по професия код 481030 „Приложен програмист“

специалност код 4810301 „Приложно програмиране“

Тема: WEB БАЗИРАНА СИСТЕМА ЗА РЕГУЛАЦИЯ НА ДЪРВОДОБИВА В СТРАНАТА

ДИПЛОМЕН ПРОЕКТ   
  
ДЪРЖАВЕН ИЗПИТ ЗА ПРИДОБИВАНЕ НА ТРЕТА СТЕПЕН НА ПРОФЕСИОНАЛНА КВАЛИФИКАЦИЯ – ЧАСТ ПО ТЕОРИЯ НА ПРОФЕСИЯТА   
  
ИЗПИТНА СЕСИЯ МАЙ 2023

**Автор:**

Радина Величкова, ученичка в 12В клас

**Ръководител:**

Петър Георгиев

**град Бургас**

**СЪДЪРЖАНИЕ**

[1 Увод 2](#_Toc132900038)

[2 Цели и обхват на софтуерното приложение 3](#_Toc132900039)

[3 Анализ на решението 4](#_Toc132900040)

[3.1 Потребителски изисквания и работен процес 4](#_Toc132900041)

[3.2 Примерен потребителски интерфейс 6](#_Toc132900042)

[3.3 Диаграми на анализа 8](#_Toc132900043)

[10](#_Toc132900044)

[3.4 Модел на данните 11](#_Toc132900045)

[3.5 Модел на съдържанието 14](#_Toc132900046)

[4 Дизайн 20](#_Toc132900047)

[4.1 Реализация на архитектурата на приложението 21](#_Toc132900048)

[4.2 Описание на слоевете, предназначението им, библиотеки и методи включени в съответния слой. 22](#_Toc132900049)

[4.2.1 Контролери 22](#_Toc132900050)

[4.2.2 Изгледи 33](#_Toc132900051)

[4.2.3 Модел 35](#_Toc132900052)

[4.2.4 Изброими типове данни (IEnumerable) 36](#_Toc132900053)

[4.3 Организация и код на заявките към база от данни 37](#_Toc132900054)

[4.4 Ефективност и бързодействие на алгоритмичното решение 39](#_Toc132900055)

[4.5 Наличие на потребителски интерфейс (конзолен, графичен, уеб) 40](#_Toc132900056)

[5 Тестване 42](#_Toc132900057)

[6 Заключение и възможно бъдещо развитие 46](#_Toc132900058)

[7 Използвани литературни източници и Уеб сайтове 47](#_Toc132900059)

# 

# Увод

В наши дни има много екологични проблеми засягащи не само планетата ни, но и нас самите, защото ние сме инициаторите на тези бедствия. България е страна богата на много ресурси като един от най-важните е именно дървесината. Страната ни разполага с уникални природни фактори, които предразполагат към изграждането на стабилна икономическа рамка в тази сфера. Всичко това обаче ще може да бъде възможно единствено и само чрез регулация.

Проблемът с износа на дървесина и незаконното обезлесяване води до разруха в икономически аспект, която влияе на всеки един гражданин на страната.

[1]Chart, bar chart

Description automatically generated

Фигура 1

Необходимо е изработване на софтуер, които да регулира реализацията на дърводобив в страната и неговия износ като използва експертни анализи на данни взети от горските стопанства в страната. С помощта на тези данни ще може да се направят нужните изчисления, чрез който да спомогнем за регулирането на количеството дървесина, която бива продавана в чужбина и да предотвратим високите такси породени от недостига на този изключително важен ресурс.

# Цели и обхват на софтуерното приложение

Diagram

Description automatically generatedГлавната цел на проекта е разработка на дигитално решение за управление на зелени зони, а именно изработване на *WEB* базирана система за регулация на дърводобива в страната. Приложението за управление на зелени зони се използва в областта на горското стопанство и дърводобивната промишленост. То ще съдържа данни за горските площи в страната, планираните годишни сечи, разпределението на дървесината за продажба на складове и цехове в страната, за износ и за местните потребители.

Фигура 2

В базата данни на приложението биват включени експертни анализи на данни, взети от горските стопанства, които спомогнат за направата на изчисления за количеството дървесина, която бива продавана в чужбина. Алгоритъмът на приложението използва тези данни за определяне на допустимото количество сеч за даден район, като взима предвид и площта и обема на горските площи.

Главната функция на платформата е да определя какви видове и какво количество сеч е допустимо спрямо площта и обема на горските площи и какъв процент от тях да бъде разпределен за продажба на складове и цехове в страната, за износ и за местните потребители. Информацията за съответните горски площи се извежда от справките на Югоизточното държавно предприятие, град Сливен. [2] Уеб-базираният софтуер позволява на потребителите да визуализират и анализират данните в реално време, като могат да правят справки и да генерират отчети за съответния период. като използва експертни анализи на данни взети от горски стопанства. Тези данни помагат да се направят препоръки за количеството дървесина, която е препоръчително да бъде продавана в чужбина.

Icon, bubble chart

Description automatically generated

Фигура 3

Проектът е предназначен за организации и лица, свързани с горското стопанство и управлението на зелените зони в страната. Това могат да бъдат горски инженери, горски стопани, дърводобивни дружества, административни институции и други заинтересовани граждани.

# Анализ на решението

## Потребителски изисквания и работен процес

Всички потребители си взаимодействат със системата посредством уебсайт. На потребителя се представя началната страница, в която се среща с обща информация относно самата платформа и нейните цели, налични процедури и регулационни решения, както и най-скорошните или предстоящи новини и събития.

Чрез активната навигация и бутоните за пренасочване към друга страница, потребителите достигат до страницата, в която се намира менюто за вход в системата, която ще бъде разпределена на три нива на достъп обикновен потребител, който има достъп до преглеждане на информацията в платформата, експерт, който ще има за цел да докладва чрез прикачване или добавяне на информация за годишните планове за използване на дървесина в определени горски или ловни стопанства в страната.

Също така експертите ще имат достъп до редакцията на новините или информация свързана с техническото управление на платформата. Администраторът е висшата роля в платформата, и чрез тези привилегии има достъп до всички части на програмата

Данните се обработват чрез потвърждаване след прикачване на утвърдени от регионалните органи планове за реализиране на сечи. Данните ще се запазват в база от данни.

Изходът от програмата съдържа регулационни препоръки за нормализирането на реализираната дървесина в сектора на дърводобива, както и следене и анализиране на обезлесяването в страната. Всеки потребител ще има достъп до крайната информация.

Graphical user interface

Description automatically generated

Фигура 4

## Примерен потребителски интерфейс

Фигура 5

В системата има навигация, с помощта, на която се достъпват различните видове страници, от които са налични: Главна страница, Страница за новини/събития, Страница за вход в системата, Страница за различните горски институции, Страница за регулации както и контактна форма, чрез, която потребителите ще могат да се свързват с експертите при констатация за грешка или запитване.

Горната част на примерния интерфейс съдържа лента с информация за контакти и търсачка, след това е поставено логото на приложението и навигационно меню, което позволява на потребителя да се придвижва към различни функционалности на приложението. В този случай, те могат да изберат между различните страници.

Главната страница разполага с начален надпис и бутон за вход в системата. Бутона за вход в системата е предназначен за оторизирани лица те имат възможност да добавят, редактират и изтриват информация. Този бутон пренасочва потребителите към формата за вход в системата. Ако даден потребител няма съществуващ профил, той има възможност да си направи регистрация.

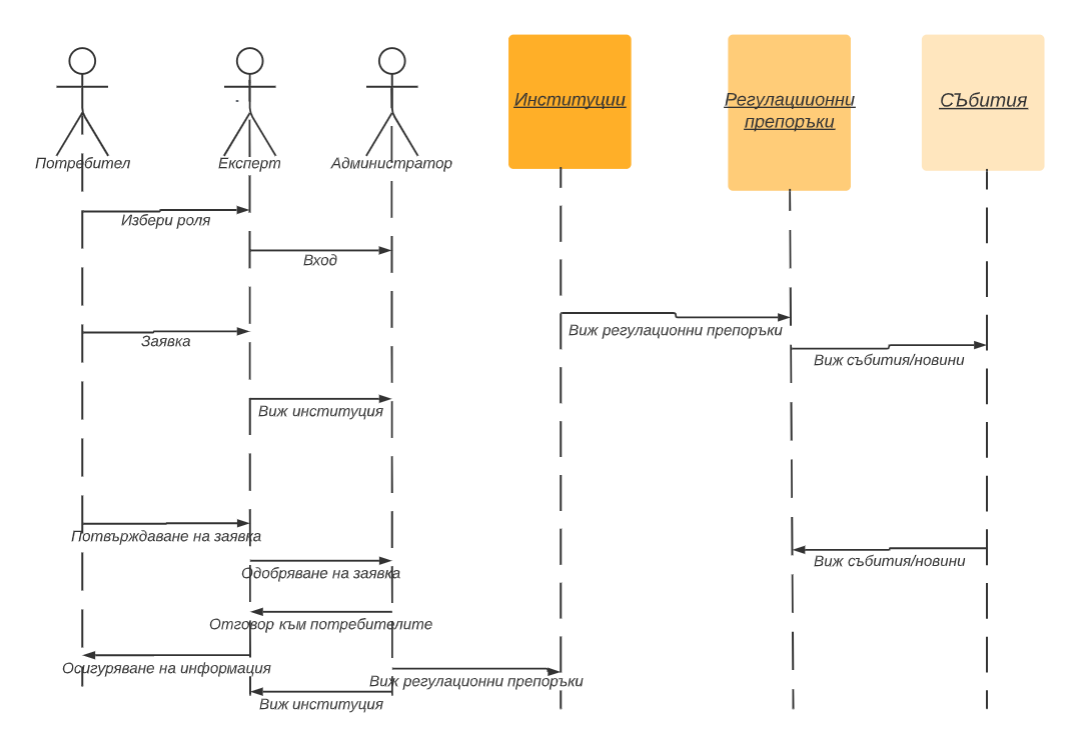
При кликване върху страницата с институции, потребителят може да види списък с всички горски институции. В допълнение, интерфейсът показва бутон пред името на всяка институция, който насочва към персонализирана страница с подробна информация за конкретната институция.

В страницата за новини и събития, се проектира календарна контрола на редица дейности свързани с горските стопанства вписани в системата.

Примерният потребителски интерфейс на система за регулации на дървесината и може да се промени в зависимост от нуждите и изискванията на потребителите. Интерфейсът цели да бъде лесен за използване, интуитивен и да предоставя на потребителите нужната функционалност.

## Диаграми на анализа

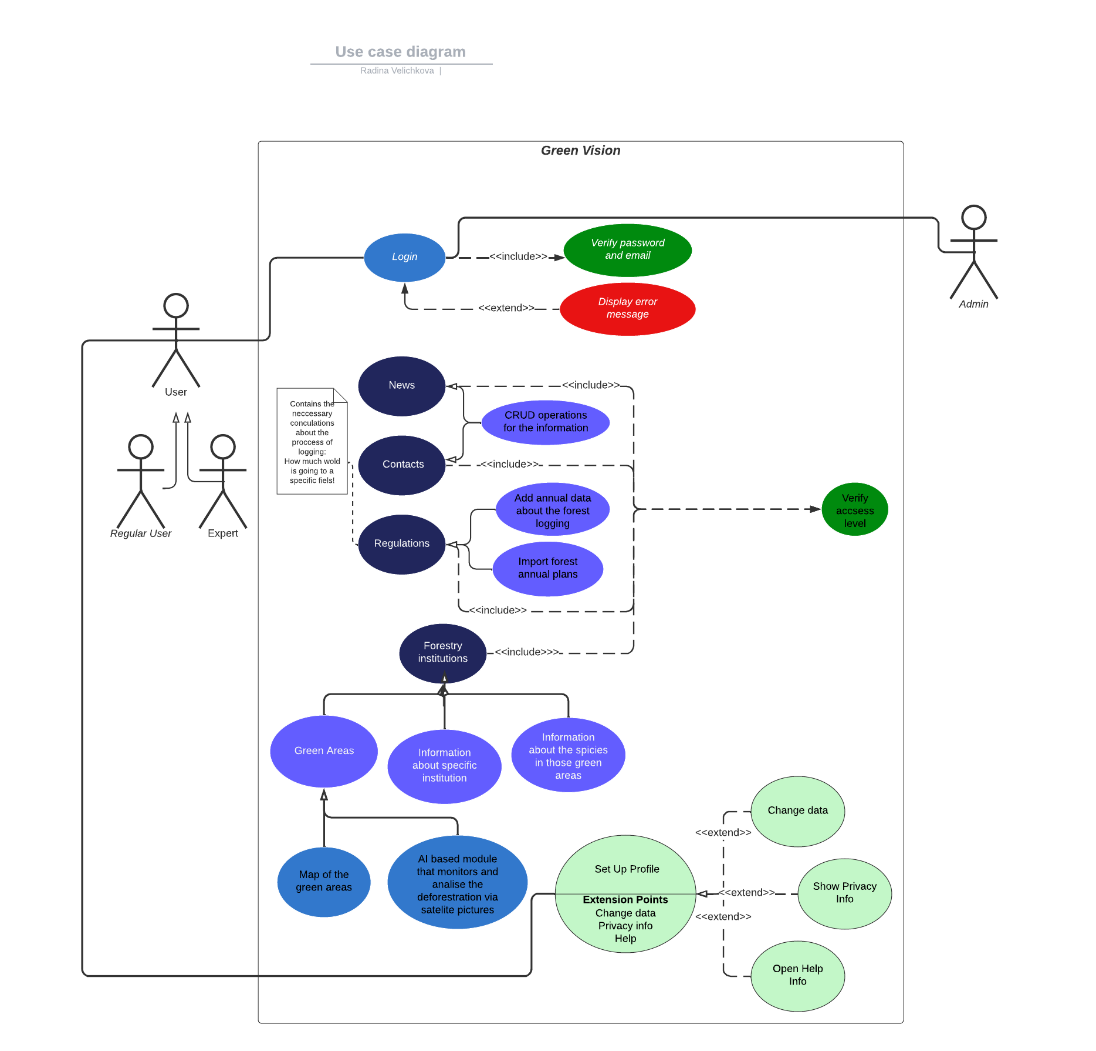
Диаграмата на последователността е представена на фигура 6 и показва взаимодействията между различните участници в уеб базираната система за регулиране на добива на дървесина, включително експерт, потребител и администратор.



Фигура 6

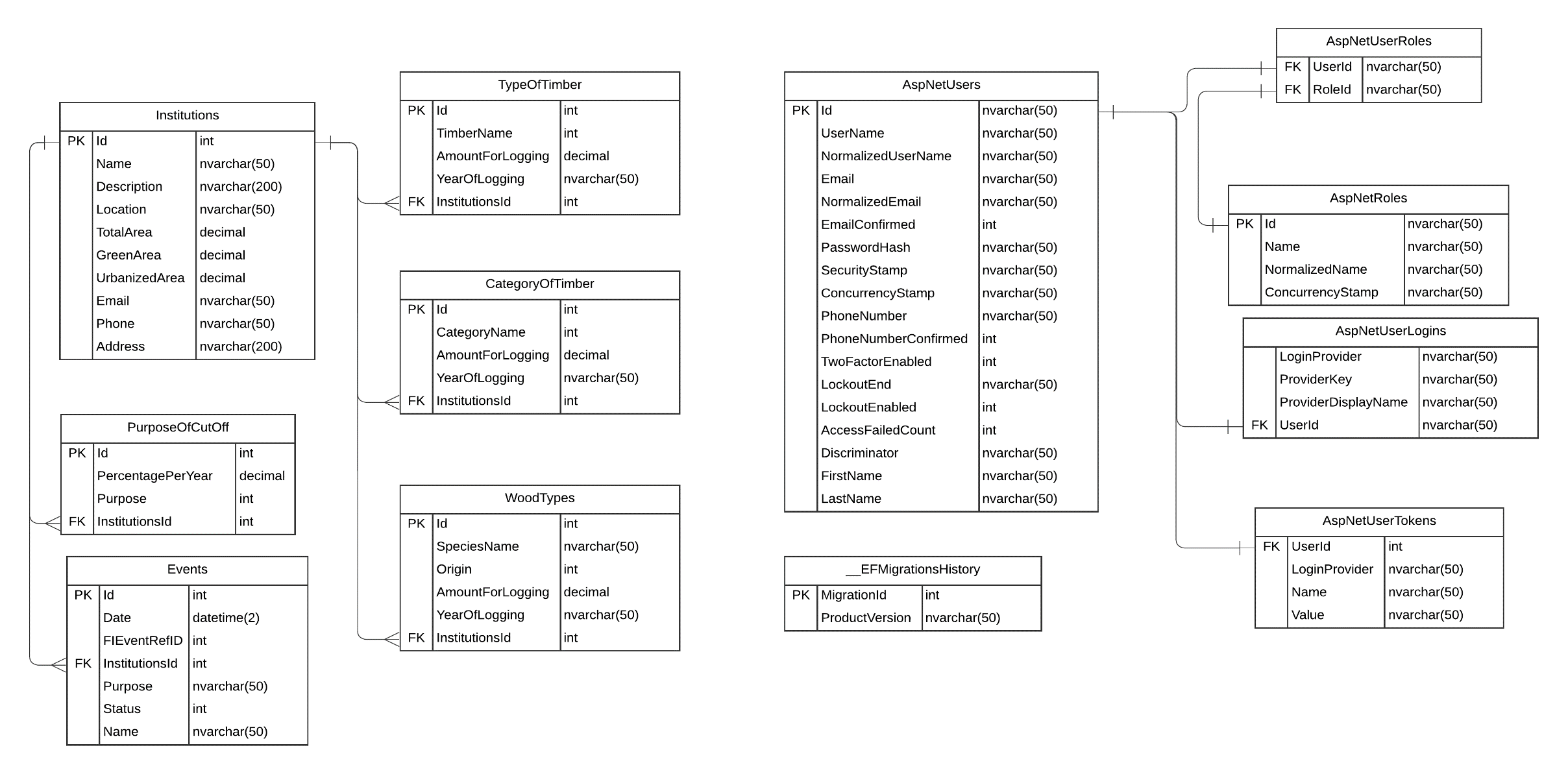
Фигура 7

Диаграмата на дейностите е представена на фигура визуализира поведението на системата идентифицирането на потенциални затруднения или неефективност.



Фигура 8

## 



Фигура 9

## Модел на данните

Базата данни, използвана за съхраняване на данните на проекта, е организирана, както е показано на фигура 9. Към базата данни се прилага трета нормална форма. Нека накратко обясним защо нормализацията в база данни е нещо, към което се стремим. Разминаванията при различните записи наричаме аномалии те възникват при вмъкване или модифициране на ненормализирани данни поради множество повторения и тавтологии, които правят обектите трудни за манипулиране. В резултат на това са стандартизирани много конструктивни нормални форми. При разработката на проекта се цели постигането на трета форма на нормализация. Тъй като нормализирането е постепенен процес, трябва да покажем, че предпоставките за всички предишни нива също са изпълнени, за да се установи степента на нормализиране. Предпоставките за първата нормална форма са да няма повтарящи се колони или редове и всеки атрибут да съдържа само неделими единични стойности.

Според втората нормална форма, всеки неключов атрибут трябва да бъде зависим от пълния първичен ключ. Третата нормална форма разширява втората, като задължава първичния ключ да бъде единственият фактор, който влияе на всички неключови свойства. [3] Това на практика изключва наличието на транзитивни зависимости. Описанието на всяка една таблица от базата данни и формата на нейната нормализация са както следва:

Таблица "*AspNetUserTokens*": тази таблица съдържа полета за потребителите на приложението. Има връзка към таблицата "*AspNetUsers*" чрез колоната "*UserId*". Всички колони са атомарни и не съществува функционална зависимост между тях. Това доказва, че таблицата е в трета нормална форма.

Таблица "*AspNetRoles*": тази таблица съдържа ролите на потребителите в приложението. Има само един основен ключ - колоната "Id". Всички колони са атомарни и не съществува функционална зависимост между тях. Следователно таблицата е в трета нормална форма.

Таблица "*AspNetUserLogins*": тази таблица съхранява информация за потребителите в системата за управление на идентичности на *Microsoft ASP.NET*. Тя има четири колони: *LoginProvider*, *ProviderKey*, *ProviderDisplayName* и *UserId*. Тази таблица не е в трета нормална форма, защото колоните *ProviderDisplayName* и *UserId* са функционално зависими от външния ключ, образуван от колоните *LoginProvider* и *ProviderKey*.

Таблица "*AspNetUsers*": тази таблица съхранява информация за потребителите в системата за управление на идентичности на *Microsoft ASP.NET*. Тя има много колони, но само някои от тях са необходими за описание на връзките. Първата колона „*Id*“ е първичен ключ (*PK)*. Тази таблица е в трета нормална форма, защото няма повтарящи се групи от данни и всички данни са атомарни. Всички колони са функционално зависими от първичния ключ (*PK*).

Таблица "*AspNetUserRoles*" има две колони - "*UserId*" и "*RoleId*", като всеки запис отговаря на връзка между определен потребител и роля в системата. Тази таблица е във втора нормална форма (*2NF*), тъй като няма частично зависими атрибути.

Таблица "*Institutions*": тази таблица съдържа информация за институциите, които използват приложението. Таблицата има само един първичен ключ - колоната "*Id*". Всички колони са атомарни и не съществува функционална зависимост между тях, това доказва наличието на трета форма на нормализация.

Таблица "*WoodTypes*": тази таблица съдържа информация за различните видове дървесина, които институциите могат да вписват. Таблицата има връзка към таблицата "*Institutions*" чрез колоната "*InstitutionsId*". Всички колони са атомарни и зависят само от първичния ключ "*Id*" и колоната "*InstitutionsId*", това доказва наличието на трета форма на нормализация.

Таблица "*CategoryOfTimber*": тази таблица съдържа информация за различните категории дървесина, които институциите могат да вписват. Таблицата има връзка към таблицата "*Institutions*" чрез колоната "*InstitutionsId*". Всички колони са атомарни и зависят само от основния ключ "*Id*" и колоната "*InstitutionsId*". Следователно таблицата е в трета нормална форма.

Таблицата "*TypeOfTimber*": тази таблица има пет колони - "*Id*", "*TimberName*", "*AmountForLogging*", "*YearOfLogging*" и "*InstitutionsId*". Първичният ключ е в колоната "*Id*". "*TimberName*" съхранява името на видът дървесина, "*AmountForLogging*" съхранява количеството на дървесината, която може да се изкористи, "*YearOfLogging*" показва годината на сечта на дървесината, а "*InstitutionsId*" е външен ключ, който указва към таблицата "*Institutions*". Тази таблица е в трета нормална форма (*3NF*), тъй като няма транзитивни функционални зависимости. Всички нейни атрибути са атомарни и не зависят от други неключови атрибути.

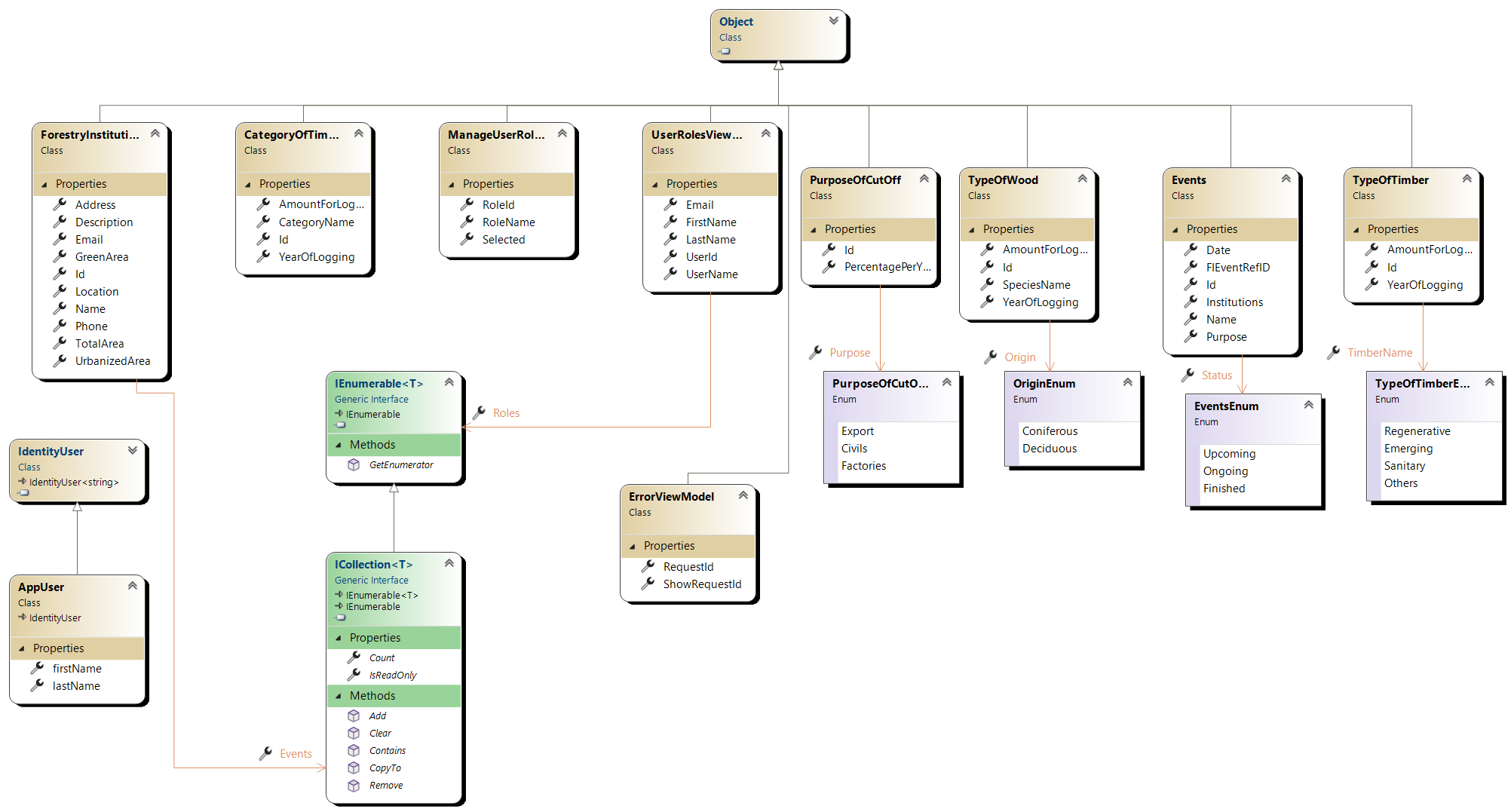
Таблица "PurposeOfCutOff": тази таблица съхранява информация за целите на изрязване на дървесина. Тя има четири колони: *Id*, *PercentagePerYear*, *Purpose* и *InstitutionsId*. Първата колона е първичен ключ „PK“. Колоната InstitutionsId е свързана с таблицата "Institutions". Тази таблица е в трета нормална форма (*3NF*), тъй като няма транзитивни функционални зависимости.

Таблица *"\_\_EFMigrationsHistory":* тази таблица съдържа историята на миграциите на базата данни. Има само един основен ключ - колоната "*MigrationId*". Всички колони са атомарни и не съществува функционална зависимост между тях, това доказва наличието на трета форма на нормализация.

При наличие на нормализация във всяка една от посочените таблици в базата данни следва, че базата данни е приведена в трета нормална форма.

## Модел на съдържанието

Проектът включва няколко разработки на класове, които се използват за дефиниране на моделите на данни за приложението. контролерите и изгледите. При класовете на методите се наблюдава принципът на енкапсулация се прилага във всички класове, като данните са скрити вътрешно в класовете и могат да бъдат достъпвани само чрез методи или свойства. Полиморфизмът се използва в няколко от класовете, като например в класа „*Events“*, където се използва свойството Status от изброим тип „*EventsEnum“ (IEnumerable)*. Това позволява да се използва една и съща променлива от различни типове, което улеснява работата със системата.



Фигура 10

Класове на моделите посочени на фигура 10:

При дефинирането на класа *"AppUser"* се използва един от главните принципи на Обектно Ориентираното програмиране, а именно наследяването. Класът е наследник на класа *"IdentityUser"* от библиотеката *"Microsoft.AspNetCore.Identity"*. Това позволява да се използват всички свойства и методи на класа „*IdentityUser“* в класа „*AppUser“* Този клас представлява потребител в системата и има две свойства - *"firstName"* и *"lastName",* които пазят съответно първото и последното име на потребителя.

Класът "*CategoryOfTimber*" представлява категория от дървен материал. Има свойства, които пазят името на категорията ("*CategoryName*"), количеството за изсичане ("*AmountForLogging*") и годината на изсичане ("*YearOfLogging*").

Класът "*ErrorViewModel*" се използва за показване на грешки в приложението. Съдържа свойства, които пазят идентификатора на заявката ("*RequestId*") и флаг, който определя дали да се показва идентификаторът на заявката ("*ShowRequestId*").

Класът "*Events*" представлява събитие в горското стопанство. Съдържа свойства, които пазят името на събитието ("*Name*"), датата на провеждане ("*Date*"), статуса на събитието ("*Status*"), целта на събитието ("*Purpose*") и препратка към институцията, която организира събитието ("*Institutions*"). Този клас съдържа и препратка към класа "*ForestryInstitution*".

Класът "*ForestryInstitution*" представлява горска институция. Съдържа свойства, които пазят името на институцията ("*Name*"), описание ("*Description*"), местоположение ("*Location*"), обща площ ("*TotalArea*"), зелена площ ("*GreenArea*"), урбанизирана площ ("*UrbanizedArea*"), имейл адрес ("*Email*"), телефонен номер ("*Phone*") и адрес ("*Address*"). Този клас има препратка към класа "*Events*".

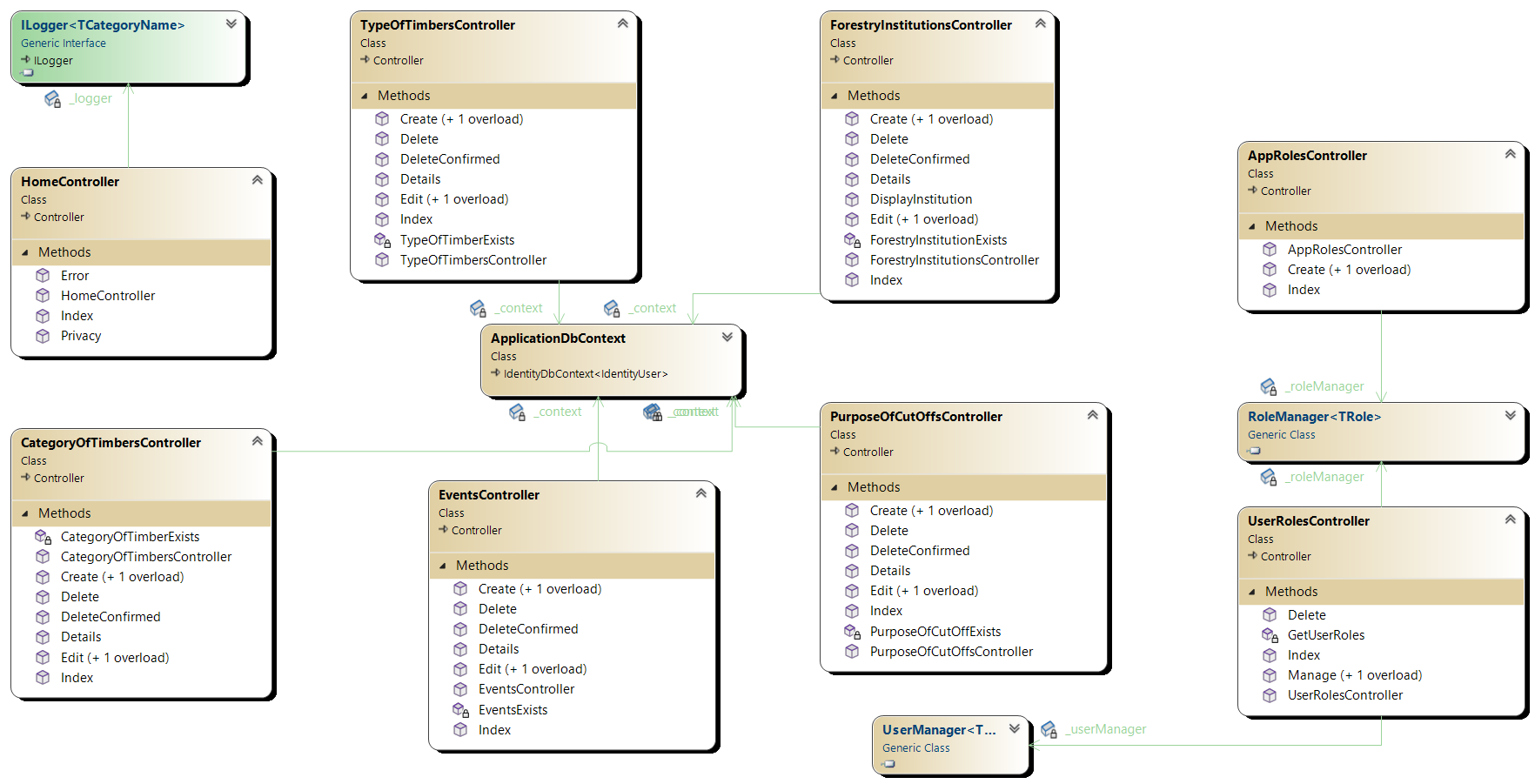
Класът "*ManageUserRolesViewModel*" се използва за управление на ролите на потребителите. Съдържа свойства, които пазят идентификатора на ролята ("*RoleId*"), името на ролята ("*RoleName*") и флаг, който определя дали ролята е избрана ("*Selected*").

Класът "*PurposeOfCutOff*" представлява целта на изсичане на дървесината. Съдържа свойства, които пазят изброимите типове (*IEnumerable*) на целта на изсичане ("*Purpose*"), като например "заради изграждане на пътища" или "заради изграждане на жилища", и "*PercentagePerYear*", който пази процента на изсичане за една година.

Класът "*TypeOfTimber*" представлява вид дървесина. Съдържа свойства, които пазят изброимите типове (*IEnumerable*) на вида на дървесината ("*TimberName*"), като например "дъб" или "бук", "*AmountForLogging*", който пази количеството на дървесината, подходящо за изсичане, и "*YearOfLogging*", който пази годината, в която дървесината е била изсечена.

Класът "*TypeOfWood*" представлява вид дърво. Съдържа свойства, които пазят името на вида дърво ("*SpeciesName*"), изброимите типове (*IEnumerable*) на произхода на дървото ("*Origin*"), като например "северноамерикански" или "европейски", "*AmountForLogging*", който пази количеството на дървото, подходящо за изсичане, и "*YearOfLogging*", който пази годината, в която дървото е било изсечено.

Класът "*UserRolesViewModel*" представлява модел на потребителските роли. Съдържа свойства, които пазят идентификатора на потребителя ("*UserId*"), първото име на потребителя ("*FirstName*"), фамилията на потребителя ("*LastName*"), потребителското име ("*UserName*"), имейл адреса ("*Email*") и списък от ролите, които потребителят има ("*Roles*").



Фигура 11

Класове на контролерите посочени на фигура 11:

В примерната диаграма се използва наследяване, което е основен принцип в обектно-ориентираното програмиране. Контролерите наследяват базов клас „*Controller“*, който предоставя редица функционалности за работа с уеб заявките. Описание на водещи класове представени в диаграмата:

В контролера „*AppRolesController“* се използва класът „*RoleManager<IdentityRole>“,* който предоставя функционалности за управление на потребителските роли в приложението. Той също така демонстрира полиморфизъм, като има два метода „*Create“*, но с различни параметри (единият приема модел, а другият - име на роля).

В контролера „*CategoryOfTimbersController“* се използва класът „*ApplicationDbContext“*, който е обвързан с базата данни и позволява достъп до записите в нея. Тук отново се използва наследяване от базов клас, като този път е „*ControllerBase“*. Освен това се използва и атрибута „*Authorize“*, който контролира достъпа до определени действия в контролера, в зависимост от потребителската роля.

Класът „*EventsController*“ използва инстанция на класа "*ApplicationDbContext*" за достъп до данните в базата данни, която се инжектира чрез конструктора му. Той е наследник на класа "*Controller*" от „*Microsoft.AspNetCore.Mvc*“, който му позволява да обработва *HTTP* заявки.

Класът „*ForestryInstitutionsController*“ използва полиморфизъм за определяне на типа на обектите, които получава като параметри в методите си, като например в метода *“Create()”.* Той също така използва наследяване чрез наследяването на основния клас “*Controller”*, който му предоставя функционалността на контролер.

Класът *„HomeController“* аналогично наследява класа *„Controller“*. Той има един конструктор, който инициализира член-променливата *\_logger* от тип интерфейс *„ILogger<HomeController>“.*

# Дизайн

В този проект е използвана технологията на уеб приложение *ASP.NET* като основна рамка за изграждане на уеб базираната система. Вградените възможности на *ASP.NET* за внедряване на модела *MVC* (*Model-View-Controller*) позволяват създаване на модулно и лесно поддържано приложение.

За разработката на главния изглед на системата са използвани технологиите *HTML* и *CSS*. Тези технологии позволяват изграждането на удобен за потребителя интерфейс, който е както визуално привлекателен, така и лесен за навигация. Също така бе използван езика за програмиране *JavaScript*, чрез който се бе добавена интерактивност в приложението и подобри потребителското изживяване.

За съхранение и управление на данни, бе използван софтуера *SQLite*. Тази лека и интегрираща се система за управление на бази данни позволява лесно да се съхраняват и манипулират данни в приложението. *SQLite* е особено полезен за съхраняване и извличане на големи количества данни бързо и ефективно, което бе критично за успеха на проекта. Изборът за използване на тази функционалност под формата на разширение към средата за програмиране *VisualStudio*.[[1]](#footnote-2)

Комбинацията от *ASP.NET*, *MVC* архитектура, *HTML*, *CSS*, *JavaScript* и *SQLite* осигури стабилен и гъвкав технологичен стек за изграждане на модерно и бързо реагиращо уеб приложение. И използването на *Visual Studio* като интегрирана среда за разработка *(IDE),* правейки процеса на работа по-рационализиран и продуктивен.

## Реализация на архитектурата на приложението

Diagram

Description automatically generatedЗа разработването на платформата е използвана архитектурата *MVC (Model-View-Controller) фигура 12.* Архитектурата е разделена на три основни компонента: модел, изглед и контролер. *MVC* архитектурата позволява работата върху различни части на приложението независима, което води до по-ефективен процес на разработка.

Фигура 12

Уеб приложението *ASP.NET* използва различни технологии, за да осигури функционалност на *MVC* архитектурата. Например *Entity* Framework се използва за манипулиране на данните в модела, *Razor* се използва за създаване на изгледи, инфраструктурата на *ASP.NET MVC* *Framework* осигурява контролерите и координира взаимодействието между тях. Тези технологии са интегрирани в средата за разработка на *Visual Studio* и могат да бъдат използвани за създаване на мощни уеб приложения с *MVC* архитектура. Тя осигурява разделяне на отговорностите в приложението, което прави по-лесно да се поддържа, разширява и тества. [4]

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

### Контролери

Контролерът действа като посредник между модела и изгледа. Той обработва въвеждането от потребителя, извлича или актуализира данни от модела и предава данните към изгледа за показване. В платформата са реализирани контролерите, използвайки класове, които обработват заявки и управляват потока от данни между модела и изгледа. Контролерите, предоставени в кода, отговарят за управлението на различни аспекти на уеб приложение *Forestry System*,реализацията на останалите контролери в приложението е аналогична на описаните. Кратко описание на три контролера:

13

**Контролер: UserRolesController**

using ForestrySystem.Models;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using System.Data;

namespace ForestrySystem.Controllers

{

[Authorize(Roles = "Admin")]

public class UserRolesController : Controller

{

private readonly UserManager<AppUser> \_userManager;

private readonly RoleManager<IdentityRole> \_roleManager;

public UserRolesController(UserManager<AppUser> userManager, RoleManager<IdentityRole> roleManager)

{

\_roleManager = roleManager;

\_userManager = userManager;

}

public async Task<IActionResult> Index()

{

var users = await \_userManager.Users.ToListAsync();

var userRolesViewModel = new List<UserRolesViewModel>();

foreach (AppUser user in users)

{

var thisViewModel = new UserRolesViewModel();

thisViewModel.UserId = user.Id;

thisViewModel.Email = user.Email;

thisViewModel.FirstName = user.firstName;

thisViewModel.LastName = user.lastName;

thisViewModel.Roles = await GetUserRoles(user);

userRolesViewModel.Add(thisViewModel);

}

return View(userRolesViewModel);

}

private async Task<List<string>> GetUserRoles(AppUser user)

{

return new List<string>(await \_userManager.GetRolesAsync(user));

}

public async Task<IActionResult> Manage(string userId)

{

ViewBag.userId = userId;

var user = await \_userManager.FindByIdAsync(userId);

if (user == null)

{

ViewBag.ErrorMessage = $"User with Id = {userId} cannot be found";

return View("NotFound");

}

ViewBag.UserName = user.UserName;

var model = new List<ManageUserRolesViewModel>();

foreach (var role in \_roleManager.Roles)

{

var userRolesViewModel = new ManageUserRolesViewModel

{

RoleId = role.Id,

RoleName = role.Name

};

if (await \_userManager.IsInRoleAsync(user, role.Name))

{

userRolesViewModel.Selected = true;

}

else

{

userRolesViewModel.Selected = false;

}

model.Add(userRolesViewModel);

}

return View(model);

}

[HttpPost]

public async Task<IActionResult> Manage(List<ManageUserRolesViewModel> model, string userId)

{

var user = await \_userManager.FindByIdAsync(userId);

if (user == null)

{

return View();

}

var roles = await \_userManager.GetRolesAsync(user);

var result = await \_userManager.RemoveFromRolesAsync(user, roles);

if (!result.Succeeded)

{

ModelState.AddModelError("", "Cannot remove user existing roles");

return View(model);

}

result = await \_userManager.AddToRolesAsync(user, model.Where(x => x.Selected).Select(y => y.RoleName));

if (!result.Succeeded)

{

ModelState.AddModelError("", "Cannot add selected roles to user");

return View(model);

}

return RedirectToAction("Index");

}

public async Task<IActionResult> Delete(List<ManageUserRolesViewModel> model, string Id)

{

var UserToDelete = await \_userManager.FindByIdAsync(Id);

if (UserToDelete != null)

{

IdentityResult result = await \_userManager.DeleteAsync(UserToDelete);

if (result.Succeeded)

{

return View(model);

}

}

return RedirectToAction("Index");

}

}

}

**Описание:** Този контролер управлява управлението на ролите за потребителите. Има методи за показване на списък с всички потребители и техните роли, управление на ролите на конкретен потребител и изтриване на потребител.

**Цел:** Целта на този контролер е да позволи на оторизираните потребители да управляват ролите на други потребители в системата.

**Библиотеки и методи:**

Използване на *ForestrySystem.Models:* Тази библиотека включва моделите *AppUser* и *UserRolesViewModel*, които се използват в този контролер.

Използване на *Microsoft.AspNetCore.Identity*: Тази библиотека включва класовете IdentityRole и UserManager, които се използват за управление на роли и потребители в системата.

Използване на *Microsoft.AspNetCore.Mvc*: Тази библиотека включва класа Controller, от който този клас наследява, както и други класове и методи за обработка на HTTP заявки и отговори.

Използване на *Microsoft.EntityFrameworkCore*: Тази библиотека включва класовете и методите на *Entity Framework Core* за взаимодействие с базата данни.

Методи:

* *Index():* Показва списък с всички потребители и техните роли.
* *GetUserRoles():* Помощен метод, който връща списък с ролите, към които принадлежи даден потребител.
* *Manage():* Показва страница за управление на ролите на конкретен потребител.
* *Manage(List<ManageUserRolesViewModel>* model, string userId): Обработва HTTP POST заявката за управление на ролите на конкретен потребител.
* *Delete(List<ManageUserRolesViewModel>* model, string Id): Обработва HTTP POST заявката за изтриване на потребител. [5]

**Контролер: AppRolesController**

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

namespace ForestrySystem.Controllers

{

[Authorize(Roles = "Admin")]

public class AppRolesController : Controller

{

private readonly RoleManager<IdentityRole> \_roleManager;

public AppRolesController(RoleManager<IdentityRole> roleManager)

{

\_roleManager = roleManager;

}

public IActionResult Index()

{

var roles = \_roleManager.Roles;

return View(roles);

}

[HttpGet]

public IActionResult Create()

{

return View();

}

[HttpPost]

public async Task<IActionResult> Create(IdentityRole model)

{

if (!\_roleManager.RoleExistsAsync(model.Name).GetAwaiter().GetResult())

{

\_roleManager.CreateAsync(new IdentityRole(model.Name)).GetAwaiter().GetResult();

}

return RedirectToAction("Index");

}

}

}

**Описание:** Този контролер е отговорен за управлението на ролите в приложението, особено ролите, които имат разрешение „Администратор“.

**Библиотеки и методи:** Контролерът използва библиотеките *Microsoft.AspNetCore.Authorization*, *Microsoft.AspNetCore.Identity* и *Microsoft.AspNetCore.Mvc*.

Методи:

* *Index():* Връща изглед, който показва всички роли в приложението.
* *Create():* Връща изглед, който позволява създаването на нова роля.
* *Create(IdentityRole model):* Създава нова роля с предоставения обект *IdentityRole* и пренасочва към метода *Index().*

**Контролер: CategoryOfTimbersController**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using ForestrySystem.Data;

using ForestrySystem.Models;

using Microsoft.AspNetCore.Authorization;

namespace ForestrySystem.Controllers

{

public class CategoryOfTimbersController : Controller

{

private readonly ApplicationDbContext \_context;

public CategoryOfTimbersController(ApplicationDbContext context)

{

\_context = context;

}

public async Task<IActionResult> Index(string SearchString)

{

ViewData["CurrentFilter"] = SearchString;

var categs = from cats in \_context.CategoryOfTimber

select cats;

if (!String.IsNullOrEmpty(SearchString))

{

categs = categs.Where(x => x.CategoryName.ToString().Contains(SearchString));

}

return View(categs);

}

public async Task<IActionResult> Details(int? id)

{

if (id == null || \_context.CategoryOfTimber == null)

{

return NotFound();

}

var categoryOfTimber = await \_context.CategoryOfTimber

.FirstOrDefaultAsync(m => m.Id == id);

if (categoryOfTimber == null)

{

return NotFound();

}

return View(categoryOfTimber);

}

[Authorize(Roles = "Expert")]

public IActionResult Create()

{

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

[Authorize(Roles = "Expert")]

public async Task<IActionResult> Create([Bind("Id,CategoryName,AmountForLogging,YearOfLogging")] CategoryOfTimber categoryOfTimber)

{

if (ModelState.IsValid)

{

\_context.Add(categoryOfTimber);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(categoryOfTimber);

}

[Authorize(Roles = "Expert")]

public async Task<IActionResult> Edit(int? id)

{

if (id == null || \_context.CategoryOfTimber == null)

{

return NotFound();

}

var categoryOfTimber = await \_context.CategoryOfTimber.FindAsync(id);

if (categoryOfTimber == null)

{

return NotFound();

}

return View(categoryOfTimber);

}

[HttpPost]

[ValidateAntiForgeryToken]

[Authorize(Roles = "Expert")]

public async Task<IActionResult> Edit(int id, [Bind("Id,CategoryName,AmountForLogging,YearOfLogging")] CategoryOfTimber categoryOfTimber)

{

if (id != categoryOfTimber.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(categoryOfTimber);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!CategoryOfTimberExists(categoryOfTimber.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

return View(categoryOfTimber);

}

[Authorize(Roles = "Expert")]

public async Task<IActionResult> Delete(int? id)

{

if (id == null || \_context.CategoryOfTimber == null)

{

return NotFound();

}

var categoryOfTimber = await \_context.CategoryOfTimber

.FirstOrDefaultAsync(m => m.Id == id);

if (categoryOfTimber == null)

{

return NotFound();

}

return View(categoryOfTimber);

}

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

if (\_context.CategoryOfTimber == null)

{

return Problem("Entity set 'ApplicationDbContext.CategoryOfTimber' is null.");

}

var categoryOfTimber = await \_context.CategoryOfTimber.FindAsync(id);

if (categoryOfTimber != null)

{

\_context.CategoryOfTimber.Remove(categoryOfTimber);

}

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool CategoryOfTimberExists(int id)

{

return \_context.CategoryOfTimber.Any(e => e.Id == id);

}

}

}

**Описание:** Този контролер е отговорен за управлението на категорията дървен материал, включително създаването, редактирането и изтриването им.

**Библиотеки и методи:** Контролерът използва библиотеките *Microsoft.AspNetCore.Mvc,* *Microsoft.EntityFrameworkCore* и *ForestrySystem*.

Методи:

* *Index():* Връща изглед, който показва цялата категория дървен материал в приложението.
* *Details (int? id):* Връща изглед, който показва подробностите за определена категория дървен материал с предоставения id.
* *Create():* Връща изглед, който позволява създаването на нова категория дървен материал.
* *Create([Bind("Id,CategoryName,AmountForLogging,YearOfLogging")] CategoryOfTimber categoryOfTimber):* Създава нова категория дървен материал с предоставения обект *CategoryOfTimber* и пренасочва към метода *Index().*
* *Edit(int? id):* Връща изглед, който позволява редактиране на определена категория дървен материал с предоставения id.
* *Edit(int id, [Bind("Id,CategoryName,AmountForLogging,YearOfLogging")] CategoryOfTimber categoryOfTimber):* Актуализира определена категория дървен материал с предоставения идентификатор с обекта *CategoryOfTimber* и пренасочва към метода *Index().*
* *Delete(int? id):* Връща изглед, който позволява изтриване на определена категория дървен материал с предоставения *id*.
* *DeleteConfirmed(int id):* Изтрива конкретна категория дървен материал с предоставения id и пренасочва към метода *Index().*

### Изгледи

Изгледът представлява потребителския интерфейс и е отговорен за показването на данните на модела на потребителя. Той се реализира с помощта на *HTML*, *CSS* и *JavaScript* и може също да използва библиотеки като *jQuery* или *Angular*. Методите, включени във *Views* (Изгледите), могат да варират в зависимост от конкретните библиотеки и рамки, които се използват, но обикновено включват функции за манипулиране на *DOM*, обработка на потребителски вход и взаимодействие с данни от страна на сървъра.

@using Microsoft.AspNetCore.Identity

@model IdentityRole

<div class="row">

<div class="col-md-4">

<**form** method="post">

<h2>Create Role</h2>

<hr/>

<**div** **asp-validation-summary**="ModelOnly" class="text-danger"></**div**>

<div class="form-floating">

<**input** **asp-for**="Name" class="form-control" aria-required="true" />

<**label** **asp-for**="Name"></**label**>

<**span** **asp-validation-for**="Name" class="text-danger"></**span**>

</div>

<button type ="submit" class="btn btn-primary">Save</button>

</**form**>

</div

</div>

В дадения шаблон изгледът отговаря за показването на формуляра за създаване на нова *IdentityRole*. Той използва *ASP.NET Core Tag Helpers* за генериране на *HTML* за формуляра и обработва подаването на формуляр с помощта на *POST* заявка. Изгледът включва библиотеки като *Bootstrap* за стилизиране и валидиране и *Razor* за изобразяване на *HTML*.

@model ForestrySystem.Models.CategoryOfTimber

@{

ViewData["Title"] = "Edit";

}

<h1>Edit</h1>

<h4>CategoryOfTimber</h4>

<hr />

<div class="row">

<div class="col-md-4">

<**form** **asp-action**="Edit">

<**div** **asp-validation-summary**="ModelOnly" class="text-danger"></**div**>

<**input** **type**="hidden" **asp-for**="Id" />

<div class="form-group">

<**label** **asp-for**="CategoryName" class="control-label"></**label**>

<**select** **asp-for**="CategoryName" class="form-control" **asp-items**="Html.GetEnumSelectList<CategoryOfTimberEnum>()"></**select**>

<**span** **asp-validation-for**="CategoryName" class="text-danger"></**span**>

</div>

<div class="form-group">

<**label** **asp-for**="AmountForLogging" class="control-label"></**label**>

<**input** **asp-for**="AmountForLogging" class="form-control" />

<**span** **asp-validation-for**="AmountForLogging" class="text-danger"></**span**>

</div>

<div class="form-group">

<**label** **asp-for**="YearOfLogging" class="control-label"></**label**>

<**input** **asp-for**="YearOfLogging" class="form-control" />

<**span** **asp-validation-for**="YearOfLogging" class="text-danger"></**span**>

</div>

<div class="form-group">

<input type="submit" value="Save" class="btn btn-primary" />

</div>

</**form**>

</div>

</div>

<div>

<**a** **asp-action**="Index">Back to List</**a**>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

}14

Изгледът в предоставения примерен код е написан в синтаксиса на Razor и е строго въведен с директивата *"@model",* която указва, че очаква екземпляр на модела *"CategoryOfTimber"* от пространството на имената *"ForestrySystem.Models".* Той показва формуляр за редактиране на свойствата на екземпляр *"CategoryOfTimber",* включително падащ списък с възможни стойности на *"CategoryName",* въвеждане на текст за *"AmountForLogging"* и въвеждане на текст за *"YearOfLogging".* Той също така включва валидиране от страна на клиента с помощта на вградената система за валидиране *ASP.NET Core.* За да се използва този изглед в приложението на *ASP.NET Core*, машината за преглед на *Razor* се използва за изобразяване на изгледа и генериране на *HTML*, който ще бъде изпратен до браузъра на потребителя. Изгледът може да бъде достъпен чрез метод за действие на контролера, като се използва методът *"View()",* като се подават съответните данни за модела.

### Модел

Моделите в *MVC* са класове, които представляват данните и бизнес логиката на дадено приложение. Тяхната цел е да дефинират структурата и поведението на данните на приложението и да осигурят последователен начин за достъп и манипулиране на тези данни. Моделите могат да включват свойства, методи, правила за валидиране и връзки с други модели.

using ForestrySystem.Enums;

using System.ComponentModel;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace ForestrySystem.Models

{

public class Events

{

[Key]

public int Id { get; set; }

[DisplayName("Date/Time")]

public string Name { get; set; }

public DateTime Date { get; set; }

public EventsEnum Status { get; set; }

public string Purpose { get; set; }

[ForeignKey("ForestryInstitution")]

[DisplayName("Institution ID")]

public int FIEventRefID { get; set; }

public ForestryInstitution? Institutions { get; set; }

}

}

Предоставеният шаблон е пример за модел в приложението *ForestrySystem*, който се използва за управление на горски институции и дейности по дърводобив. Класът *Events* дефинира свойства за датата, състоянието и целта на горските събития, както и препратка към външен ключ към модела *ForestryInstitution*. Класовете *EventsEnum* и *ForestryInstitution* също се използват в приложението и са дефинирани в отделни файлове. Библиотеките *System.ComponentModel*, *System.ComponentModel.DataAnnotations* и *System.ComponentModel.DataAnnotations.Schema* се използват в примерния модел, за да осигурят характеристики като показвани имена за свойства, валидиране на данни и връзки с външни ключове. Тези библиотеки обикновено се използват в *MVC* приложения за добавяне на функционалност към моделите.

### Изброими типове данни (IEnumerable)

Изброимите типове данни (IEnumerable) в ASP.NET проектите представляват множество константи, които могат да бъдат използвани като стойности на променливи. Изброимите типове данни помагат за по-добро управление на данните и избягване на грешки при задаване на стойности на променливи, като се гарантира, че използваните стойности са само тези, които са дефинирани в типа (IEnumerable).

В проекта са реализирани следните изброими типове данни:

*CategoryOfTimberEnum* - използва се за дефиниране на различни категории на дървесина, които могат да бъдат използвани в системата за горски стопански дейности.

*EventsEnum* - използва се за дефиниране на статусите на различни събития, свързани с горското стопанство, като се определя дали те са предстоящи, в процес на провеждане или вече приключили.

*OriginEnum* - използва се за дефиниране на произхода на дървесината, като може да бъде иглолистна или широколистна.

*PurposeOfCutOffsEnum* - използва се за дефиниране на различните цели на отсичане на дървесина, като може да бъде за износ, за строителство или за производство.

*RoleEnum* - използва се за дефиниране на различните роли на потребителите в системата, като могат да бъдат потребител, експерт или администратор.

*TypeOfTimberEnum* - използва се за дефиниране на различни типове на дървесина, като може да бъде регенеративна, изкачваща се, санитарна или други.

Тези изброими типове данни се използват в проекта за задаване на стойности на променливи, които да отговарят на дефинираните константи в изброимите типове данни, както и за лесно управление на тези стойности в кода. Например, в таблицата "*CategoryOfTimber*" се използва „*CategoryOfTimberEnum“*.

## Организация и код на заявките към база от данни

Инструментариумът за достъп до базата данни е предоставен чрез класа *ApplicationDbContext*. Той наследява *IdentityDbContext<IdentityUser>* и използва библиотеката *Microsoft.EntityFrameworkCore* за достъп до базата данни. В класа са дефинирани различни свойства за достъп до таблиците в базата данни, които се репрезентират чрез *DbSet*. Тези свойства включват *Institutions, WoodTypes, PurposeOfCutOff, TypeOfTimber, Events, CategoryOfTimber и Users.*

using ForestrySystem.Models;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore;

namespace ForestrySystem.Data

{

public class ApplicationDbContext: IdentityDbContext<IdentityUser>

{

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> option) : base(option){

}

public DbSet<ForestryInstitution> Institutions { get; set; }

public DbSet<TypeOfWood> WoodTypes { get; set; }

public DbSet<ForestrySystem.Models.PurposeOfCutOff> PurposeOfCutOff { get; set; }

public DbSet<ForestrySystem.Models.TypeOfTimber> TypeOfTimber { get; set; }

public DbSet<ForestrySystem.Models.Events> Events { get; set; }

public DbSet<ForestrySystem.Models.CategoryOfTimber> CategoryOfTimber { get; set; }

public DbSet<AppUser> Users { get; set; }

}

}

За извличане на обекти от базата данни, могат да се използват методите *FirstOrDefaultAsync(), ToListAsync(), FindAsync()* и други методи, предоставени от Microsoft.EntityFrameworkCore. За добавяне на нови обекти в базата данни, може да се използва методът *Add()*, а за изтриване на обекти - *Remove().*

За управление на данните във *ApplicationDbContext* се използва *Linq*, което позволява заявки за извличане на данни по определени критерии и сортиране на резултатите. *Collections Generic* също е включен в инструментариума и предоставя удобен начин за манипулиране на колекции от обекти.

Технологията *Threading.Tasks* е включена за поддръжка на асинхронно програмиране и по-ефективна обработка на заявки към базата данни. Това може да доведе до по-бързи отговори на заявките от страна на потребителите на приложението.

В цялост, инструментариумът предоставя удобни и ефективни методи за достъп до базата данни *SQLite*, като същевременно осигурява сигурност и издръжливост на данните.

## Ефективност и бързодействие на алгоритмичното решение

За реализация на регулационните препоръки е реализиран алгоритъма Дърво на решенията (Decision tree). Като алгоритъм за контролирано обучение, дърветата на решенията са най-добрият избор за класифициране както на категорични, така и на непрекъснати зависими променливи. В този алгоритъм общото се разделя на два или повече сходни набора от данни, разчитайки на най-значимите характеристики или независими променливи.

Нотациите на дървото на решенията спадат към O(n). Възлите на дървото са представени от кръгове, като всеки връх съответства на едно решение. Върхът има два или повече наследника, които са представени от допълнителни кръгове. Ребрата на дървото са маркирани с атрибути и стойности на тези атрибути, които се използват за вземане на решение.

Ефективността на алгоритъма "Decision tree" зависи от няколко фактора, като съотношението между броя на примерите в обучителния набор и броя на атрибутите, както и от разпределението на класовете. Ако обучителният набор има много атрибути в сравнение с броя на примерите, дървото на решенията може да стане твърде голямо и да се претовари. С други думи, моделът може да стане прекалено сложен и да запомни детайли, които не са от значение.

Бързодействието на дървото на решенията е добро в сравнение с някои други методи за машинно обучение, особено при класификационни проблеми. Въпреки това, при решаване на задачи с много атрибути, дървото на решенията може да се окаже по-бавно от други методи. [6] [7]

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

Потребителският интерфейс на уеб базираната система за регулация на дървесината е проектиран така, че да бъде лесен за използване и да предоставя на потребителите достъп до информацията, отнасяща се до регулациите за дървесината, кратка информация за горските институции, както и информация за новини и събития на тяхната територия. A forest of trees

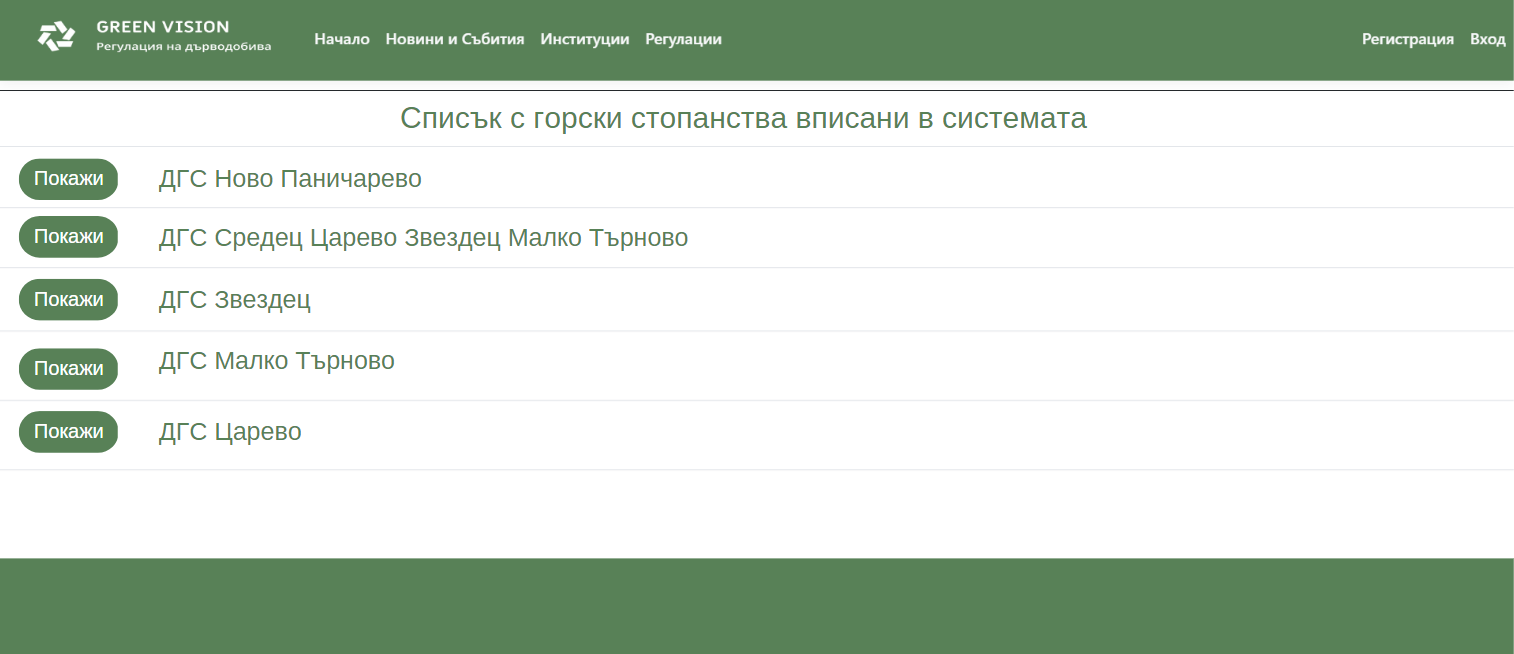
Description automatically generated with low confidence

Фигура 15

В публичната част на приложението има навигация, която е достъпна във всеки момент. С нейна помощ се достъпват различните видове страници, от които са налични: Главна страница, Страница за новини/събития, Страница за вход в системата, Страница за различните горски институции, Страница за регулации както и контактна форма, чрез, която потребителите ще могат да се свързват с експертите при констатация за грешка или запитване.

Горната част на потребителския изглед съдържа лента с информация за контакти и търсачка, след това е поставено логото на приложението и навигационно меню, което позволява на потребителя да се придвижва към различни функционалности на приложението. В този случай, те могат да изберат между различните страници.

Главната страница разполага с начален надпис. Бутона за вход в системата се намира в горния десен ъгъл на менюто за навигация и е предназначен за потребителите с роля „Expert“ и “Admin“, в ролята на оторизирани лица те имат различен изглед на представените страници. Тези лица имат възможност да добавят, редактират и изтриват информация от системата, като всяка една от тези опции е изобразена в отделна форма за осъществяване на конкретната функция. Този бутон пренасочва потребителите към формата за вход в системата. Ако даден потребител няма съществуващ профил, той има възможност да си направи регистрация от страницата за регистрация, която е в ляво на страницата за вход в системата. Фигура 15.



Фигура 16

При кликване върху страницата с институции, потребителят може да види списък с всички горски институции. В допълнение, интерфейсът показва бутон пред името на всяка институция, който насочва към персонализирана страница с подробна информация за конкретната институция. Персонализираната страница разполага с карта, която показва границите на горското стопанство и възможност за извеждане на справки по години за реализираната дървесина на територията на конкретната институция. Фигура 16.

В персонализираната страница потребителят има възможност да направи различни справки по година на база категории визуализирани в падащо меню с избор.

В страницата за новини и събития, се проектира календарна контрола на редица дейности свързани с горските стопанства вписани в системата.

В страницата за регулации се изобразяват процентно по години различните препоръки, които се отнасят до разпределението на реализираната дървесина: за местно потребление, за износ, за производство.

Цветовата гама на потребителския интерфейс е с цел спокойна и професионална, съчетаваща се с тематичната област на проекта. Chart, bar chart

Description automatically generated

Фигура 17

На фигура 17 е изобразена цветовата схема на потребителския интерфейс, която е изработена с помощта на платформата *Coolors*.

# Тестване

В реалното изпълнение на проекта е препоръчително да се извършват различни видове тестване, включително функционално, системно, интеграционно, unit testing, performance testing и други. За целта в бъдеще може да се използват различни инструменти и браузъри за тестване на интерфейса на приложението.

Тестовите случаи са написани с помощта на *xUnit* и *FluentAssertions*, които улесняват писането на тестове. За изпълнението на тестовете се използва *in-memory database*, която позволява бързо и лесно тестване на приложението, без да се изисква инсталация на реалната база данни.

Тестовите случаи имат за цел да проверят функционалността на приложението. В този пример те са насочени към проверка на *CRUD* операциите (*create, read, update, delete*) в базата данни. Използван е тестов клас *CRUDTests*, който тества всички методи за *CRUD* в контролера *TypeOfTimbersController*.

**Примерен тест план за Unit Testing:**

**Тестване на добавянето на нов обект в базата данни**

Входни данни: Обект от клас *TypeOfTimber* с валидни стойности за *TimberName*, *AmountForLogging* и *YearOfLogging.*

Очакван резултат: Обектът да бъде успешно добавен в базата данни и броят на записите в таблицата *TypeOfTimber* да се увеличи с единица.

[Fact]

async void UnitTestCreate()

{

var TypeOfTimber = new TypeOfTimber()

{

TimberName = TypeOfTimberEnum.Sanitary,

AmountForLogging = 1,

YearOfLogging = DateTime.Now

};

var databaseContext = await GetDbContext();

var controller = new TypeOfTimbersController(databaseContext);

controller.Create(TypeOfTimber);

Assert.Equal(1, databaseContext.TypeOfTimber.Count());

}

**Тестване на извличането на обект от базата данни**

Входни данни: Идентификатор на обект от клас *TypeOfTimber*, който съществува в базата данни.

Очакван резултат: Обектът да бъде успешно възстановен от базата данни и да съответства на входните данни.

[Fact]

async void UnitTestRead()

{

var databaseContext = await GetDbContext();

var controller = new TypeOfTimbersController(databaseContext);

var result = controller.Details(1);

result.Equals(true);

}

**Тестване на актуализирането на обект в базата данни**

Входни данни: Идентификатор на обект от клас TypeOfTimber, който съществува в базата данни, и обект с нови стойности за TimberName, AmountForLogging и YearOfLogging.

Очакван резултат: Обектът да бъде успешно актуализиран в базата данни и да съответства на входните данни.

[Fact]

async void UnitTestUpdate()

{

var TypeOfTimber = new TypeOfTimber()

{

TimberName = TypeOfTimberEnum.Sanitary,

AmountForLogging = 1,

YearOfLogging = DateTime.Now

};

var databaseContext = await GetDbContext();

var controller = new TypeOfTimbersController(databaseContext);

var result = controller.Edit(11,TypeOfTimber);

result.Equals(true);

}

**Тестване на изтриването на обект от базата данни**

Входни данни: Идентификатор на обект от клас TypeOfTimber, който съществува в базата данни.

Очакван резултат: Обектът да бъде успешно изтрит от базата данни и броят на записите в таблицата TypeOfTimber да се намали с единица.

[Fact]

async void UnitTestDelete()

{

var TypeOfTimber = new TypeOfTimber()

{

TimberName = TypeOfTimberEnum.Sanitary,

AmountForLogging = 1,

YearOfLogging = DateTime.Now

};

var databaseContext = await GetDbContext();

var controller = new TypeOfTimbersController(databaseContext);

controller.Create(TypeOfTimber);

await controller.DeleteConfirmed(TypeOfTimber.Id);

databaseContext.SaveChanges();

Assert.Equal(0, databaseContext.TypeOfTimber.Count());

}

18

**Извършване на тестове в различни браузъри**

Проектът може да се тества в различни браузъри като Google Chrome, Mozilla Firefox, Safari и Internet Explorer, за да се потвърди, че функционалността работи правилно във всички тях.

**Тестване за производителност**

При извършването на тестове на производителност е важно да се проверят времето за отговор на уеб приложението, времето за зареждане на страници и други параметри, които могат да доведат до забавяне на уеб сайта.

# Заключение и възможно бъдещо развитие

Реализирано решение на представения проект има Югоизточното Държавно предприятие със седалище град Сливен, в тяхната платформа има информация за абсолютно всички горски и ловни стопанства, които попадат в територията на предприятието. Платформата разполага със следните страници Начало, Информация, Новини, Структура, Процедури, Профил на купувача, Разсадници, Контакти. Всяка от тези страници съдържа информация и голямо количество документация, за разлика от тази платформа, “Greenvision” цели да резюмира това количество от информация и да улесни достъпа на потребителя до важни данни и синтезирани решения.

Резултатите от работата върху проекта са разработена функционална платформа, която дава възможност на потребителите да получат информация относно зелените зони в страната и тяхното правилно управление, както и да се включат данни за регистрация и добавяне на информация. Платформата предлага няколко нива на достъп, като експертите и администраторите имат възможност да обработват и добавят информация в системата. Разработеният алгоритъм помага за вземане на решение за нивото на реализирано обезлесяване в дадено стопанство.

Предимствата на използваните технологии и методи включват лесната достъпност до потребителите и възможността за добавяне на информация в системата. Ограниченията включват липсата на функционалности като автоматично добавяне на информация и възможност за масова обработка на данни.

Алтернативите на използваните технологии включват други езици и програмни платформи, като например *Python*, *Java*, *SpringBoot*. Предимствата на тези алтернативи включват по-голяма гъвкавост в програмирането и възможност за добавяне на функционалности.

Подобни решения имат широко приложение в практиката, като предоставят достъпна информация за зелените зони и помагат за управлението им.

За бъдещо развитие на решението може да се добавят допълнителни функционалности, като автоматично добавяне на информация и възможност за масова обработка на данни. Освен това, може да се разработи и мобилно приложение за по-лесен достъп до информацията.

# Използвани литературни източници и Уеб сайтове

|  |  |
| --- | --- |
| [1] | Национален Статистически Институт, „Цени на производител в горското стопанство,“ 17 02 2023. [Онлайн]. Available: https://www.nsi.bg/bg/content/16220/цени-на-производител-в-горското-стопанство. |
| [2] | Югоизточно държавно предприятие, „Югоизточно държавно предприятие - гр. Сливен,“ [Онлайн]. Available: https://www.uidp-sliven.com/. |
| [3] | M. Porter, „Бази данни,“ 21 03 2021. [Онлайн]. Available: https://bg.warbletoncouncil.org/tercera-forma-normal-4842. |
| [4] | S. Smith, „Overview of ASP.NET Core MVC,“ 27 06 2022. [Онлайн]. Available: https://learn.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-7.0. |
| [5] | Microsoft, „Tutorial: Implement CRUD Functionality with the Entity Framework in ASP.NET MVC,“ 07 01 2022. [Онлайн]. Available: https://learn.microsoft.com/en-us/aspnet/mvc/overview/getting-started/getting-started-with-ef-using-mvc/implementing-basic-crud-functionality-with-the-entity-framework-in-asp-net-mvc-application#overpost. |
| [6] | T. Hastie, „Decision Trees,“ 22 05 2022. [Онлайн]. Available: https://scikit-learn.org/stable/modules/tree.html. |
| [7] | J. R. Quinlan, „Decision Tree,“ 18 04 2023. [Онлайн]. Available: https://www.geeksforgeeks.org/decision-tree/. |
| [8] |  |

1. SQLite and SQL Server Compact Toolbox – разширение към VisualStudio [↑](#footnote-ref-2)