

ПРОФЕСИОНАЛНА ГИМНАЗИЯ ПО ЕЛЕКТРОТЕХНИКА И
ЕЛЕКТРОНИКА „КОНСТАНТИН ФОТИНОВ“, ГР. БУРГАС

ДИПЛОМЕН ПРОЕКТ

на Велин Владинов Куртев

ученик/ученичка от XII б клас

професия - „Приложен програмист“, код: 481030

специалност - „Приложно програмиране“, код: 4810301

Тема: Оценка на застрахователен риск

Ръководител-консултант: Петър Петров

Сесия: май-юни 2022г.

Дата:.....

Съдържание

1. Увод.....	3
2. Изложение.....	4
2.1 Цел на дипломния проект.....	4
2.2 Използвани технологии.....	4
2.2.1 Уеб приложение.....	4
2.2.2 ASP.NET Core MVC.....	4
2.2.2.1 MVC.....	5
2.2.2.2 Отговорности на модела.....	6
2.2.2.3 Отговорности на изгледа.....	6
2.2.2.4 Отговорности на контролера.....	6
2.2.2.5 Маршрутизиране.....	7
2.2.2.5 Model binding.....	7
2.2.2.6 Области.....	7
2.2.3 C#.....	8
2.2.3.1 Класове и обекти.....	8
2.2.3.2 Интерфейси.....	8
2.2.3.3 Полета.....	9
2.2.3.4 Методи.....	9
2.2.3.5 Параметри.....	10
2.2.3.6 Виртуални, override , абстрактни методи.....	10
2.2.3.7 Overload-ване на методи.....	11
2.2.3.8 Конструктори.....	11
2.2.4 ООП.....	12
2.2.4.1 Капсулиране.....	12
2.2.4.2 Наследяване.....	12
2.2.4.3 Полиморфизъм.....	12
2.2.4.4 Повторна употреба.....	13
2.2.5 HTML.....	13
2.2.6 CSS.....	14
2.2.7 JavaScript.....	14
2.2.8 SQL.....	15
2.3 Задачи , които трябва да се решат за постигането на целта.....	15
2.4 Приноси.....	16
2.5 Изпълнение на задачите.....	16
2.5.1. Проектиране на база данни.....	16
2.5.2 Проектиране на слой за достъп до данни.....	22
2.5.3 Проектиране на слой за услуги.....	27
2.5.4 Проектиране на презентационен слой.....	28
2.5.4.1 Проектиране на контролери.....	28
2.5.4.2 Проектиране на модели за изгледи.....	28
2.5.4.4 Проектиране на изгледи.....	28
2.5.5 Проектиране на алгоритъм за оценка на застрахователният риск.....	29
3. Заключение.....	36
4. Информационни източници.....	36
5. Приложения.....	37
6. Рецензия на дипломен проект.....	38

1. Увод

Приложението „*Insurance Risk Assessment*” разработка в областта на застрахователния сектор има за цел да улесни работата на застрахователя , като му помогне да вземе правилния избор дали си заслужава застраховката на дадено имущество или не. Това е възможно чрез оценка на риска от настъпването на застрахователно събитие. Риска е потенциалната опасност за застрахованото лице, но не е известно кога и дали изобщо тази опасност ще се осъществи.

Този тип рискове или опасности могат да причинят финансови загуби като имуществени вреди или телесни повреди. Ако настъпи неблагоприятното събитие и бъде предявен иск, застрахователната компания е длъжна да обезщети притежателя на полицата в уговорената сума. Риск от пожар, загуби от земетресение и унищожаване на имущество са само няколко примера за застрахователни рискове. Автомобилните злополуки са друг пример за риск за автомобилната застраховка. Описано просто, застрахователните рискове са тези, за които застрахователната компания се е ангажирала да осигури парично обезщетение. Застрахователните рискове включват широк спектър от събития.

В резултат на това е изключително важно застрахователят да вземе правилната преценка, като определи дали процентът на риска от настъпване на конкретно събитие е достатъчно висок, за да предложи сума, достатъчна за покриване на част от щетата към момента на настъпване на събитието. Трябва да се подчертае, че оценката на риска е субективна, тъй като е невъзможно да се обхванат всички възможни рискови сценарии.

Приложението помага на застрахователния орган като чрез получените данни изготвя резултат на оценка на риска и по този начин застрахователят се улеснява при взимането на правилното решение. След обстойно премисляне на обстоятелствата, застрахователят сам преценя дали да застрахова даденото имущество.

2. Изложение

2.1 Цел на дипломния проект

Главната цел на дипломния проект е разработката на приложение за оценка на застрахователен риск. Функционалността на приложението е да подпомага оценката на застрахователя и менажира запазените от застрахователя резултати.

2.2 Използвани технологии

2.2.1 Уеб приложение

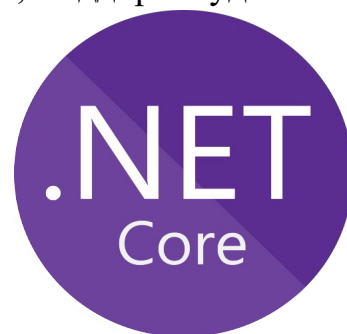
Уеб приложение е програма, до която потребителите имат достъп чрез мрежа като Интернет. Поддържан от браузър език за програмиране (като JavaScript), комбиниран с изобразен от браузър език за маркиране (като HTML), също може да се използва за описание на софтуерно приложение, което се изобразява от типичните уеб браузъри.

Уеб приложенията са широко използвани поради широкото разпространение на уеб браузъри. Тъй като използват както уеб браузъри, така и уеб технологии за извършване на специфични операции през Интернет, тези технологии имат огромна целева аудитория.

2.2.2 ASP.NET Core MVC

ASP.NET Core MVC е framework, с отворен код, притежава силно тестуем framework за презентации, оптимизиран за използване с ASP.NET Core.

ASP.NET Core MVC предоставя базиран на модели начин за изграждане на динамични уеб-сайтове, който позволява ясно разделяне на проблемите. Той ви дава пълен контрол върху маркирането, поддържа удобна за TDD разработка, използвайки най-новите уеб стандарти.



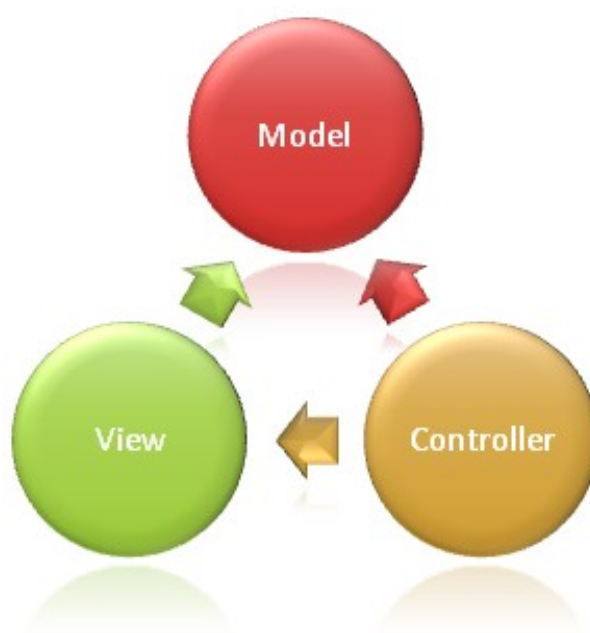
Снимка 1: .Net Core лого

2.2.2.1 MVC

Архитектурният модел Model-View-Controller (MVC) разделя приложението на три основни групи компоненти: модели, изгледи и контролери.

Използвайки този модел, потребителските заявки се насочват към контролер, който е отговорен за работата с модела за извършване на потребителски действия и/или извличане на резултати от заявки. Контролерът избира изгледа за показване на потребителя и му предоставя всички данни за модела, които изисква.

Това разграничаване на отговорностите ви помага да мащабирате приложението по отношение на сложността, защото е по-лесно да кодирате, отстранявате грешки и тествате нещо (модел, изглед или контролер), което има една задача. По-трудно е за актуализиране, тестване и отстраняване на грешки в код, който има зависимости, разпространени в две или повече от тези три области. Например, логиката на потребителския интерфейс има тенденция да се променя по-често от бизнес логиката. Ако кодът за представяне и бизнес логиката се комбинират в един обект, обект, съдържащ бизнес логика, трябва да се променя всеки път, когато се променя потребителският интерфейс. Това често въвежда грешки и изисква повторно тестване на бизнес логиката след всяка минимална промяна на потребителския интерфейс.



Снимка 2: MVC диаграма

2.2.2.2 Отговорности на модела

Моделът в MVC приложение представлява състоянието на приложението и всяка бизнес логика или операции, които трябва да се изпълняват от него. Бизнес логиката трябва да бъде капсулирана в модела, заедно с всяка логика на изпълнение за запазване на състоянието на приложението. Строго въведените изгледи обикновено използват типове `ViewModel`, предназначени да съдържат данните за показване в този изглед. Контролерът създава и попълва тези екземпляри на `ViewModel` от модела.

2.2.2.3 Отговорности на изгледа

`View`-тата са отговорни за представянето на съдържание чрез потребителския интерфейс. Те използват механизма за преглед на `Razor` за вграждане на `.NET` код в `HTML` маркиране. В `ASP.NET Core MVC` изгледите са `.cshtml` файлове, които използват езика за програмиране `C#` в маркирането на `Razor`. В изгледите трябва да има минимална логика и всяка логика в тях трябва да се отнася до представянето на съдържание. Ако установите, че е необходимо да изпълнявате голяма част от логиката във файловете за изглед, за да покажете данни от сложен модел, помислете за използването на `View Component`, `ViewModel` или шаблон за изглед, за да опростите изгледа.

2.2.2.4 Отговорности на контролера

Контролерите са компонентите, които управляват взаимодействието с потребителя, работят с модела и в крайна сметка избират изглед за изобразяване. В MVC приложение изгледът показва само информация; контролерът обработва и реагира на въвеждането и взаимодействието на потребителя. В модела MVC контролерът е началната входна точка и е отговорен за избора с кои типове модели да работи и кой изглед да изобрази (оттук и името му – той контролира как приложението отговаря на дадена заявка).

2.2.2.5 Маршрутизиране

ASP.NET Core MVC е изграден върху маршрутизирането на ASP.NET Core, мощен компонент за картографиране на URL адреси, който ви позволява да създавате приложения, които имат разбираеми и достъпни за търсене URL адреси.

Това ви позволява да дефинирате шаблоните за именуване на URL адреси на вашето приложение, които работят добре за оптимизация за търсачки (SEO) и за генериране на връзки, без да се съобразява с това как са организирани файловете на вашия уеб сървър. Можете да дефинирате вашите маршрути, като използвате удобен синтаксис на шаблон за маршрут, който поддържа ограничения за стойността на маршрута, стойности по подразбиране и незадължителни стойности.

2.2.2.5 Model binding

Обвързването на модела на ASP.NET Core MVC преобразува данните от клиентската заявка (стойности на формуляри, данни за маршрут, параметри на низ за заявка, HTTP заглавки) в обекти, които контролерът може да обработва. В резултат на това логиката на вашия контролер не трябва да върши работата по установяване на входящите данни за заявка; той просто има данните като параметри на своите методи за действие.

2.2.2.6 Области

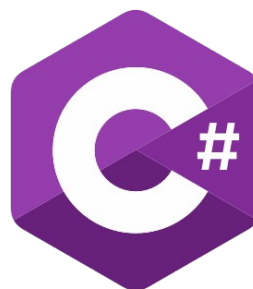
Областите предоставят начин за разделяне на голямо ASP.NET Core MVC уеб приложение на по-малки функционални групи. Областта е MVC структура вътре в приложение. В MVC проект логическите компоненти като модел, контролер и изглед се съхраняват в различни папки и MVC използва конвенции за именуване, за да създаде връзката между тези компоненти. За голямо приложение може да е изгодно приложението да се раздели на отделни функционални области от високо ниво. Например, приложение за електронна търговия с множество бизнес единици, като плащане, фактуриране и търсене и т.н. Всяко от тези единици има свои собствени изгледи на логически

компоненти, контролери и модели.

2.2.3 C#

C# е модерен, обектно-ориентиран, типизиран, език за програмиране. Той позволява на разработчиците да създават много видове сигурни и стабилни приложения, които работят в .NET.

C# програмите работят на .NET, виртуална система за изпълнение, наречена обща езикова среда за изпълнение (CLR) и набор от библиотеки на класове. CLR е внедряването от Microsoft на общата езикова инфраструктура (CLI), международен стандарт. CLI е основата за създаване на среди за изпълнение и разработка, в които езиците и библиотеките работят безпроблемно.



Снимка 3: C# лого

2.2.3.1 Класове и обекти

Класовете са най-фундаменталните типове на C#. Класът е структура от данни, която съчетава състояние (полета) и действия (методи и други членове на функции) в една единица. Класът предоставя дефиниция за екземпляри на класа, известни също като обекти. Класовете поддържат наследяване и полиморфизъм, механизми, чрез които производните класове могат да разширяват и специализират базови класове.

2.2.3.2 Интерфейси

Интерфейсът дефинира договор, който може да бъде реализиран от класове и структури. Вие дефинирате интерфейс за деклариране на

способности, които са споделени между отделни типове. Интерфейсът обикновено не предоставя реализации на членовете, които дефинира – той просто указва членовете, които трябва да бъдат предоставени от класове или структури, които реализират интерфейса. `interfaces` може да използва множествено наследяване.

2.2.3.3 Полета

Полето е променлива, която е свързана с клас или с екземпляр на клас. Поле, декларирано с модификатора `static`, дефинира статично поле. Статично поле идентифицира точно едно място за съхранение. Без значение колко екземпляра на клас са създадени, винаги има само едно копие на статично поле. Поле, декларирано без модификатора `static`, дефинира екземплярно поле. Всеки екземпляр на клас съдържа отделно копие на всички полета на екземпляра на този клас.

2.2.3.4 Методи

Методът е член, който реализира изчисление или действие, което може да бъде извършено от обект или клас. Достъпът до статичните методи се осъществява чрез класа. Достъпът до методите на екземпляра се осъществява чрез екземпляри на класа.

Методите могат да имат списък с параметри, които представляват стойности или препратки към променливи, предадени на метода. Методите имат тип на връщане, който определя типа на стойността, изчислена и върната от метода. Типът на връщането на метода е невалиден, ако не връща стойност.

Подобно на типовете, методите могат също да имат набор от параметри на типа, за които аргументите на типа трябва да бъдат посочени при извикване на метода. За разлика от типовете, аргументите на типа често могат да бъдат изведени от аргументите на извикване на метод и не е необходимо да се дават изрично.

2.2.3.5 Параметри

Параметрите се използват за предаване на стойности или препратки към променливи към методи. Параметрите на метод получават действителните си стойности от аргументите, които са посочени при извикване на метода. Има четири вида параметри: стойностни параметри, референтни параметри, изходни параметри и масиви от параметри.

Параметър стойност се използва за предаване на входни аргументи. Параметър на стойност съответства на локална променлива, която получава първоначалната си стойност от аргумента, който е бил предаден за параметъра. Промените на параметър на стойност не засягат аргумента, който е бил предаден за параметъра.

2.2.3.6 Виртуални, `override` , абстрактни методи

Използвайте виртуални, заменящи и абстрактни методи, за да дефинирате поведението за йерархия от типове класове. Тъй като един клас може да произлиза от базов клас, тези производни класове може да се наложи да променят поведението, внедрено в базовия клас. Виртуален метод е деклариран и реализиран в базов клас, където всеки производен клас може да осигури по-специфична реализация. Методът за отмяна е метод, внедрен в производен клас, който променя поведението на реализацията на базовия клас.

Абстрактният метод е метод, деклариран в основен клас, който трябва да бъде отменен във всички производни класове. Всъщност абстрактните методи не дефинират реализация в базовия клас.

Извикванията на метод към методите на екземпляра могат да се разрешат или до реализации на базов клас, или до производен клас. Типът на променливата определя нейния тип по време на компилиране. Типът по време на компилиране е типът, който компилаторът използва, за да определи своите членове. Въпреки това, променлива може да бъде присвоена на екземпляр от всеки тип, извлечен от неговия тип по време на компилиране. Типът по време на изпълнение е типът на действителния екземпляр, към който се отнася променливата.

Когато се извика виртуален метод, типът по време на изпълнение на екземпляра, за който се извършва това извикване, определя действителната реализация на метода, която да се извика. При извикване на неvirtуален метод, типът по време на компилиране на екземпляра е определящият фактор.

Виртуален метод може да бъде отменен в произведен клас. Декларацията на виртуален метод въвежда нов метод. Декларацията на метод за отмяна специализира съществуващ наследен виртуален метод, като предоставя нова реализация на този метод. Абстрактният метод е виртуален метод без реализация. Абстрактен метод се декларира с абстрактния модификатор и е разрешен само в абстрактен клас. Абстрактен метод трябва да бъде отменен във всеки неабстрактен извлечен клас.

2.2.3.7 Overload-ване на методи

Overload-ването на метод позволява на множество методи в един и същи клас да имат едно и също име, стига да имат уникални подписи. Когато компилира извикване на претоварен метод, компилаторът използва разделителна способност при претоварване, за да определи конкретния метод за извикване. Разрешаването на претоварване намира метода, който най-добре отговаря на аргументите. Ако не може да бъде намерено нито едно най-добро съвпадение, се съобщава за грешка.

2.2.3.8 Конструктори

C# поддържа както инстанционни, така и статични конструктори. Конструкторът на екземпляр е член, който изпълнява действията, необходими за инициализиране на екземпляр на клас. Статичният конструктор е член, който изпълнява действията, необходими за инициализиране на самия клас при първото му зареждане.

Конструкторът се декларира като метод без тип на връщане и същото име като съдържащия клас. Ако декларацията на конструктора включва статичен модификатор, тя декларира статичен конструктор. В противен случай

той декларира конструктор на инстанция.

2.2.4 ООП

Обектно-ориентираното програмиране (ООП) е основната съставка на .NET framework-а. ООП-то е толкова важно, че преди да тръгнете по пътя към .NET, трябва да разберете неговите основни принципи и терминология, за да напишете дори проста програма. Основната идея зад ООП е да се комбинират в едно цяло както данните, така и методите, които оперират с тези данни; такива единици се наричат обект. Всички ООП езици предоставят механизми, които ви помагат да приложите обектно-ориентирания модел. Те са капсулиране, наследяване, полиморфизъм и повторна употреба.

2.2.4.1 Капсулиране

Капсулирането свързва кода и данните, които манипулира, и ги предпазва от външна намеса и злоупотреба. Капсулирането е защитен контейнер, който предотвратява достъпа до код и данни от друг код, дефиниран извън контейнера.

2.2.4.2 Наследяване

Наследяването е процесът, чрез който един обект придобива свойствата на друг обект. Типът произлиза от основен тип, като взема всички полета и функции на членовете на базовия тип. Наследяването е най-полезно, когато трябва да добавите функционалност към съществуващ тип. Например всички .NET класове наследяват от класа System.Object, така че класът може да включва нова функционалност, както и да използва функциите и свойствата на класа на съществуващия обект.

2.2.4.3 Полиморфизъм

Полиморфизмът е функция, която позволява един интерфейс да се използва за общ клас на действие. Тази концепция често се изразява като „един

интерфейс, множество действия“. Конкретното действие се определя от точния характер на обстоятелствата.

2.2.4.4 Повторна употреба

След като класът е написан, създаден и отстранен, той може да бъде разпространен на други програмисти за използване в тяхната собствена програма. Това се нарича повторна употреба или в терминологията на .NET тази концепция се нарича компонент или DLL. В ООП обаче наследяването предоставя важно разширение на идеята за повторна употреба. Програмистът може да използва съществуващ клас и без да го модифицира, да добавя допълнителни функции към него.

2.2.5 HTML

HTML (HyperText Markup Language) е най-основният градивен елемент на мрежата. Той определя значението и структурата на уеб съдържанието. Други технологии освен HTML обикновено се използват за описване на външния вид/презентацията (CSS) или функционалността/поведението на уеб страницата (JavaScript).

„Хипертекст“ се отнася до връзки, които свързват уеб страници една с друга, или в рамките на един уеб-сайт, или между уеб-сайтове. Връзките са основен аспект на мрежата. Като качвате съдържание в Интернет и го свързвате със страници, създадени от други хора, вие ставате активен участник в World Wide Web.

HTML използва "маркиране" за аотиране на текст, изображения и друго съдържание за показване в уеб браузър.



Снимка 4: HTML5 logo

2.2.6 CSS

Каскадните стилкови таблици (CSS) е език за стилове, използван за описване на представянето на документ, написан в HTML или XML.

CSS описва как елементите трябва да бъдат изобразени на екран, на хартия, в реч или на друга медия.

CSS е сред основните езици на отворената мрежа и е стандартизиран в уеб браузърите според спецификациите на W3C.



Снимка 5: CSS лого

2.2.7 JavaScript

JavaScript (JS) е лек, интерпретиран или компилиран навреме език за програмиране с първокласни функции. Въпреки че е най-известен като скриптов език за уеб страници, много среди, които не са браузъри, също го използват, като Node.js, Apache CouchDB и Adobe Acrobat. JavaScript е базиран на прототип, многопарадигмен, еднонишков, динамичен език, поддържащ обектно-ориентирани, императивни и декларативни (например функционално програмиране) стиллове.



JavaScript

Снимка 6: JavaScript лого

2.2.8 SQL

SQL е специфичен за домейна език, използван в програмирането и предназначен за управление на данни, съхранявани в система за управление на релационна база данни (RDBMS) или за обработка на потоци в система за управление на релационни потоци от данни (RDSMS). Той е особено полезен при работа със структурирани данни, т.е. данни, включващи връзки между обекти и променливи.



Снимка 7: SQL лого

2.3 Задачи , които трябва да се решат за постигането на целта

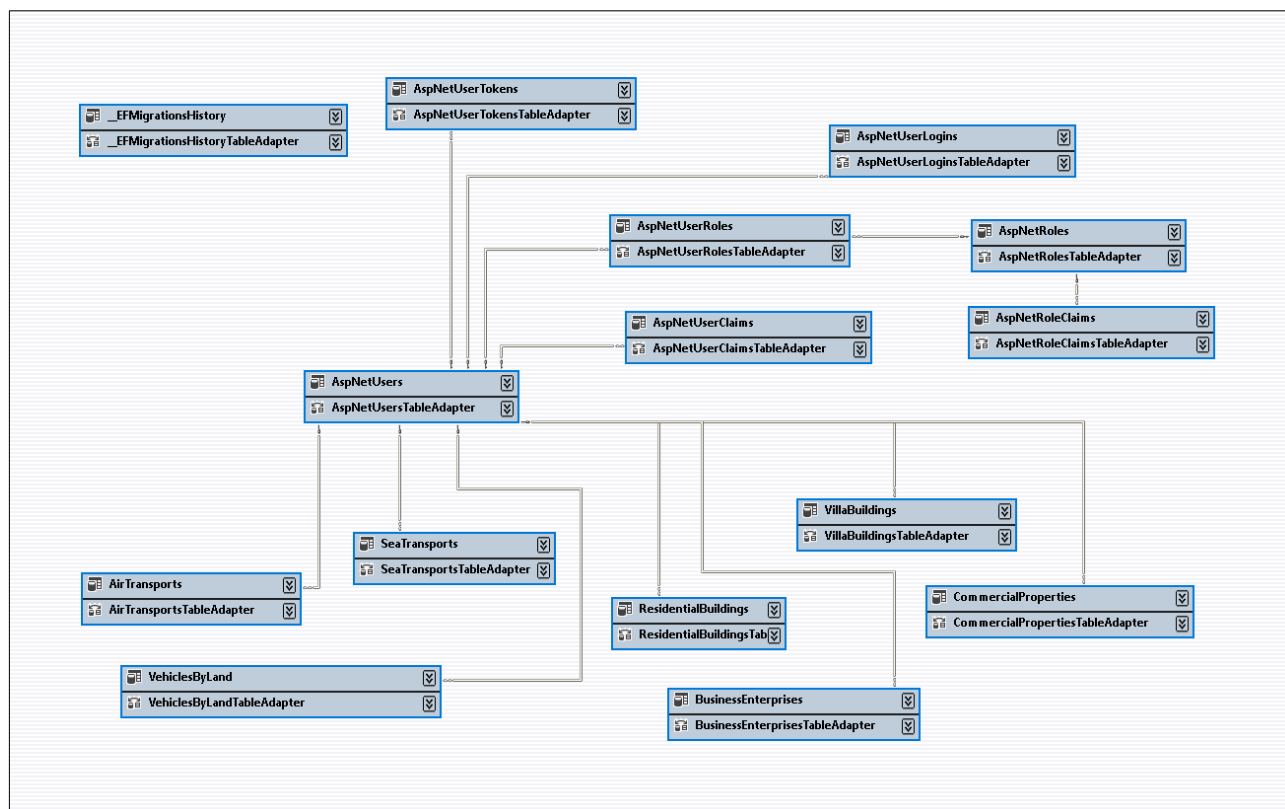
- ◆ Проектиране на база данни
- ◆ Проектиране на слой за достъп до данни
- ◆ Проектиране на слой за услуги
- ◆ Проектиране на презентационен слой
 - Проектиране на контролери
 - Проектиране на модели за изгледи
 - Проектиране на изгледи
 - Проектиране на потребителски интерфейс
 - Проектиране на потребителско изживяване
- ◆ Проектиране на алгоритъм за оценка на застрахователният риск

2.4 Приноси

Приложението, което се разработва в този дипломен проект е насочено към застрахователни компании или лица занимаващи се със застрахователна дейност. До този момент приложението има функцията да изчислява застрахователния риск на три вида движимо имущество , а именно пътен, воден и въздушен транспорт , както и на четири вида недвижимо имущество – бизнес предприятия, вилни сгради, стопанско имущество и жилищни сгради.

2.5 Изпълнение на задачите

2.5.1. Проектиране на база данни



Снимка 8: Диаграма на БД

Създадена е база данни с 15 таблици, които имат един-към-много и много-към-много връзки. Базата данни е изградена чрез scaffolding на entity класовете в Data Access Layer-а. Основните таблици са:

- Въздушни транспорти(AirTransports)
 - Id – PK – int
 - Name – nvarchar
 - CreatedAt - DateTime
 - DistanceTraveled – int
 - Functionality – nvarchar
 - Height – float
 - ManufactureYear – DateTime
 - ModifiedAt – DateTime
 - PreviousAccidents – bit
 - RegisteredCity – nvarchar
 - RegisteredCountry – nvarchar
 - RegisteredRegion – nvarchar
 - SecurityEquipmentPossession – bit
 - TechnicalServiceability – bit
 - Weight – float
 - Width – float
 - InsuranceBrokerId – int
 - ResultValue – int
- Водни транспорти(SeaTransports)
 - Id – PK – int
 - Name – nvarchar
 - CreatedAt - DateTime
 - DistanceTraveled – int
 - DoesRoutePassesPirateZones – bit
 - Functionality – nvarchar
 - Height – float
 - ManufactureYear – DateTime
 - ModifiedAt – DateTime
 - PreviousAccidents – bit

Air Transports	
Id	
Name	
CreatedAt	
DistanceTraveled	
Functionality	
Height	
ManufactureYear	
ModifiedAt	
PreviousAccidents	
RegisteredCity	
RegisteredCountry	
RegisteredRegion	
SecurityEquipmenPossession	
TechnicalServiceability	
Weight	
Width	
InsuranceBrokerId	
ResultValue	

Снимка 9: Таблица въздушни транспорти

SeaTransports	
Id	
Name	
CreatedAt	
DistanceTraveled	
DoesRoutePassesPirateZones	
Functionality	
Height	
ManufactureYear	
ModifiedAt	
PreviousAccidents	
RegisteredCity	
RegisteredCountry	
RegisteredRegion	
SecurityEquipmenPossession	
TechnicalServiceability	
TypeOfMovability	
Weight	
Width	
InsuranceBrokerId	
ResultValue	

Снимка 10: Таблица водни транспорти

- RegisteredCity – nvarchar
- RegisteredCountry – nvarchar
- RegisteredRegion – nvarchar
- SecurityEquipmentPossession – bit
- TechnicalServicability – bit
- Weight – float
- Width – float
- InsuranceBrokerId – int
- ResultValue – int

- Пътен транспорт(VehiclesByLand)

- Id – PK – int
- FuelType – nvarchar
- Parktronic – bit
- MostCommonRoutes - nvarchar
- RegisterNumber – nvarchar
- CreatedAt – DateTime
- ModifiedAt – DateTime
- PreviousAccidents – bit
- ManufactureYear – DateTime
- SecurityEquipmentPossession – bit
- TechnicalServicability – bit
- DistanceTraveled – int
- RegisteredCity – nvarchar
- RegisteredCountry – nvarchar
- RegisteredRegion – nvarchar
- Height – float
- Weight – float
- Width – float
- InsuranceBrokerId – int
- ResultValue – int

VehiclesByLand	
?	Id
	FuelType
	Parktronic
	MostCommonRoutes
	RegisterNumber
	CreatedAt
	ModifiedAt
	PreviousAccidents
	ManufactureYear
	SecurityEquipmenPossession
	TechnicalServiceability
	DistanceTraveled
	Height
	Weight
	Width
	RegisteredCountry
	RegisteredRegion
	RegisteredCity
	InsuranceBrokerId
	ResultValue

Снимка 11: Таблица пътен транспорт

- Жилищни сгради(ResidentialBuildings)

- Id – PK – int
- Address – nvarchar
- AlarmSystem – bit
- City – nvarchar
- Country – nvarchar
- CreatedAt – DateTime
- EmergencyExit – bit
- FireExtinguishers – bit
- Floor – nvarchar
- GasBottles – bit
- ModifiedAt – DateTime
- PreviousAccidents – bit
- Region – nvarchar
- SquareFeet – float
- InsuranceBrokerId – int
- ResultValue – int

ResidentialBuildings	
Id	
Address	
AlarmSystem	
City	
Country	
CreatedAt	
EmergencyExit	
FireExtinguishers	
Floor	
GasBottles	
ModifiedAt	
PreviousAccidents	
Region	
SquareFeet	
InsuranceBrokerId	
ResultValue	

Снимка 12: Таблица жилищни сгради

- Бизнес предприятия(BusinessEnterprises)

- Id – PK – int
- Address – nvarchar
- AlarmSystem – bit
- City – nvarchar
- Country – nvarchar
- CreatedAt – DateTime
- EmergencyExit – bit
- FireExtinguishers – bit
- GasBottles – bit
- ModifiedAt – DateTime
- PreviousAccidents – bit

BusinessEnterprises	
Id	
Address	
AlarmSystem	
City	
Country	
CreatedAt	
EmergencyExit	
FireExtinguishers	
GasBottles	
ModifiedAt	
PreviousAccidents	
PurposeOfTheEnterprise	
Region	
SquareFeet	
InsuranceBrokerId	
ResultValue	

Снимка 13: Таблица бизнес предприятия

- PurposeOfTheEnerprise - nvarchar
- Region – nvarchar
- SquareFeet – float
- InsuranceBrokerId – int
- ResultValue – int

- Вилни сгради(VillaBuildings)

- Id – PK – int
- Address – nvarchar
- AlarmSystem – bit
- City – nvarchar
- Country – nvarchar
- CreatedAt – DateTime
- EmergencyExit – bit
- FireExtinguishers – bit
- GasBottles – bit
- ModifiedAt – DateTime
- PreviousAccidents – bit
- Region – nvarchar
- SquareFeet – float
- InsuranceBrokerId – int

- Стопанско имущество(CommercialBuildings)

- Id – PK – int
- Address – nvarchar
- AlarmSystem – bit
- City – nvarchar
- Country – nvarchar
- CreatedAt – DateTime
- EmergencyExit – bit

VillaBuildings	
Id	
CreatedAt	
ModifiedAt	
PreviousAccidents	
Country	
Region	
City	
Address	
FireExtinguishers	
EmergencyExit	
SquareFeet	
AlarmSystem	
GasBottles	
InsuranceBrokerId	
ResultValue	

Снимка 14: Таблица вилни сгради

CommercialProperties	
Id	
Address	
AlarmSystem	
City	
Country	
CreatedAt	
EmergencyExit	
FireExtinguishers	
GasBottles	
ModifiedAt	
PreviousAccidents	
Region	
SquareFeet	
InsuranceBrokerId	
ResultValue	

Снимка 15: Таблица стопанско имущество

- FireExtinguishers – bit
- GasBottles – bit
- ModifiedAt – DateTime
- PreviousAccidents – bit
- Region – nvarchar
- SquareFeet – float
- InsuranceBrokerId – int

- Потребители(AspNetUsers)

- Id – nvarchar – GUID
- UserName – nvarchar
- NormalizedUserName – nvarchar
- Email -nvarchar
- NormalizedEmail – nvarchar
- EmailConfirmed – bit
- PasswordHash – nvarchar
- SecurityStamp – nvarchar
- ConcurrencyStamp – nvarchar
- PhoneNumber – nvarchar
- PhoneNumberConfirmed – bit
- TwoFactorEnabled – bit
- LockoutEnd – DateTime
- LockoutEnabled – bit
- AccesFailedCount - int

AspNetUsers	
🔑	Id
	UserName
	NormalizedUserName
	Email
	NormalizedEmail
	EmailConfirmed
	PasswordHash
	SecurityStamp
	ConcurrencyStamp
	PhoneNumber
	PhoneNumberConfirmed
	TwoFactorEnabled
	LockoutEnd
	LockoutEnabled
	AccessFailedCount

Снимка 16: Таблица потребители

2.5.2 Проектиране на слой за достъп до данни

Слоят за достъп до данни се състои от 5 папки :

- Abstractions – Съдържа IRepository.cs файлът, той съдържа IRepository интерфейс, който имплементира шаблон за Repository.cs файлът.

```
public interface IRepository<T>
{
    1 reference
    T Get(Func<T, bool> predicate);
    8 references
    T GetById(int id);
    8 references
    List<T> GetAll();
    1 reference
    List<T> Find(Func<T, bool> predicate);
    8 references
    bool Create(T entity);
    8 references
    bool Update(T entity);
    1 reference
    bool Remove(T entity);
    8 references
    bool RemoveById(int id);
}
```

Снимка 17: IRepository.cs код

- Data – Съдържа InsuranceRiskAssessmentDbContext.cs файлът. Неговото предназначение е да подпомогне взимането на данните от БД. Съдържа DbSet-ове на всяко едно от необходимите ни entity-та.

```
13 references
public class InsuranceRiskAssessmentDbContext : IdentityDbContext<InsuranceBrokerUser>
{
    0 references
    public InsuranceRiskAssessmentDbContext(DbContextOptions<InsuranceRiskAssessmentDbContext> options)
        : base(options)
    {
    }

    0 references
    public DbSet<AirTransport> AirTransports { get; set; }
    0 references
    public DbSet<SeaTransport> SeaTransports { get; set; }
    0 references
    public DbSet<VehicleByLand> VehiclesByLand { get; set; }
    0 references
    public DbSet<BusinessEnterprise> BusinessEnterprises { get; set; }
    0 references
    public DbSet<CommercialProperty> CommercialProperties { get; set; }
    0 references
    public DbSet<ResidentialBuilding> ResidentialBuildings { get; set; }
    0 references
    public DbSet<VillaBuilding> VillaBuildings { get; set; }
    0 references
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        base.OnConfiguring(optionsBuilder);
        optionsBuilder.UseLazyLoadingProxies();
    }
}
```

Снимка 18: InsuranceRiskAssessmentDbContext.cs код

- Migrations – Съдържа файлове с миграциите направени по БД.
- Entities - Съдържа 11 файла:
 - BaseEntity.cs - описва основните стойности за всяка от таблиците. Те са ID, дата на създаване, дата на модификация, предишни инциденти и резултат.

```

4 references
public abstract class BaseEntity
{
    0 references
    public BaseEntity()
    {
        CreatedAt = DateTime.Now;
    }

    30 references
    public int Id { get; set; }
    30 references
    public DateTime CreatedAt { get; set; }
    43 references
    public DateTime ModifiedAt { get; set; }
    42 references
    public bool PreviousAccidents { get; set; }
    42 references
    public int ResultValue { get; set; }
}

```

Снимка 19: BaseEntity.cs код

- InsuranceBrokerUser.cs – разширява IdentityUser класа който отговаря за потребителите в приложението с колекции от всеки един от необходимите ни entity класове.

```

26 references
public class InsuranceBrokerUser : IdentityUser
{
    0 references
    public virtual ICollection<AirTransport> CheckedAirTransport { get; set; }
    0 references
    public virtual ICollection<SeaTransport> CheckedSeaTransport { get; set; }
    0 references
    public virtual ICollection<VehicleByLand> CheckedVehicleByLand { get; set; }

    0 references
    public virtual ICollection<BusinessEnterprise> CheckedBusinessEnterprise { get; set; }
    0 references
    public virtual ICollection<CommercialProperty> CheckedCommercialProperty { get; set; }
    0 references
    public virtual ICollection<ResidentialBuilding> CheckedResidentialBuildings { get; set; }
    0 references
    public virtual ICollection<VillaBuilding> CheckedVillaBuildings { get; set; }
}

```

Снимка 20: InsuranceBrokerUser.cs код

- MovableProperty.cs – Наследява BaseEntity класа. Описва всички необходими стойности за таблиците с движимо имущество. Стойностите са година на производство, притежание на обезопасителни средства, техническа изправност, изминато

разстояние, височина, тегло, широчина, държава, област, град/село.

```
3 references
public abstract class MovableProperty : BaseEntity
{
    18 references
    public DateTime ManufactureYear { get; set; }
    18 references
    public bool SecurityEquipmenPossession { get; set; }
    18 references
    public bool TechnicalServiceability { get; set; }
    18 references
    public int DistanceTraveled { get; set; }
    18 references
    public double Height { get; set; }
    18 references
    public double Weight { get; set; }
    18 references
    public double Width { get; set; }
    18 references
    public string RegisteredCountry { get; set; }
    18 references
    public string RegisteredRegion { get; set; }
    18 references
    public string RegisteredCity { get; set; }
}
```

Снимка 21: MovableProperty.cs код

- RealEstateProperty.cs - Наследява BaseEntity класа. Описва всички необходими стойности за таблиците с недвижимо имущество. Стойностите са държава, област, град/село, адрес, притежание на пожарогасители, аварийен изход, площ в кв.м, притежание на алармена система/пазач, притежание на газови бутилки.

```
public abstract class RealEstateProperty : BaseEntity
{
    public string Country { get; set; }
    public string Region { get; set; }
    public string City { get; set; }
    public string Address { get; set; }
    public bool FireExtinguishers { get; set; }
    public bool EmergencyExit { get; set; }
    public double SquareFeet { get; set; }
    24 references
    public bool AlarmSystem { get; set; }
    24 references
    public bool GasBottles { get; set; }
}
```

Снимка 22: RealEstateProperty.cs код

- AirTransport.cs – Наследява MovableProperty класа. Добавени са стойностите Име, функционалност и Id на застрахователния потребител.

```
10 references
public class AirTransport : MovableProperty
{
    6 references
    public string Name { get; set; }
    6 references
    public string Functionality { get; set; }
    0 references
    public virtual InsuranceBrokerUser InsuranceBroker { get; set; }
}
```

Снимка 23: AirTransport.cs код

- SeaTransports.cs - Наследява MovableProperty класа. Добавени са стойностите Име, дали транспортът минава през пиратска зона, функционалност, начин на задвижване и Id на застрахователния потребител.

```
10 references
public class SeaTransport : MovableProperty
{
    6 references
    public string Name { get; set; }
    6 references
    public bool DoesRoutePassesPirateZones { get; set; }
    6 references
    public string Functionality { get; set; }
    6 references
    public string TypeOfMovability { get; set; }
    0 references
    public virtual InsuranceBrokerUser InsuranceBroker { get; set; }
}
```

Снимка 24: SeaTransport.cs код

- VehicleByLands.cs - Наследява MovableProperty класа. Добавени са

стойностите вид гориво, притежание на парктроник, най-чести маршрути, регистрационен номер, Id на застрахователния потребител.

```
10 references
public class VehicleByLand : MovableProperty
{
    6 references
    public string FuelType { get; set; }
    6 references
    public bool Parktronic { get; set; }
    6 references
    public string MostCommonRoutes { get; set; }
    6 references
    public string RegisterNumber { get; set; }
    0 references
    public virtual InsuranceBrokerUser InsuranceBroker { get; set; }
}
```

Снимка 25: VehicleByLand.cs код

- BusinessEnterprise.cs - Наследява RealEstateProperty класа. Добавени са стойностите дейност на предприятието и Id на застрахователния потребител.

```
10 references
public class BusinessEnterprise : RealEstateProperty
{
    6 references
    public string PurposeOfTheEnterprise { get; set; }
    0 references
    public virtual InsuranceBrokerUser InsuranceBroker { get; set; }
}
```

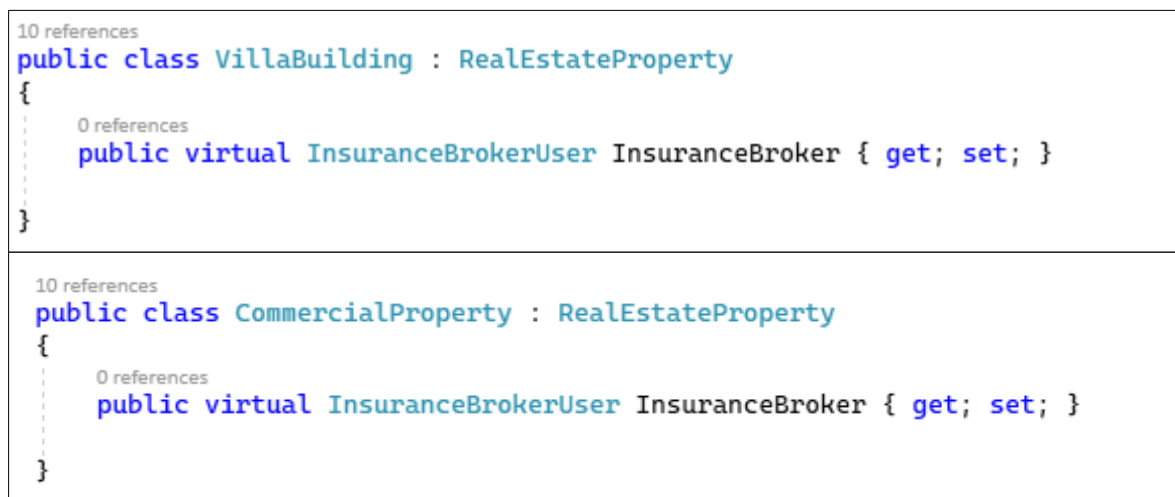
Снимка 26: BusinessEnterprise.cs код

- ResidentialBuilding.cs - Наследява RealEstateProperty класа. Добавени са стойностите етаж и Id на застрахователния потребител.

```
10 references
public class ResidentialBuilding : RealEstateProperty
{
    6 references
    public string Floor { get; set; }
    0 references
    public virtual InsuranceBrokerUser InsuranceBroker { get; set; }
}
```

Снимка 27: ResidentialBuilding.cs код

- CommercialProperty.cs и VillaBuilding.cs наследяват RealEstateProperty класа



```

10 references
public class VillaBuilding : RealEstateProperty
{
    0 references
    public virtual InsuranceBrokerUser InsuranceBroker { get; set; }
}

10 references
public class CommercialProperty : RealEstateProperty
{
    0 references
    public virtual InsuranceBrokerUser InsuranceBroker { get; set; }
}

```

Снимка 28: CommercialProperty.cs и VillaBuilding.cs код

- Repositories – Съдържа Repository.cs файлът. Чрез него и чрез IRepository.cs се имплементира т.нар Repository Design Pattern. Той е един от най-популярните модели за проектиране и за постигане на разделяне между действителната база данни, заявките и друга логика за достъп до данни от останалата част от приложението.

2.5.3 Проектиране на слой за услуги

За всяка основна единица от приложението се създават двойка файлове – интерфейс и имплементация на интерфейса под формата на услуга(service). Класовете – услуги описват методите свързани с бизнес логиката по CRUD операциите и други необходими операции по обектите. Подходът с интерфейсите позволява използването на инжекция на зависимостите(dependency injection) в приложението ,както и лесното последващо компонентно тестване.

2.5.4 Проектиране на презентационен слой

Презентационният слой се състои от папки обвързани с контролерите ,моделите за изгледи , изгледите и имплементацията на потребителския интерфейс.

2.5.4.1 Проектиране на контролери

Контролерите за entity-тата са проектирани да си взаимодействат със service-ите им като са използвани методите намиращи се в service-те. Във всеки един контролер се намират методите Index, Details, Create, Edit и Delete. Чрез тях се осъществява CRUD операциите за съответния обект.

2.5.4.2 Проектиране на модели за изгледи

В папката ViewModels , която се намираща се в Models се намират всички view model-и . За всяко entity са проектирани по 4 view model-a : Add, Details, Edit и Default ViewModel.

2.5.4.4 Проектиране на изгледи

За всяко едно entity са проектирани по 5 изгледа : Index , Create , Edit, Delete и Details. Index изгледа предоставя всички резултати до момента на потребителя. Create изгледа представлява формуляр , която при попълване създава нов резултат за даденото имущество. Details изгледа предоставя всички данни за избраното имущество. Edit изгледа дава възможност за промяна на резултата и информацията на избраното имущество. Главната функция на Delete изгледа и потвърждение за изтриване на обекта.

Проектиран е Home изглед , който представлява главна страница на приложението. От него можеш да се навигираш към Login и Register изгледите ,

които осигуряват регистрацията и входа на потребителя.

2.5.5 Проектиране на алгоритъм за оценка на застрахователният риск

Важно е да се отбележи, че алгоритъмът е изцяло измислен от разработчика на този дипломен проект и за бъдещото развитие на приложението , би било препоръчително съвместната работа с лице от застрахователната индустрия.

Като за начало всяко едно имущество има начална стойност 100. Ако имуществото е имало предишни инциденти се намаля стойността с 5 единици.

При движимото имущество, ако то е произведено преди 5 до 10 години се намаля стойността с 10 единици , от 10 до 15 с 15 единици и над 15 с 20 единици. Ако имуществото не притежава обезопасителни средства се намаля с 15 единици. Ако имуществото не притежава валидна техническа изправност се намаля с 25 единици. Ако са изминати от 100 хил. км. до 200 хил. стойността се намаля с 10 единици , над 200 хил. се намаля с 20 единици.

За движимо имущество от тип въздушен транспорт се въвежда и видът на транспорта. Ако видът е търговски се намаля с 5 единици , военен с 20 единици, товарен с 10 единици.

За движемо имущество от тип воден транспорт, се въвежда видът на транспорта, начинът на задвижване и дали транспорта минава през пиратска зона. Начинът за оценяване на видът е същият като при този за въздушен транспорт. В зависимост начина на задвижване при ветроходен се намаля резултатът с 5 единици , а при моторен с 10 единици.

За движимо имущество от тип пътен транспорт се въвеждат вид гориво, наличие на парктроник и най-чест вид на маршрут. Ако видът на горивото е бензин се стойността се намаля с 10 единици , дизел с 5, бензин + пропан с 15 и електричество с 5. Ако пътният транспорт не притежава парктроник стойността се намаля с 15 единици. Ако видът на маршрута е градски стойността се намаля с 10 единици, ако е междуградски с 5 единици.

При недвижимото имущество, ако не притежава противопожарни средства стойността се намаля с 25 единици. Ако не притежава аварийен изход

се стойността се намалява с 30 единици. Ако не притежава алармена система или охранителна фирма, стойността се намаля с 10 единици. Ако на мястото на обекта се намират газови бутилки , стойността се намаля с 20 единици.

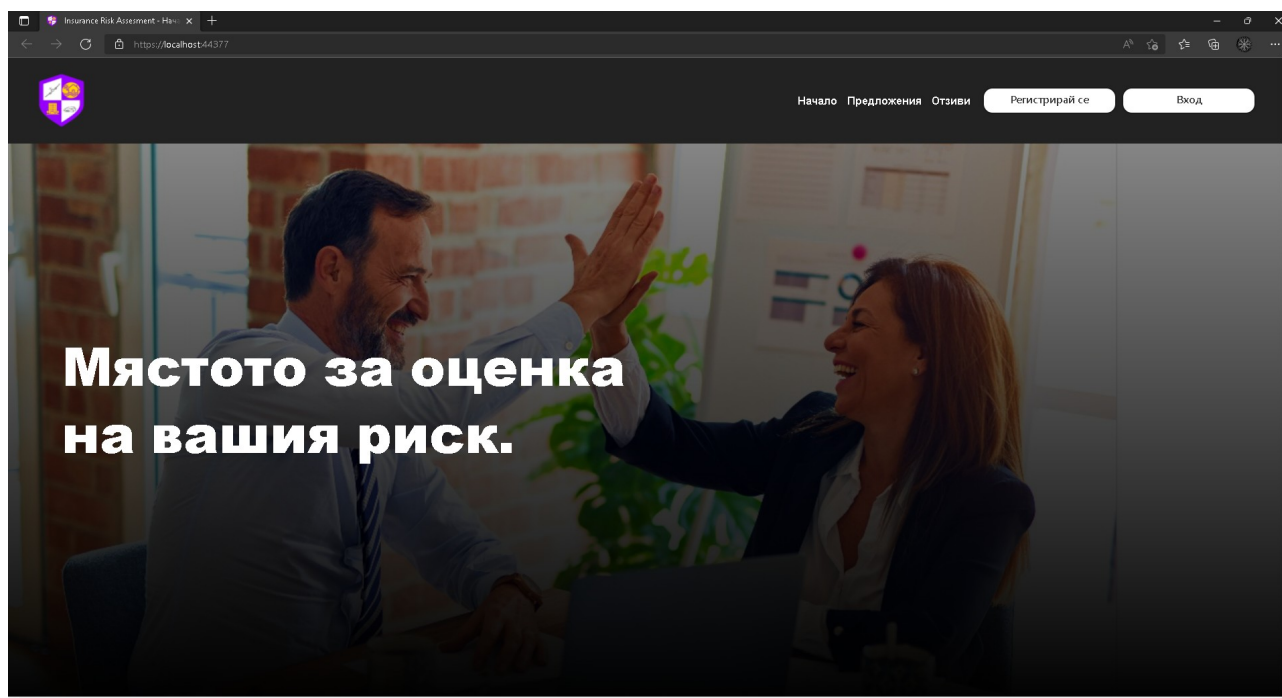
За недвижимо имущество от тип жилищни сгради се въвеждат и видът на етажа. Ако етажът е първи или последен се намалят 5 единици от резултата.

За недвижимо имущество от тип търговски предприятия се въвежда и предназначението на предприятието , ако то е за производство или преработване се намалят 10 единици от резултата.

2.6 Употреба на приложението

При влизане в приложението ще се озовете в началната страница откъдето можете да се навигирате към login(вход) и register(регистрация) страниците.

- Начална страница



Снимка 29: Начална страница

- Login(вход)

The screenshot shows a web browser window with the title "Insurance Risk Assessment - Вход" and the URL "https://localhost:44377/Identity/Account/Login". The page has a dark theme. In the top right corner, there are links for "Начало", "Предложения", "Отзиви", and buttons for "Регистрирай се" and "Вход". The main content area features a login form titled "Влезте с вашият профил:". The form includes input fields for "Имейл:" and "Парола:", a link "Нямате профил?", and a "Влез" button. At the bottom of the page, there is a small disclaimer: "Web приложение за оценка на застрахователен риск, изготвено за защита на дипломна работа, към ПГЕЕ 'К.Фотинид'".

Снимка 30: Вход

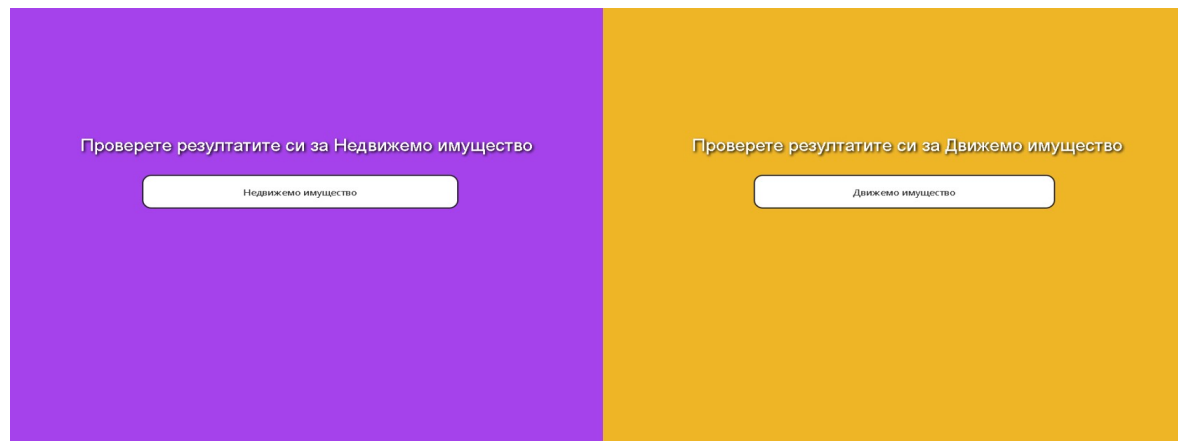
- Register(регистрация)

The screenshot shows a web browser window with the title "Insurance Risk Assessment - Регист" and the URL "https://localhost:44377/Identity/Account/Register". The page has a dark theme. In the top right corner, there are links for "Начало", "Предложения", "Отзиви", and buttons for "Регистрирай се" and "Вход". The main content area features a registration form titled "Създай нов профил:". The form includes input fields for "Имейл:", "Парола:", and "Потвърди парола:", and a "Регистрирай се" button. At the bottom of the page, there is a small disclaimer: "Web приложение за оценка на застрахователен риск, изготвено за защита на дипломна работа, към ПГЕЕ 'К.Фотинид'".

Снимка 31: Регистрация

След успешен вход или регистрация, потребителят вече се намира в навигационна страница за движимо и недвижимо имущество.

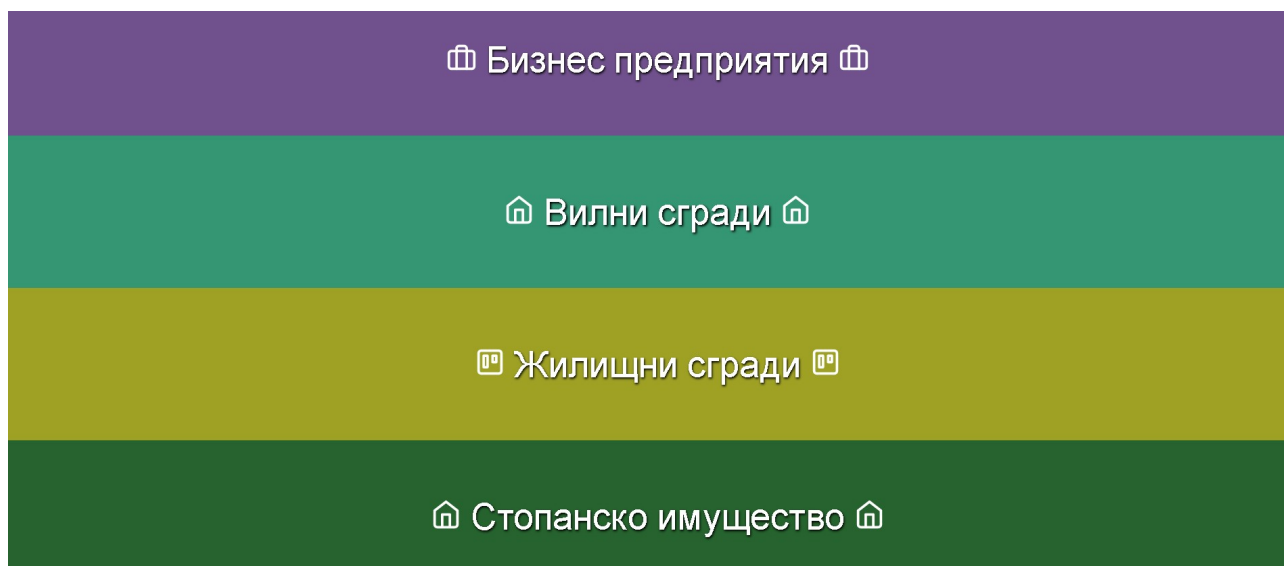
- Навигационна страница



Снимка 32: Навигационна страница

Ако потребителят е избрал да оценява недвижимо имущество , той ще бъде отведен в страница за избор на вид недвижимо имущество.

- Видове недвижимо имущество



Снимка 33: Видове недвижимо имущество

Ако потребителят е избрал да оценява движимо имущество , той ще бъде отведен в страница за избор на вид движимо имущество.

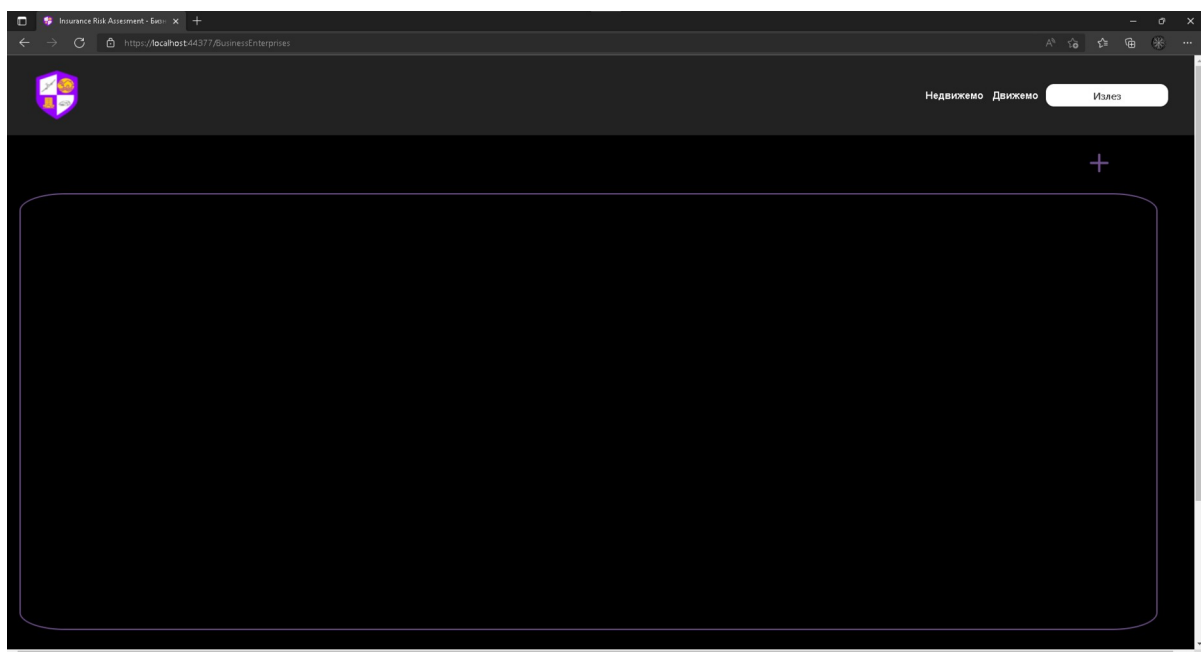
- Видове движимо имущество



Снимка 34: Видове движимо имущество

След като потребителя си е избрал видът на имуществото , той бива отведен в страница с всички текущи резултати за даденото имущество.

- Страница с резултати



Снимка 35: Страница с резултати

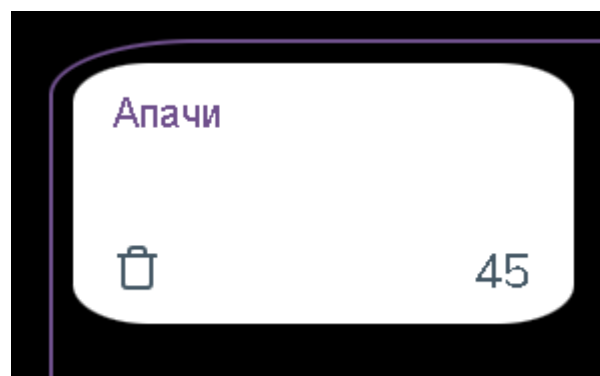
Ако потребителя иска да изчисли риска за ново имущество , при натискане на плюса , той бива навигиран в страница за изчисление и запазване на резултат.

- Страница за изчисление и запазване на резултат

Снимка 36: Страница с изчисление и запазване на резултат

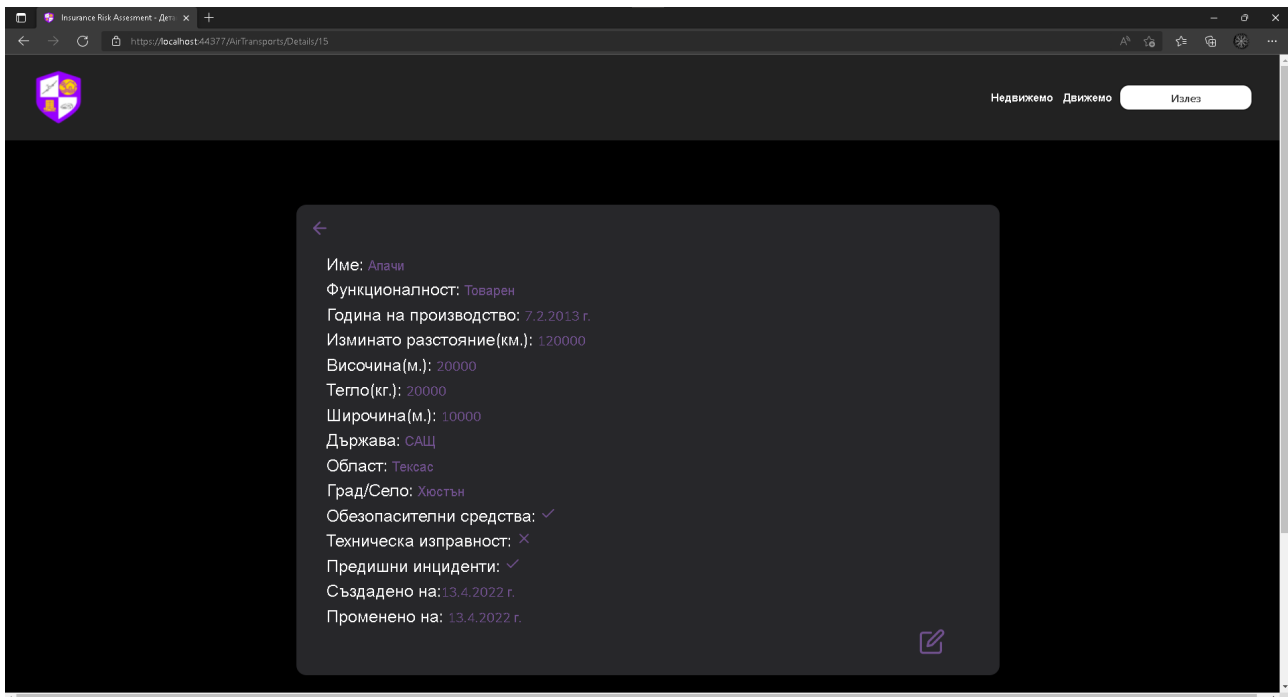
При въвеждане на данните потребителя може да види резултата си до момента при натискане на бутон „Покажи резултат“ и да запази резултата с натискане на бутон „Запази резултат“.

Ако потребителя иска да види детайлите за въведеното имущество , при натискане на бялата плочица , потребителят бива отведен към страница с детайлите на имуществото.



Снимка 37: Плочица с резултат

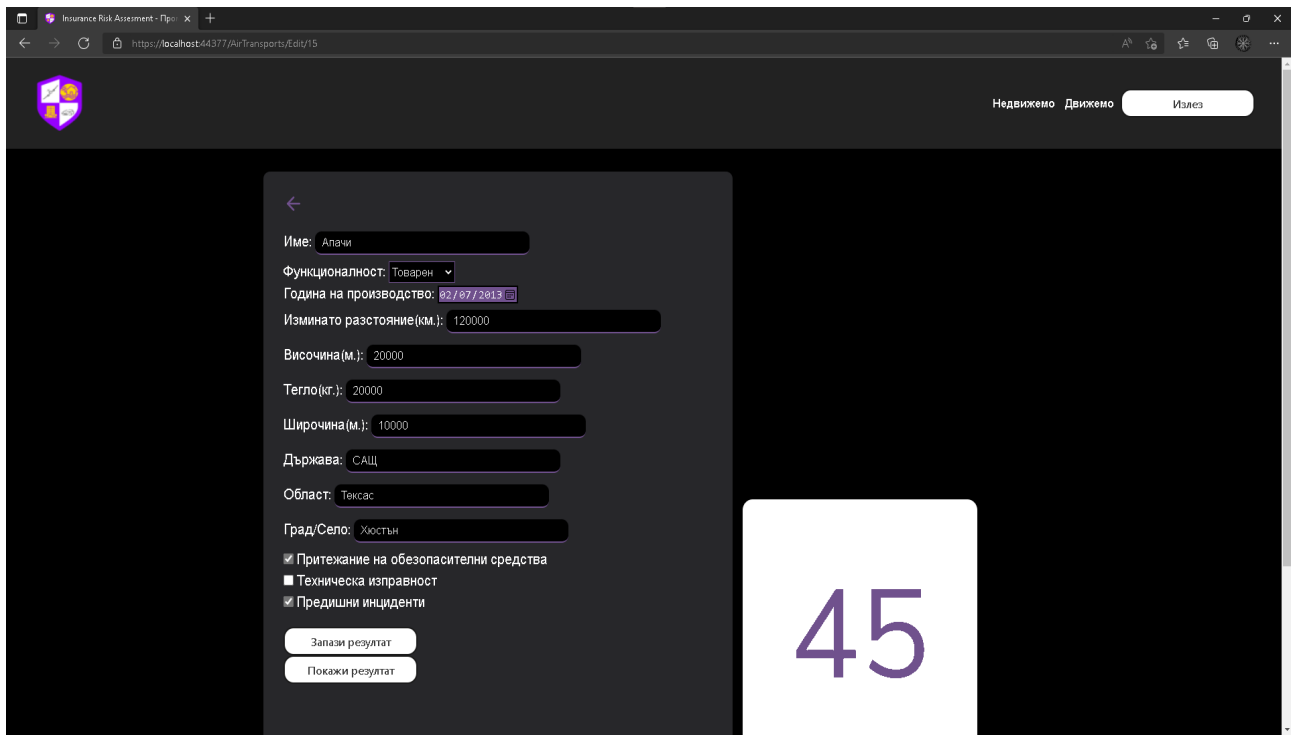
- Страница с детайли



Снимка 38: Страница с детайли

Ако потребителя иска да промени данните за имуществото при натискане на иконката долу в дясно той бива отведен в страница за промяна на данните.

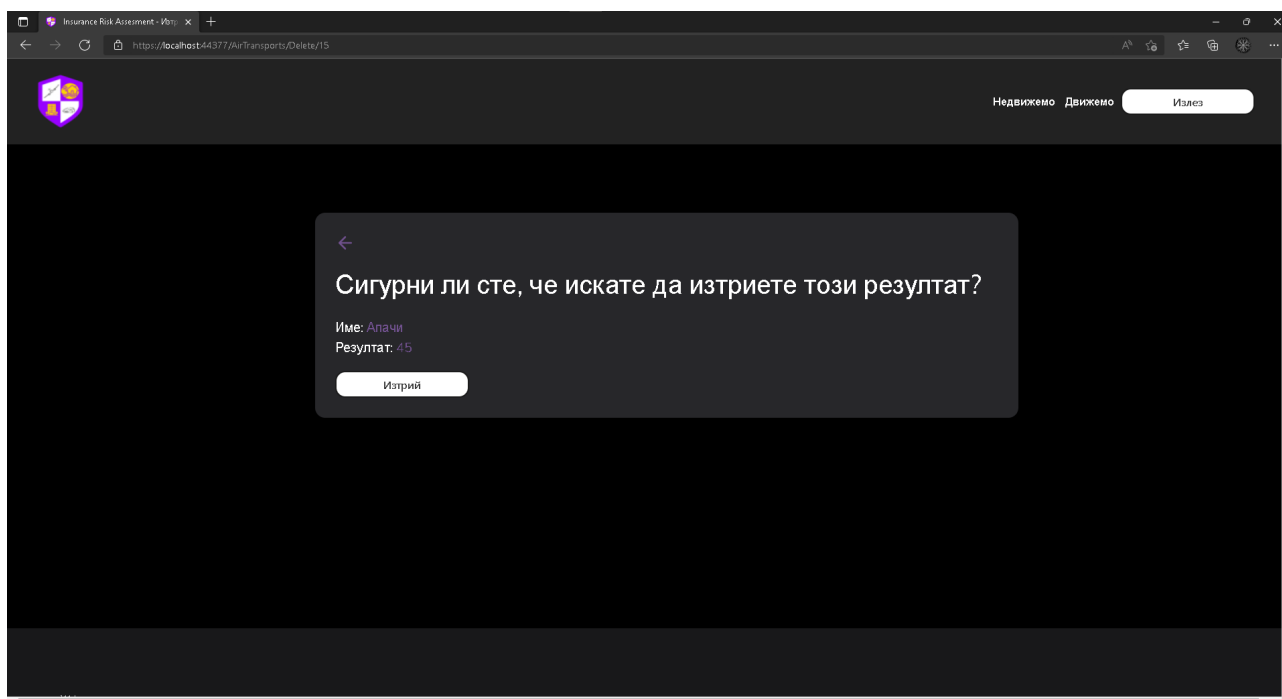
- Страница за промяна на данните



Снимка 39: Страница за промяна на данните

Ако потребителя иска да изтрие резултат за дадено имущество , той може да го направи с натискане на иконката за кошче , намираща се върху бялата плочица.

- Страница за потвърждение за изтриване



Снимка 40: Страница за потвърждение за изтриване

3. Заключение

В рамките на дипломния проект бе показано създаване на приложение с цел улеснение преценката на служителите в застрахователния сектор. Проектът може да се доразвие чрез разширение и оптимизация на начина за намиране на точен и полезен резултат. Може да се добави и изкуствен интелект подпомагащ за предсказание на бъдещи рискове за дадения обект. Освен уеб приложение , може да се разработи и мобилно с цел по-лесна употреба.

4. Информационни източници

Уеб приложение

https://bg.wikipedia.org/wiki/Уеб_приложение

<https://bg.education-wiki.com/7160560-what-is-web-application>

ASP.NET Core MVC

<https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-6.0>

C#

<https://docs.microsoft.com/en-us/dotnet/csharp/>

HTML

<https://developer.mozilla.org/en-US/docs/Web/HTML>

CSS

<https://developer.mozilla.org/en-US/docs/Web/CSS>

JAVASCRIPT

<https://developer.mozilla.org/en-US/docs/Web/JavaScript>

5. Приложения

Github repository съдържащо source код-а на приложението

<https://github.com/Velin1234/Insurance-risk-assessment>

6. Рецензия на дипломен проект

Професионална гимназия по електротехника и електроника „Константин
Фотинов“, гр. Бургас

РЕЦЕНЗИЯ

Тема на дипломния проект			
Ученик			
Клас			
Професия			
Специалност			
Ръководител- консултант			
Рецензент			
Критерии за допускане до защита на дипломен проект	Да	Не	
Съответствие на съдържанието и точките от заданието			
Съответствие между тема и съдържание			
Спазване на препоръчителния обем на дипломния проект			
Спазване на изискванията за оформление на дипломния проект			
Готовност за защита на дипломния проект			
Силни страни на дипломния проект			
Допуснати основни слабости			
Въпроси и препоръки към дипломния проект			

ЗАКЛЮЧЕНИЕ:

Качествата на дипломния проект дават основание ученикът/ученичката..... да бъде допуснат/а до защита пред членовете на комисията за подготовка, провеждане и оценяване на изпит чрез защита на дипломен проект- част по теория на професията.

.....05.2022г.

Гр. Бургас

Рецензент:.....