Edit(Amenity input)
{
    if (!ModelState.IsValid)
    {
        return View();
    }
    await _service.Edit(input);

    return RedirectToAction(nameof(All));
}
```

POST метод за редактиране на дадена услуга

Сървиси (Services)

Бизнес логиката, която трябва да се изпълни в контролерите е изнесена в отделни класове като всеки сървис трябва да бъде описан в Startup.cs (страница 17). Всеки от тези класове отговаря за отделна функционалност от проекта и наследява родителски клас, където са описани неговите методи. Всеки отделен сървис приблизително отговаря за отделна таблица в базата данни, изпълнявайки различни функции за нея.

```
2 references
public Task Add(AddDeskViewModel input);

2 references
public Task<IEnumerable<Desk>> GetAllDesks(string? searchTerm);

2 references
public Task<IEnumerable<ApplicationUser>> GetAllWorkersWithoutDesks();

2 references
public Task AddDeskService(int deskId, string serviceName);

2 references
public Task<Desk> GetDeskByWorkerId(string id);

2 references
public Task RemoveDesksSer(int deskId, string serName);

2 references
public Task Delete(int id);
```

Методите на сървиса DeskService.

Всички сървиси изпълняват почти еднакви функции спрямо един други. Например, да вземат определен ред от конкретна таблица при подадено Id на реда. Въпреки това, имат и различни дейности. Например, изтриването на заявка за определена услуга е по-различно от изтриването на данни на другите таблици. Това е, защото трябва да се променят времената на всички други заявки на същото бюро.

```
var services = await context.Services.ToArrayAsync();
var request = await context.Requests.FirstOrDefault(x => x.Id == id);
var allrequests = context.Requests.Where(x => x.DeskId == request.DeskId).ToArray();

var delTime = StringToTime(request.Time);
var serTime = double.Parse(request.Amenity.RequiredMinutes);

if (request != null)
    context.Requests.Remove(request);

foreach (var r in allrequests)
{
    var time = StringToTime(r.Time);

    if (time > delTime)
    {
        time = time.AddMinutes(-serTime);
        time = time.AddMinutes(-5);

        if (time.Hour == 12)
        {
            time = new TimeOnly(11, time.Minute);
        }

        if (allrequests.Select(x=>x.Time).Contains(time.ToString()))
        {
            time = time.AddMinutes(serTime);
        }

        r.Time = time.ToString();
    }
}

await context.SaveChangesAsync();
```

Бизнес логиката за изтриване на заявка на потребител.

2 references

```
public async Task DeleteUser(string id)
{
    var user = await context.Users.FindAsync(id);
    var requests = await context.Requests.AnyAsync(x => x.ClientId == user.Id);
    var desks = await context.Desks.AnyAsync(x => x.WorkerId == user.Id);

    if (!requests && !desks)
    {
        context.Users.Remove(user);
        await context.SaveChangesAsync();
    }
}
```

Изтриване на потребител от базата данни.

Алгоритми

В проекта има множество алгоритми за обработването на данни, които се намират в определените сървиси. Два от най-съществените за използването на приложението са за изтриването и създаването на заявки и са в класа `RequestService`.

Алгоритъмът за създаване на заявка, първо, намира желаната услуга от потребителя от базата данни. След това взема всички бюра, които могат да изпълнят исканата услуга и ги подрежда според броя на заявените услуги има за даденото гише. Ако няма намерени гишета алгоритъмът връща съобщение за грешка на контролера и прекратява следващите действия. Избира се бюрото с най-малко заявки. После се декларира отделна променлива, съхраняваща всички заявки към определеното бюро. Следващата стъпка е се създаде променлива `time` от вида `TimeOnly`, която ще съдържа часа, когато клиентът ще трябва да се яви на бюрото. Ако към бюрото няма подадени заявки за изпълнение на каквато и да е услуга, `time` се задава със стойност 8:00 часа сутринта. Ако към бюрото има подадени заявки, `time` се приравнява на часа, когато ще започне последната услуга плюс времетраенето на услугата и още пет минути за почивка на счетоводителя. След това се проверява дали часът за явяване на клиента е 12 - в такъв случай, заявката се прехвърля за 13:00 часа. После се създава променлива от тип `TimeOnly` със стойност часа, в който ще свърши обработването на услугата на клиента, и ако тя е повече от 17:00, връща към съобщение за грешка към контролера.

```
TimeOnly time = new TimeOnly();

if (requests.Any())
{
    time = LastTime(requests.Select(x => x.Time).ToArray());
    var request = await requests.FirstOrDefaultAsync(x => x.Time == time.ToString());
    var ser = context.Services.FirstOrDefault(x => x.Id == request.AmenityId);

    time = time.AddMinutes(5);
    time = time.AddMinutes(double.Parse(ser.RequiredMinutes));

    if (time.Hour == 12)
    {
        time = new TimeOnly(13, 0);
    }

    TimeOnly estimateTime = time;
    estimateTime.AddMinutes(double.Parse(Amenity.RequiredMinutes));

    if (estimateTime.Hour >= 17)
    {
        return GlobalConstants.InvalidTime;
    }
}
else
{
    time = new TimeOnly(8, 0);
}
```

Изчисляване на часа кога клиентът да се яви на гише

Алгоритъмът за изтриване на заявки за услуги цели да изтрие дадена заявка и да промени времето, в което следващите потребители трябва да се явят на гишетата си. Първата му стъпка е да вземе услугите от базата данни и желаната за изтриване заявка. Ако заявката съществува, то тогава всички останали гишета функционират без промени освен бюрото, към което е подадена заявката. От конкретното гише се отделят само заявките, на които се налага да им бъдат променени времената. Създава се нов списък със заявки, като в него се презаписват данните на заявките, които ще бъдат с променени времена, но за времето на определена заявка се използва времето на предходната. След това се изтриват не актуалните заявки и се записва новият списък в базата данни.

```
public async Task Delete(int id)
{
    var services = await context.Services.ToArrayAsync();
    var request = await context.Requests.FirstOrDefaultAsync(x => x.Id == id);

    if (request != null)
    {
        var allrequests = context.Requests.Where(x => x.DeskId == request.DeskId).ToArray();
        var nextRequests = allrequests.Where(x => x.Id >= id).ToList();
        nextRequests = nextRequests.OrderBy(x => x.Id).ToList();
        List<Request> newR = new List<Request>();

        if (nextRequests != null)
        {
            for (int i = 1; i < nextRequests.Count; i++)
            {
                newR.Add(new Request
                {
                    Id = nextRequests[i].Id,
                    DeskId = nextRequests[i].DeskId,
                    AmenityId = nextRequests[i].AmenityId,
                    ClientId = nextRequests[i].ClientId,
                    IsCompleted = nextRequests[i].IsCompleted,
                    Time = nextRequests[i - 1].Time
                });
            }

            context.RemoveRange(nextRequests);
            await context.AddRangeAsync(newR);

            await context.SaveChangesAsync();
        }
    }
}
```

Алгоритъмът за изтриване на заявка

Заклучение

Уеб-базираното приложение „Система за данъчни услуги и справки в общината“ е успешно разработено и успява да постигне поставените цели. Неговата направа е извършена чрез технологиите C#, Visual Studio, HTML, CSS и други. Проектът дава възможността да се използва като уеб приложение за управление на общински услуги. Допълнително позволява на потребителите да подават заявка за определена услуга като чрез алгоритъм се избира най-подходящото бюро за извършване. Те също така могат да видят подадените от тях заявки като могат да ги разделят на готови и негови. Чиновниците могат да проверят всички заявки, който трябва да обработят и да завършат дадена заявка. Администраторът чрез проекта може да създава и редактира нови бюра и услуги и да регистрира работници в системата.

Използвана литература

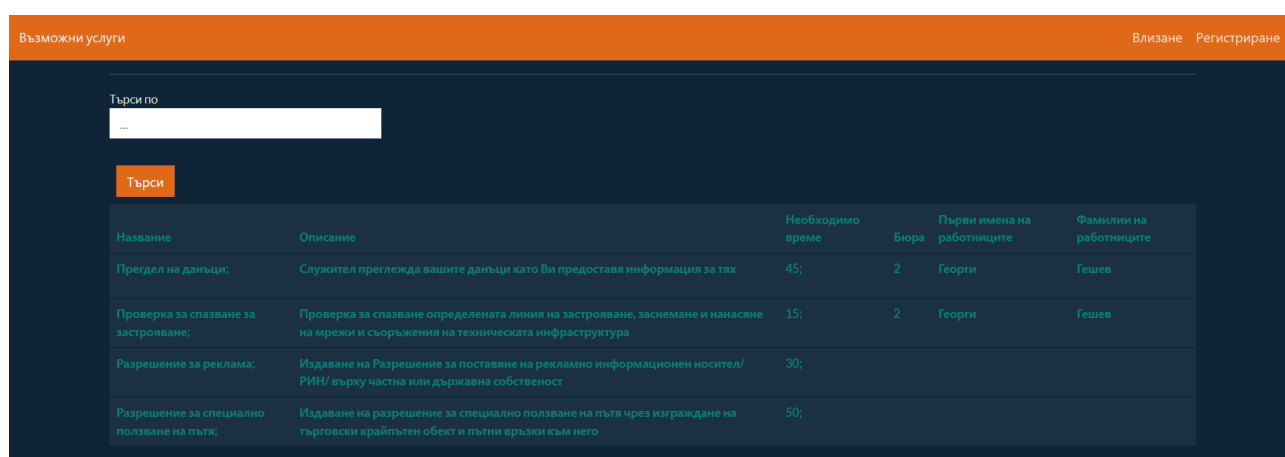
- https://bg.wikipedia.org/wiki/C_Sharp за намиране на подходяща информация за програмния език C#;
- https://en.wikipedia.org/wiki/Visual_Studio за намиране на подходяща информация за Visual Studio;
- https://en.wikipedia.org/wiki/Microsoft_SQL_Server за подбиране на правилна информация за SQL;
- <https://bg.wikipedia.org/wiki/HTML> за намиране на подходяща информация за HTML;
- <https://bg.wikipedia.org/wiki/JavaScript> за намиране на подходяща информация за JavaScript;
- https://en.wikipedia.org/wiki/.NET_Framework за намиране на подходяща информация за .NET;
- <https://en.wikipedia.org/wiki/GitHub> за намиране на подходяща информация за GitHub;
- <https://bg.wikipedia.org/wiki/Bootstrap> за намиране на подходяща информация за Bootstrap;
- <https://chat.openai.com/> за подобрене на документацията;
- <https://learn.microsoft.com/en-us/visualstudio/windows/?view=vs-2022> за проверка на възникналите трудности по разработката на проекта.

Приложение

Използване на проекта

За създаване на локална база данни, във Visual Studio ще трябва да промените файла appsettings.json като замените DESKTOP-KRAF4O6 с името на вашия сървър. След това ще трябва да използвате командата Update-Database в Package Manager Console.

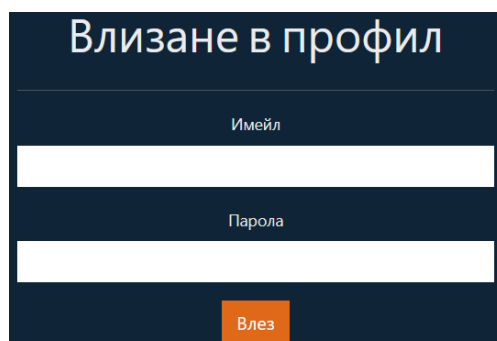
При стартиране проектът показва началната страница, където могат да бъдат видени всички услуги, които се предлагат за изпълнение. Тази страница също може да бъде достигната чрез бутона *Възможни услуги*. Тя показва информация за всяка от услугите като името ѝ, описанието ѝ и на кое гише и от кой работник може да бъде изпълнена. С търсачката потребителят може да филтрира услугите по името и описанието им. Тази страница може да бъде видяна от всеки потребител на приложението без значение неговата роля.



Название	Описание	Необходимо време	Бюра	Първи имена на работниците	Фамилии на работниците
Предел на данъци:	Служител преглежда вашите данъци като Ви предоставя информация за тях	45;	2	Георги	Гешев
Проверка за спазване за застрояване:	Проверка за спазване определената линия на застрояване, заснемане и нанасяне на мрежи и съоръжения на техническата инфраструктура	15;	2	Георги	Гешев
Разрешение за реклама:	Издаване на Разрешение за поставяне на рекламно информационен носител/РИН/ върху частна или държавна собственост	30;			
Разрешение за специално ползване на пътя:	Издаване на разрешение за специално ползване на пътя чрез изграждане на търговски крайпътен обект и пътни връзки към него	50;			

Страницата показваща всички възможни услуги.

В горния ляв ъгъл се намират два бутона за *Влизане* в съществуващ профил или за *Регистриране* на нов профил. При избиране да се регистрира нов профил, потребителят ще бъде пренасочен към формуляр, в който да попълни необходимите си лични данни за създаването на профил. Като при успешно създаване на профил, по подразбиране новият акаунт е за клиент на общината. При успешно попълване на формуляра, потребителят ще бъде изпратен към страницата за *Влизане* в профил, където ще трябва да въведе своята парола и имейл, за да достъпи до профила си. Ако потребителят е вече влязъл в профил, на мястото на бутоните *Регистрация* и *Влизане*, ще има бутон *Излез* за излизане от профила



Влизане в профил

Имейл

Парола

Влез

В проекта има един готов профил на администратор, с имейл: admin1@gmail.com и парола: *admin123*. Администраторът има различни права от обикновените потребители. Например чрез страницата за визуализация на всички налични услуги, той може да променя, добавя или изтрива услуги.

Търси по

Търси

Добави нова услуга

Название	Описание	Необходимо време	Бюра	Първи имена на работниците	Фамилии на работниците	Обработи	Изтрий
Преглед на данъци;	Служител преглежда вашите данъци като Ви предоставя информация за тях	45;	2	Георги	Гешев	Обработи	Изтрий
Проверка за спазване за застрояване;	Проверка за спазване определената линия на застрояване, заснемане и нанасяне на мрежи и съоръжения на техническата инфраструктура	15;	2	Георги	Гешев	Обработи	Изтрий
Разрешение за реклама;	Издаване на Разрешение за поставяне на рекламно информационен носител/РИН/ върху частна или държавна собственост	30;				Обработи	Изтрий
Разрешение за специално ползване на пътя;	Издаване на разрешение за специално ползване на пътя чрез изграждане на търговски крайпътен обект и пътни връзки към него	50;				Обработи	Изтрий

Страницата за всички услуги през администраторския профил

При избиране на *Добави нова услуга*, администраторът ще бъде пренасочен към нова страница, където ще трябва да попълни формуляр и чрез неговото правилно попълване ще бъде отново пренасочен към началната страница. При избиране на *Обработи*, администраторът ще може да промени названието, описанието или необходимото време за извършване на конкретната услуга като автоматично ще му бъдат заредени сегашните данни за избраната услуга.

Редактирай услуга

Название

Проверка за спазване за застрояване

Описание

Проверка за спазване определената линия на застрояване

Необходимо време за извършване в минути

15

Редактирай

Добави услуга

Название

Описание

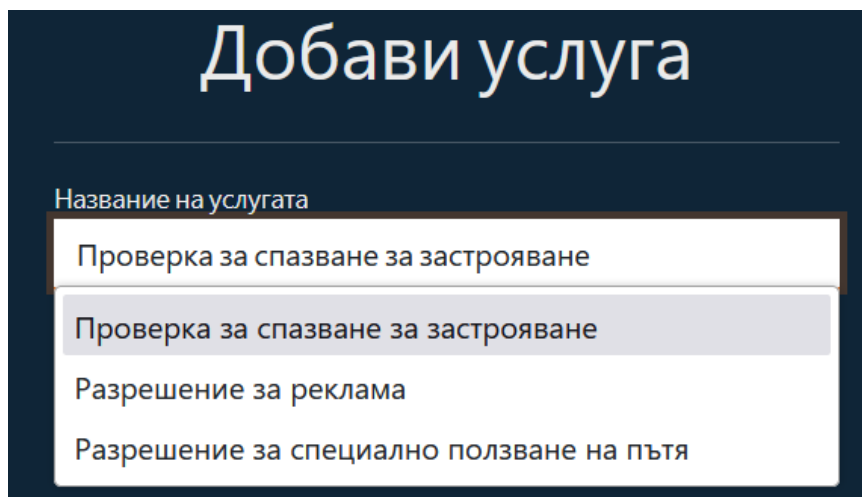
Необходимо време за извършване в минути

Добави

Формулярите за обработването и добавянето на услуги.

Също така с бутона *Изтрий*, администраторът ще изтрие определена услуга. Като при изтриване на услуга, всички заявки за определената услуга също ще бъдат изтрети.

Администраторът също така има възможността да види всички потребители и гишета в проекта чрез линковете *Всички потребители* и *Всички бюра*. При добавянето на гише администраторът трябва само да избере кой работник, ще работи на бюрото. Допълнително, че при гишета, администраторът може да добави или премахне услуги, които се извършват на гишето. При добавянето на услуга към бюро, администраторът трябва да я избере от списък, с всички услуги, които не се извършват на определеното гише. Когато услуга бъде добавена, нейната информация в страницата *Всички услуги*, ще бъде обновена.



Добави услуга

Название на услугата

- Проверка за спазване за застрояване
- Проверка за спазване за застрояване
- Разрешение за реклама
- Разрешение за специално ползване на пътя

Ако администраторът иска да премахне услуга от това тя да се извършва на дадено гише, той трябва да я избере по аналогичен начин като добавянето на услуга. Но ако към желаната за премахване услуга има подадени заявки от потребители, администраторът няма да я види като възможна за изтриване.

Също така е невъзможно изтриването на конкретно гише ако към него има подадени заявки за изпълнение на услуга.

Освен гишета администраторът може да редактира и информацията за профилите на потребителите и клиентите или изцяло да изтрива техните профили. Като не може да изтрие профил на потребител на приложението ако той е подал заявка за изпълнение на услуга. Също така не може да изтрие профил на работник ако към гишето, на което работи е подадена услуга за изпълнение. Администраторът може да филтрира работниците и клиентите по ролята им в сайта или по потребителското им име, първо име и фамилия. Администраторът може да редактира потребителското име, първото име, фамилията, имейла, телефонния номер или ролята на всеки потребител чрез бутона *Редактирай*. Като ще му бъдат автоматично генерирани сегашните стойности на тези параметри на желания за промяна профил.

Потребителско име	Първо име	Фамилия	Имейл	Телефонен номер	Роля	Редактирай	Изтрий
Rabotnik1	Георги	Гешев	rab@gmail.com	3623432515	Worker	Редактирай	Изтрий
Misho	Михаил	Мешев	misho123@gmail.com	14189161897	User	Редактирай	Изтрий

Страница за визуализиране на всички профили.

Редактирай потребител

Username

Rabotnik1

Име

Георги

Фамилия

Гешев

Имейл

rab@gmail.com

Телефонен номер

3623432515

Роля

Worker

Редактирай

Формуляр за редакция на потребител

Приложението също се ползва и от чиновници в общината. Те могат да видят всички услуги, които са изпратени към тяхното *гиле*, виждайки кой имената на клиента, които е подал заявката, телефона му, наименованието на заявката и кога трябва да започне изпълнението ѝ. Като могат да ги филтрират по това дали са завършени или не. Ако конкретна заявка не е готова, те могат да я завършат, с бутона *Завърши услугата*. При завършване на услуга, потребителите също ще могат да видят нейните актуализирани данни.

Заявки

Първо име на клиента	Фамилия на клиента	Телефон на клиента	Име на услугата	Час за започване	Завършеност
Михаил	Мешев	14189161897	Проверка за спазване за застрояване	8:20 AM	<input type="checkbox"/> Завърши услугата

Клиентите на общината също имат различна функционалност от администраторът и от работниците. Те могат да видят всички свой подадени заявки, да подадат нова заявка или да изтрият съществуваща тяхна заявка. При избирането на линка за създаване на нова заявка потребителят ще бъде изпратен към страница, където ще може да избере коя услуга иска да бъде извършена.

Заяви услуга

Название на услугата

Проверка за спазване за застрояване

Проверка за спазване за застрояване

Разрешение за реклама

Разрешение за специално ползване на пътя

Клиентът избира от всички услуги записани в базата данни, независимо дали някое гише ги изпълнява. Ако потребителят избере услуга, която не се предлага от никое бюро, заявката му няма да бъде приета и ще му бъде показано следното съобщение.

Няма подходящо бюро за изпълнение на услугата.

Клиентът може да види всички свой успешно подадени заявки от *Моите заявки*, където ще получи информация за всяка от заявките като име на услугата, време на започване, номер на гишето, на което ще се изпълни и дали е готова. Потребителят може да филтрира заявките по това дали са извършени или не. Също така, може да изтрие дадена заявка, като тогава времената на другите следващите заявки на същото бюро ще бъдат променени.

Моите заявки

[Всички](#)

[Завършени](#)

[Незавършени](#)

Име на услугата	Време на започване	Номер на бюро	Завършеност	
Проверка за спазване за застрояване	8:00 AM	3	<input checked="" type="checkbox"/>	Изтрий
Проверка за спазване за застрояване	8:20 AM	3	<input type="checkbox"/>	Изтрий
Проверка за спазване за застрояване	8:40 AM	3	<input type="checkbox"/>	Изтрий