## ЧАСТНА ПРОФЕСИОНАЛНА ГИМНАЗИЯ ПО ДИГИТАЛНИ НАУКИ “СОФТУНИ СВЕТЛИНА“

# Дипломна работа

ЗА ПРИДОБИВАНЕ НА КВАЛИФИКАЦИЯ

НА ТЕМА:

КАЛКУЛАТОР ЗА ОПРЕДЕЛЯНЕ НА ПРЕМИЯТА

ПО ЗАСТРАХОВКА ГРАЖДАНСКА ОТГОВОРНОСТ НА АВТОМОБИЛИ

ИЗГОТВИЛ: НАУЧЕН РЪКОВОДИТЕЛ:

ЕЛТИМИР СТЕФАНОВ СТОИЛКОВ МАРИЯН ЙОРДАНОВ

28.02.2023

ГР.СОФИЯ

# Съдържание

[Дипломна работа 1](#_Toc128640107)

[Съдържание 2](#_Toc128640108)

[Увод 3](#_Toc128640109)

[Проблем 4](#_Toc128640110)

[Цели на дипломния проект 4](#_Toc128640111)

[Задачи, произтичащи от целите на дипломната работа 5](#_Toc128640112)

[Глава 1. Проучване 6](#_Toc128640113)

[Глава 2. Задание 11](#_Toc128640114)

[Глава 3. Проектиране и имплементация 12](#_Toc128640115)

[3.1. Архитектура на системата 14](#_Toc128640116)

[3.2. Имплементация на системата 17](#_Toc128640117)

[3.3. Тестове 24](#_Toc128640118)

[3.4. Внедряване (deployment) 26](#_Toc128640119)

[Глава 4. Ръководство за потребителя 27](#_Toc128640120)

[Заключение 29](#_Toc128640121)

[Информационни източници 30](#_Toc128640122)

[Рецензия на дипломен проект 31](#_Toc128640123)

# Увод

Дипломният проект представя разработка на калкулатор за определяне на премията по застраховка Гражданска отговорност на автомобили. Предложеният калкулатор дава възможност на лицата, които желаят за сключат застраховка, дистанционно (online) да проверят цената, на която могат да се застраховат, без да бъде необходимо да ходят в офис на застраховател. Калкулаторът може да бъде достъпен чрез интернет сайта на застраховател или застрахователен поредник.

Застраховка Гражданска отговорност на автомобили е задължителна и покрива отговорността на водача за причинени от него на трети лица имуществени и неимуществени вреди, които са резултат от застрахователното събитие1. Това законово изискване води до необходимост всеки собственик на моторно превозно средство да сключи застрахователен договор. Предвид че полицата покрива период от дванадесет месеца, се налага всяка една година да се сключва нов договор и да се посети офис на застраховател или застрахователен посредник. Настоящият калкулатор дава възможност за спестяване на времето за проучване на цената и посещение на застрахователен посредник. При въвеждане на данни за собственика и моторното превозно средство (МПС) се получава премия, представляваща цената, по която може да се сключи застраховката. Премията се определя в зависимост от характеристики на водача и застрахования автомобил. Основните фактори, спрямо които застрахователите в България определят своите цени, са свързани с:

а) моторното превозно средство като: вида на МПС; мощност или обем на двигателя (за леки автомобили тази характеристика се измерва чрез конски сили или кубатура на двигателя); начин на ползване (за лични нужди, производствена и търговска дейност, такси, учебни автомобили); район, в който се управлява автомобила;

б) собственика - възраст на водача; налични предишни пътно-транспортни произшествия; шофьорски стаж.

За целите на настоящата разработка са ползвани следните ценообразуващи фактори, спрямо които е определена премията: вид на МПС; регион на движение; начин на ползване; възраст на собственика.

На база реални данни на застрахователна компания за сключени полици по застраховка Гражданска отговорност на автомобили, стойност на премийния приход; данни за МПС и

1 Кодекс за застраховането, чл. 429, ал.1, т.1

собственика; информация за броя и стойността на платените и предявени, неизплатени претенции по всяка една полица, се извършва калкулация на необходимата премия за покритие на плащанията във връзка с риска. Тази премия се увеличава с разходи на застрахователя, за да се определи брутната премия, съответстваща на превозно средство с определени зададени характеристики.

## Проблем

Чрез предложения калкулатор за изчисление на премията по застраховка Гражданска отговорност на автомобили, потребителите на тази услуга ще имат възможност дистанционно да проверят условията, при които могат да сключат полица с дадена компания. Това ще спести време за посещение на офис и възможност за съпоставка на цените на различни компании и вземане на информирано решение.

Предлаганото решение е и в полза на застрахователните компании, предвид че така по-лесно могат да стигнат до своите клиенти и да улеснят сключването на застраховка. При използване на калкулатора се улеснява и самия застраховател, тъй като се избягва процеса по въвеждане на данните на клиента, тяхната обработка и изготвяне на оферта.

От предлаганото решение могат да се възползват всички собственици на МПС, чийто брой на територията на България е над 3 милиона. Застраховката е задължителна за всяко едно превозно средство. Предвид че покритието по полицата е дванадесет месеца и необходимостта от подновяване възниква поне веднъж годишно. Потенциални потребители на това решение са множество лица, които могат да проверят цената на застраховката в сайта на застрахователя, като не е задължително да се стигне до сключване на полица.

## Цели на дипломния проект

Целта на настоящата дипломна работа е да се разработи калкулатор за определяне на премията по застраховка Гражданска отговорност на автомобили и предоставянето му за ползване на масовия потребител чрез REST API и браузър базиран интерфейс.

## Задачи, произтичащи от целите на дипломната работа

Във връзка с поставената цел за разработване на технологията за калкулация на гражданска отговорност в уеб среда са поставени следните задачи:

* Извършване на проучване и подготовка на обзор на проблемната област
* Проучване на инструментите и технологиите за работа с REST API и уеб базиран интерфейс с .Net и JavaScript
* Проектиране на система за калкулация на Гражданска отговорност на автомобили в уеб среда
* Реализация на системата за калкулация на Гражданска отговорност на автомобили в уеб среда:
* клиентска част - браузър базирано приложение, визуализиращо параметрите и резултатът от калкулацията,
* сървърна част - уеб приложение за предоставяне на необходимите номенклатури и извършване на калкулацията.
* Извеждане на изводи и заключения на базата на разработения проект.

# Глава 1. Проучване

Разработеният калкулатор определя премията на база данни за сключените от застраховател на територията на България полици по застраховка Гражданска отговорност на автомобили и изплатените и предявени, неизплатени претенции по всяка една полица.

Извършено е проучване на характеристиките на МПС и водача, спрямо които застрахователите в България определят цената на застраховката. Разгледани са сайтове на застрахователи, предлагащи застраховката, както и интернет страници на застрахователни брокери1.

Избрани са следните характеристики, спрямо които е определена премията в предложения калкулатор:

1. Вид на моторното превозно средство
2. Населено място на регистрация на МПС
3. Начин на ползване
4. Възраст на собственика / водача, посочен в полицата

Освен горепосочените характеристики, в данните, на база на които е извършено изчислението на премията се съдържа и информация за номер на полица, дата на сключване, срок на действие – начална и крайна дата на покритие, тип МПС, тип МПС по тарифа (обем на двигателя), предназначение, населено място, брой щети, стойност на платените щети и стойност на висящите щети (предявени, неизплатени претенции).

Данните са в json формат и представляват таблица със следното съдържание:



1 <https://www.sdi.bg/onlineinsurance/calculator.php>; https://[24ins.bg](https://24ins.bg/mtpl)/mtpl

https://www.online.ozk.bg/grazhdanska-otgovornost/kalkulator

Данните са групирани в зависимост от факторите, от които се определя размера на премията:

1. Вид на превозното средство
   1. Леки автомобили с обем на двигателя до 1 500 куб.см
   2. Леки автомобили с обем на двигателя от 1 500 куб.см. до 2 000 куб.см.
   3. Леки автомобили с обем на двигателя над 2 000 куб.см.
   4. Мотоциклети и АТВ
2. Населено място, в което е регистрирано и се управлява МПС. За целите на настоящата дипломна работа и разработвания калкулатор са използвани данни само за полици, сключени за превозни средства в населените места, съгласно следния списък:

|  |
| --- |
| Благоевград |
| Бургас |
| Варна |
| Велико Търново |
| Кюстендил |
| Пловдив |
| Русе |
| София |
| Стара Загора |
| Други |

1. Начин на ползване
   1. Лични нужди
   2. Производствена и търговска дейност
   3. Такси / Учебен
2. Възраст на собственика / водача, посочен в полицата. Възрастта се използва като фактор при определяне на премията само за леки автомобили, използвани за лични нужди
   1. От 18 до 20 години
   2. От 21 до 30 години
   3. От 31 до 50 години
   4. Над 50 години

Размерът на премията е определен на база честотата на събития (вероятността да настъпи щета) и размера на загубата в случай на събитие (средната стойност на една щета)1.

Рискова премия = честотата \* средния размер на щетата

Честотата = брой щети / брой застраховани МПС

Средния размер на щета = сума на платените и висящи щети / броя на щетите

Така на база параметрите, от които зависят честотата и средната стойност на една щета, се получава съкратена формула за рисковата премия, при изчислението на която не участват броя на щетите, а само сумата на щетите и броя застраховани МПС. По този начин необходимата премия за покритие на риска се определя чрез средната стойност на щетите, изплатени за едно застраховано превозно средство.

Рискова премия = (стойност на платените + висящи щети) / брой застраховани МПС

Брутна премия = Рискова премия + Добавка за разходи

Пример за определяне на рисковата премия за мотоциклети и АТВ по населени места, като произведение на честотата и средния размер на щетите е посочен в следващата таблица:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Населено място | Брой полици | Брой щети | Сума висящи в лв. | Сума платени в лв. | Стойност на щетите | Честота | Среден размер на щетите | Рискова премия |
| Благоевград | 691 | 3 | 0 | 20 643 | 20 643 | 0,434% | 6 881 | 30 |
| Бургас | 2 110 | 11 | 0 | 72 854 | 72 854 | 0,521% | 6 623 | 35 |
| Варна | 1 273 | 10 | 0 | 45 567 | 45 567 | 0,786% | 4 557 | 36 |
| Велико Търново | 1 082 | 4 | 0 | 38 653 | 38 653 | 0,370% | 9 663 | 36 |
| Други | 3 190 | 8 | 0 | 106 926 | 106 926 | 0,251% | 13 366 | 34 |
| Кюстендил | 731 | 3 | 0 | 23 308 | 23 308 | 0,410% | 7 769 | 32 |
| Пловдив | 2 158 | 12 | 2 934 | 86 811 | 89 745 | 0,556% | 7 479 | 42 |
| Русе | 416 | 4 | 0 | 13 963 | 13 963 | 0,962% | 3 491 | 34 |
| София | 2 150 | 14 | 4 890 | 79 872 | 84 761 | 0,651% | 6 054 | 39 |
| Стара Загора | 1 341 | 7 | 0 | 43 903 | 43 903 | 0,522% | 6 272 | 33 |
| Общо | 15 142 | 76 | 7 823 | 532 500 | 540 323 | 0,502% | 7 110 | 36 |

*Данните са в лева*

Данните, на база на които е определена рисковата премията за леки автомобили са представени по населени места в следващата таблица:

1 „Актюерство“, доц. д-р Йото Йотов, Академично издателство „Ценов“ Свищов, 2000 г.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Населено място | Брой полици | Брой щети | Сума висящи в лв. | Сума платени в лв. | Стойност на щетите | Честота | Среден размер на щетите | Рис-кова пре-мия |
| Благоевград | 20 069 | 447 | 125 318 | 2 444 008 | 2 569 326 | 2,227% | 5 748 | 128 |
| Бургас | 16 913 | 439 | 194 363 | 2 413 587 | 2 607 950 | 2,596% | 5 941 | 154 |
| Варна | 16 851 | 453 | 70 201 | 2 692 765 | 2 762 965 | 2,688% | 6 099 | 164 |
| Велико Търново | 3 214 | 77 | 22 800 | 400 165 | 422 965 | 2,396% | 5 493 | 132 |
| Други | 14 842 | 364 | 115 654 | 1 704 135 | 1 819 790 | 2,452% | 4 999 | 123 |
| Кюстендил | 10 882 | 243 | 24 342 | 1 523 433 | 1 547 775 | 2,233% | 6 369 | 142 |
| Пловдив | 29 764 | 887 | 39 358 | 5 244 991 | 5 284 349 | 2,980% | 5 958 | 178 |
| Русе | 6 096 | 225 | 13 953 | 843 381 | 857 333 | 3,691% | 3 810 | 141 |
| София | 53 713 | 1 790 | 364 870 | 9 493 497 | 9 858 367 | 3,333% | 5 507 | 184 |
| Стара Загора | 8 878 | 224 | 11 080 | 1 218 042 | 1 229 122 | 2,523% | 5 487 | 138 |
| Общо | 181 222 | 5 149 | 981 939 | 27 978 004 | 28 959 943 | 2,841% | 5 624 | 160 |

*Данните са в лева*

Полиците могат да бъдат групирани на по-детайлно ниво – например премията за 3-та възрастова група, леки автомобили над 2000 куб.см., използвани за лични нужди, по отделни населени места, е получена на база следните данни:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| възрастови групи | 3 |  |  |  |  |  |  |  |
| Тип МПС по тарифа | Лек автомобил над 2000 см3 обем на двигателя | | | | | | | |
| Предназначение на МПС | Лични нужди | | | | | |  |  |
|  |  |  |  |  |  |  |  |  |
| Населено място | Брой полици | Брой щети | Сума висящи в лв. | Сума платени в лв. | Стойност на щетите | Честота | Сре-ден размер на щетите | Рис-кова пре-мия |
| Благоевград | 3 751 | 100 | 12 811 | 615 922 | 628 733 | 2,666% | 6 287 | 168 |
| Бургас | 2 940 | 94 | 1 854 | 591 186 | 593 040 | 3,197% | 6 309 | 202 |
| Варна | 2 787 | 98 | 311 | 578 262 | 578 573 | 3,516% | 5 904 | 208 |
| Велико Търново | 439 | 11 | 4 089 | 73 219 | 77 308 | 2,506% | 7 028 | 176 |
| Други | 2 145 | 60 | 0 | 332 623 | 332 623 | 2,797% | 5 544 | 155 |
| Кюстендил | 1 904 | 62 | 4 890 | 359 233 | 364 122 | 3,256% | 5 873 | 191 |
| Пловдив | 4 630 | 163 | 0 | 954 898 | 954 898 | 3,521% | 5 858 | 206 |
| Русе | 1 200 | 56 | 1 786 | 201 399 | 203 186 | 4,667% | 3 628 | 169 |
| София | 9 774 | 407 | 107 738 | 1 943 395 | 2 051 133 | 4,164% | 5 040 | 210 |
| Стара Загора | 1 407 | 30 | 0 | 230 519 | 230 519 | 2,132% | 7 684 | 164 |
| Общо | 30 977 | 1 081 | 133 478 | 5 880 657 | 6 014 135 | 3,490% | 5 563 | 194 |

Рисковата премия е завишена с административни разходи и разходи за сключване на договорите, определени чрез коефициент от 1,3, приложен към премия за покритие на риска, чието изчисление е представено по-горе.

Например при рискова премия от 200 лв. брутната премия, която ще се сключи полицата, ще бъде в размер на 260 лв. (260 = 200 \* 1,3).

В този пример застраховката ще бъде продадена за 260 лв., като 200 лв. от тях ще бъдат предназначени за покритие на плащания по щети и 60 лв. за покритие на разходите във връзка със сключване и управление на договора.

# Глава 2. Задание

Поставяме си за цел да създадем уеб базирана система, която изчислява премията по застраховка Гражданска отговорност на автомобили. За целта системата съдържа база данни, бекенд и фронтенд компоненти, като за всяка от тях е избрана подходяща за целта технология:

* **Frontend** – Използваната технология, която показва резултатите е jQuery. Този избор е направен съобразявайки сложността на разработка и времето, за което трябва да се изготви. jQuery предлага много удобен и познат интерфейс за разработка на нови проекти.
* **Backend** – За бекенд е избран технологичния стек на .Net Core поради множеството от предимствата, които предлага спрямо своите конкуренти. Езикът е статично типизиран и това ни позволява по-голяма сигурност при различни операции, съвременен и разполага с удобствата за разработка на код отговарящи на съвременните езици и предлагащ удобен за ползване интерфейс за асинхронни операции, което ни помага при по-голям обем от входящи заявки.
* **Database** – MS SQL Server, в който се държат необходимите данни и се записват новите калкулации. Избран е този сървър, тъй като той най-добре подхожда на технологичният стек, включващ .Net Core приложение.

# Глава 3. Проектиране и имплементация

**а/ Части на приложението**

* Frontend
* Backend
* Database

За frontend ще се използва framework на JavaScript – jQuery. Чрез библиотеката ще заредим нужните номенклатури за изчислението на премията. Проекта се състой от: index.html, style.css и main.js. Основната структура на сайта е в index.html. Стилизацията на всички полета се намира в style.css, а основната функционалност се намира в main.js.

За backend ще се използва framework ASP.NET Core с език за програмиране C#, с който се имплементира REST API, а за база се използва SQL Server. Backend частта на приложението отговаря за предоставяне на необходимите номенклатури, за приемане и обработване на необходимите параметри, изчисляването на премията по Гражданска отговорност на автомобили и връщане на стойността към клиентската част.

*Diagram

Description automatically generated*

**б/ Frontend**

Front end частта се състои от три файла:

* 1. Index.html
* 2. Style.css
* 3. Main.js

В index.html е сама структура на страницата. В началото са сложени всички библиотеки, от които имам нужда. В моя случай съм използвал jQuery версия 3.6.1. Използвам съм и още една библиотека, която е обвързана с модалния прозорец, който показва резултата от изчисленията.

В style.css за всички стилове за формата. Използвал съм няколко класове, както и идентификатор.

В main.js е основната функционалност, имплементирана чрез jQuery. Има няколко заявки, които се използват за зареждане на всички номенклатури като (тип МПС, населено място, предназначение, обем на двигателя и възраст). За да вложа резултатите от тези номенклатура, използвам идентификатор, които избирам от възможните негови опции.

**в/ Backend**

Backend частта се състои от многослойна архитектура с три нива – презентационен, бизнес и достъп до базата данни. Всеки от тези нива е създаден в отделен C# проект. Бизнес проектът зависи от този за достъп до базата данни, а презентационният - от този за бизнес. Така няма циклични референции и се постига всеки проект да отговаря за конкретно нещо.

Презентационният слой отговаря за това да приема информация от клиента и да я подава към бизнес слоя. Също така, след като информацията от бизнес слоя е обработена, тя се подава обратно към презентационния слой, който се грижи тази информация да бъде върната на браузър клиента.

Бизнес проектът съдържа всичката логика обвързана със самата калкулация на дадена гражданска отговорност. Този слой приема от презентационния необходимата информация и валидира дали подадените параметри са валидни според бизнес изискванията. След това този слой изисква необходимата му информация за калкулацията от слоя за достъп до базата данни на база на подадените входни параметри, и по-точно извличане на коефициентите от отделните номенклатури и извличане на статистика за изплатени и чакащи полици. При успешно събиране на цялата необходима информация, започва процесът на калкулация, от който се получава финалната премия. Резултатът се записва с коефициентите, използвани при сметката му в базата, а резултатът се връща към клиента.

Проектът за достъп до базата съдържа абстракцията за достъп до самия SQL сървър. Това е имплементирано чрез Repository Pattern, който ни позволява в бъдеще да можем да подменяме слоя за достъп до базата с фалшив такъв за определени тестове. Това ни освобождава от нуждата да имаме подготвена тестова база. Допълнително, при разделяне на този слой от бизнес слоя, се отделя и логиката за достъп до базата и бизнес логиката. Така бизнес процесът не зависи от къде получава необходимата си информация.

## 3.1. Архитектура на системата

Фронтенд-ът се състои от html, CSS и JavaScript част, като е изпълнен като една отделна страница. Той консумира предоставените REST Api-та от бекенд-а ни и визуализира необходимите падащи менюта, от които клиента избира входните си параметри. След това е отговорен за изпращането на необходимите данни към бекенд-а. При успешен отговор (Http status OK 200) от нашето API, то той визуализира и резултатът от калкулацията. Всичко това се получава при комуникацията на фронтенд-а и презентационната част на бекенд-а.

Презентационната част се състои от контролери, които имат за цел да приемат пристигащите HTTP заявки и да връщат на същите отговор. Те определят и пътя и методите на всеки един ендпоинт, съвкупността ,от които определя нашия приложен програмен интерфейс (API). Там се извличат параметрите необходими на бизнес слоя за да определените изчисления.

Създадени са контролери, които връщат необходимите номенклатури и контролер за калкулация. Тези за номенклатури се ползват от фронтенд-а за получаване на необходимите падащи менюта, които избира клиента, а тези за калкулация – за същинската сметка на премията за застраховка Гражданска отговорност на автомобили.

За събиране необходимата информация за номенклатурите имаме следните контролери AgeGroupsController, MunicipalityController, PurposesController, TariffTypeController, VehicleTypesController. Те предоставят методи за вземане на всички стойности от тях, и методи за вземана на стойност по идентификатор.

Калкулацията се извършва в контролер CalculationsController, който предоставя опцията да се видят предходни калкулации, както и да се създаде нова такава от подадените параметри.

Важно е да се отбележи, че няма контролери, които да зависят от други контролери. Всеки контролер е обвързан само с подчинения си сървис.

След като е приета заявка от даден контролер и са форматирани коректно входните данни, те се подават на бизнес слоя, който е представен от различни сървиси (Services). Те отговарят на разнообразните бизнес изисквания, а в случая изчисляват премията по застраховка Гражданска отговорност за автомобили.

Сървисите отново се делят на такива, които отговарят за номенклатурите, и такива, които се отнасят за калкулацията. За номенклатурите, това са AgeGroupService, MunicipalityService, PurposeService, TariffTypeService, VehicleTypeService. Те вземат необходимата информация от слоя за достъп до базата и я предоставят на презентационния слой.

CalculationService държи основната част от бизнес логиката и в него се случват същинските калкулации. Той съдържа алгоритъма и бизнес валидациите. Този сървис зависи от повече от едно хранилище, тъй като са му необходими по-голям обем от данни за да изпълни своето предназначение. За това той има достъп и до други такива.

Останалите сървиси употребяват хранилищата(repositories) необходими само за неговите цели.

Бизнес слоя зависи от своя страна на хранилищата(repositories), които служат за комуникация с базата данни. Този слой ни осигурява лесен достъп чрез интерфейси към информацията в базата и възможност да записваме нови данни в нея. Чрез него се изграждат самите SQL заявки до базата данни, подържа се връзката с нея, изпращането на заявките, получаването на отговор и преработката на информацията в удобните за ползване от наша страна класове.

По-долу може да видите схема на всички контролери, сървиси и хранилища и техните зависимости.

Diagram

Description automatically generated

*Фигура 1. Схема на слоевете в приложението*

От своя страна хранилищата комуникират директно с SQL сървърът, който е хранилището на данни на нашето приложение. Тъй като нашата информация има определен формат, е много удобно бизнес логиката да бъде моделирана като таблици в релационна база данни, какъвто е и SQL Server. Така се изграждат ясно и недвусмислено зависимостите между отделните таблици или бизнес модели, като между тях се поставят foreign key-ове.

## 3.2. Имплементация на системата

Системата се състои от част фронтенд, бекенд и база данни.

**а/ Фронтенд**

За фроненд е имплементирано браузър базирано приложение с JavaScript и по-точно jQuery. По-долу можете да видите шаблонът на Html файлът, от който стартира приложението. То от своя страна си взимама необходимите стилове от styles.css файлът и инструкциите за изпълнение от main.js файлът.

<!DOCTYPE html>

<html>

<head>

<title>Проект 1</title>

<link rel="stylesheet" href="style.css">

<link href="https://code.jquery.com/ui/1.10.4/themes/ui-lightness/jquery-ui.css" el="stylesheet">

</head>

<body id="selector">

…

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.1/jquery.min.js"></script>

<script src="https://code.jquery.com/jquery-1.10.2.js"></script>

<script src="https://code.jquery.com/ui/1.10.4/jquery-ui.js"></script>

<script src="main.js"></script>

</body>

</html>

*Фигура 2. Схема на html файлът*

**б/ Бекенд**

Бекендът, както споменахме по-горе се състои от три основни проекта и проект за тестове. Приложението се стартира от презентационния проект Vehicles.Web. В този проект се намира конфигурация на нашия бекенд и регистрацията на всички наши сървиси и хранилища. Това ни позволява да се възползваме от удобството на Dependency Injection чрез конструктора предоставен от ASP NET Core. По този начин всички инстанции на необходимите ни класове се създават автоматично на база на налични обекти в контейнра на зависимости (IoC Container). Като пример, такива са всички контролери, сървиси и хранилища. Тъй като тези обекти, които се съдържат в IoC контейнерът имат и време на живот, ние сме избрали те да бъдат валидни или активни само за една HTTP заявка, или да бъдат “Scoped”. Това ни позволява да можем да използваме нови обекти за съвисите, контролерите и хранилищата за всяка една заявка. По този начин се улеснява работата при конкурентен достъп до информацията от приложението. В следния откъс от класа Program.cs може да се видят тези конфигурации:

public class Program  
{  
 private static void *Main*(string[] args)  
 {  
 var builder = WebApplication.*CreateBuilder*(args);  
 *ConfigureServices*(builder.Services, builder.Configuration);  
 var app = builder.Build();  
 *Configure*(app);  
  
 app.Run();  
  
 }  
  
 private static void *ConfigureServices*(IServiceCollection services, IConfiguration configuration)  
 {  
 services.AddDbContext<VehicleContext>(  
 options => options.UseSqlServer(configuration.GetConnectionString("DefaultConnection")));  
  
   
 services.AddSingleton(configuration);  
 services.AddControllers();  
  
 // Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle  
  
 services.ConfigureAutomapper();  
 services.AddEndpointsApiExplorer();  
 services.AddSwaggerGen();  
  
 //register repositories   
 services.AddScoped<IMunicipalityRepository, MunicipalityRepository>();  
 services.AddScoped<IPurposeRepository, PurposeRepository>();

…  
 services.AddScoped<ICalculationRepository, CalculationRepository>();  
  
 //register services  
 services.AddScoped<ICalculationService, CalculationService>();  
 services.AddScoped<IPurposeService, PurposeService>();  
 …  
 services.AddScoped<IVehicleTypeService, VehicleTypeService>();  
  
 }

…  
}

*Фигура 3. Откъс от Program.cs за регистрация на сървиси*

Също така, е необходимо да се позволи REST Api-то да бъде викано от различен домейн, тоест CORS рестрикциите са изключени.

private static void *Configure*(WebApplication app) {  
 …  
 app.UseCors(options =>  
 options  
 .AllowAnyOrigin()  
 .AllowAnyMethod()  
 .AllowAnyHeader()  
 );  
 …  
 }

*Фигура 4. Откъс от Program.cs за конфигурация на CORS*

Както споменах по-горе, използвам Dependency Injection за инстанцирането на отделните обекти. За да е възможно това, контролерът трябва да бъде означен с атрибут [ApiController] и да наследява от ControllerBase. Също така самият контролер оказва началния път на всички негови методи с атрибутът [Route]. Неговите методи могат да доуточнят този път отново с атрибута [Route].

[Route("api/**[controller]**")]  
[ApiController]  
public class CalculationsController : ControllerBase  
{  
 private readonly ICalculationService \_vehicleService;  
  
 public CalculationsController(ICalculationService vehicleService)  
 {  
 \_vehicleService = vehicleService;  
 }  
  
 [HttpPost]  
 public async Task<CalculationResponse> CalculateAsync(CalculationRequest request)  
 {  
 CalculationResponse response = await \_vehicleService.CalculateAsync(request);  
 return response;  
 }  
  
 [HttpGet]  
 public async Task<PageContainer<CalculationResponse>> GetAllAsync([FromQuery] QueryParameters parameters)  
 {  
 PageContainer<CalculationResponse> responses = await \_vehicleService.GetAllAsync(parameters);  
 return responses;  
 }  
  
 [HttpGet("{id:int}")]  
 public async Task<CalculationResponse> GetByIdAsync(int id)  
 {  
 CalculationResponse response = await \_vehicleService.GetByIdAsync(id);  
 return response;  
 }

…  
}

*Фигура 5. Откъс от CalculationsController.cs*

Както се вижда по-горе, контролерът получава своя сървис през конструктора чрез Dependency Injection, и то не на конкретна инстанция, а на интерфейса. Имплементацията на сървисът не е явна в конструктора. Това прави кодът устойчив на промени във времето.

По-долу е демонстрирана част от имплементацията за калкулацията на премията за Гражданска отговорност на автомобили.

public class CalculationService : ICalculationService  
{  
 private readonly IVehicleInfoRepository \_vehicleInfoRepository;  
 private readonly ITariffTypeRepository \_tariffTypeRepository;  
 private readonly IEngineVolumeRepository \_engineVolumeRepository;  
…  
 private readonly IPurposeRepository \_purposeRepository;  
 private readonly ILogger<CalculationService> \_logger;  
 private readonly IMapper \_mapper;  
  
 public CalculationService(  
 IVehicleInfoRepository vehicleInfoRepository,  
 ITariffTypeRepository tariffTypeRepository,  
 IEngineVolumeRepository engineVolumeRepository,  
…  
 IPurposeRepository purposeRepository,  
 ILogger<CalculationService> logger,  
 IMapper mapper)  
 {  
 \_vehicleInfoRepository = vehicleInfoRepository;  
 \_tariffTypeRepository = tariffTypeRepository;  
 \_engineVolumeRepository = engineVolumeRepository;  
  
…  
 \_logger = logger;  
 \_purposeRepository = purposeRepository;  
 \_mapper = mapper;  
 }  
  
 public async Task<CalculationResponse> CalculateAsync(CalculationRequest request)  
 {  
 decimal pendingAmount = await \_vehicleInfoRepository.GetTotalPendingAmountAsync();  
 decimal paidAmount = await \_vehicleInfoRepository.GetTotalPaidAmountAsync();  
 int totalCount = await \_vehicleInfoRepository.GetPolicyCountAsync();  
  
 decimal basePremium = ((pendingAmount + paidAmount) / totalCount) \* 1.3M;  
  
 decimal engineVolumeMultiplier = await GetEngineVolumeMultiplierAsync(request.VehicleTariffTypeId);  
  
 decimal ageMultiplier = await GetAgeMultiplierAsync(request.VehiclePurposeId, request.VehicleTypeId, request.OwnerAgeId);  
  
 decimal municipalityMultiplier = await GetMunicipalityMultiplierAsync(request.MunicipalityId);  
  
 decimal premium = basePremium \* engineVolumeMultiplier \* ageMultiplier \* municipalityMultiplier;  
  
 ...  
  
 Calculation calculation = new Calculation  
 {  
 VehicleTariffTypeId = request.VehicleTariffTypeId,  
 EngineVolumeMultiplier = engineVolumeMultiplier,  
 VehiclePurposeId = request.VehiclePurposeId,  
 VehicleTypeId = request.VehicleTypeId,  
 OwnerAgeId = request.OwnerAgeId,  
 AgeMultiplier = ageMultiplier,  
 MunicipalityId = request.MunicipalityId,  
 MunicipalityMultiplier = municipalityMultiplier,  
 BasePremium = basePremium,  
 FinalPremium = premium,  
 };  
  
 await \_calculationRepository.AddCalculationAsync(calculation);  
 await \_calculationRepository.SaveChangesAsync();  
  
 CalculationResponse calculationResponse = \_mapper.Map<CalculationResponse>(calculation);  
 return calculationResponse;  
 }  
…  
}

*Фигура 6. Откъс от CalculationsController.cs демонстрираш алгоритъма за калкулация*

По-горе ясно може да се види силата на IoC контейнерите, които ни предоставят всички необходими сървиси на конструктора.

public async Task<PageContainer<CalculationResponse>> GetAllAsync(QueryParameters parameters)  
 {  
 IList<Calculation> calculations = await \_calculationRepository.GetAllAsync(parameters);  
 int itemsCount = await \_calculationRepository.GetCountAsync(parameters);  
  
 IList<CalculationResponse> response = \_mapper.Map<IList<CalculationResponse>>(calculations);  
  
 PageContainer<CalculationResponse> pageContainer =  
 new PageContainer<CalculationResponse>(  
 response,  
 itemsCount,  
 parameters.PageSize!.Value,  
 parameters.PageNumber!.Value);  
  
 return pageContainer;  
 }

*Фигура 7. Откъс от CalculationsController.cs демонстрираш алгоритъма за визуализиране на калкулации*

На същия принцип са изградени и хранилищата(repositories). При тях интересното е, че приемат наследник на DbContext, част от EntityFramework Core библиотеката, чрез който комуникираме с базата. Този DbContext се регистрира през Program.cs класът и може да се види на определената фигура към коя база данни е свързан.

public class CalculationRepository : ICalculationRepository  
{  
 private readonly VehicleContext \_vehicleContext;  
  
 public CalculationRepository(VehicleContext vehicleContext)  
 {  
 \_vehicleContext = vehicleContext;  
 }  
  
 public async Task AddCalculationAsync(Calculation calculation)  
 {  
 await \_vehicleContext.AddAsync(calculation);  
 }  
  
 public async Task<Calculation> GetByIdAsync(int id)  
 {  
 Calculation? calculation = await \_vehicleContext.Calculations  
 .FirstOrDefaultAsync(x => x.Id == id);  
 return calculation;  
 }  
  
 public async Task SaveChangesAsync()  
 {  
 await \_vehicleContext.SaveChangesAsync();  
 }  
  
 public async Task<IList<Calculation>> GetAllAsync(QueryParameters parameters)  
 {  
 IQueryable<Calculation> calculations = \_vehicleContext.Calculations;  
 calculations = *FilterCalculations*(parameters, calculations);  
  
 calculations = calculations  
 .Skip(parameters.SkipCount)  
 .Take(parameters.PageSize!.Value);  
  
 return await calculations.ToListAsync();  
 }

…  
}

*Фигура 8. Откъс от CalculationsRepository.cs демонстрираш извличането на информация от базата данни*

Структурата на базата данни се състои от няколко номенклатурни таблици, тоест такива, които съдържат параметри за нашата калкулация, таблица VehicleInfos, която държи информация за предходни полици и таблица Calculations, която съдържа нашите калкулации.

Основните таблици Calculations и VehicleInfos съдържат Foreign key-ове към номенклатурните таблици. Тези връзки за обозначени на долната схема.

Diagram

Description automatically generated

*Фигура 9. Схема на базата данни*

## 3.3. Тестове

Създадени са unit тестове за Сървисът за калкулации. Те целят да обхвана бизнес логиката на приложението, нейните основни и частни случаи. Направени са тестове за при невалидни входни данни и такива при валидни входни данни в различни бизнес сценарии. Използвани са библиотеките xUnit. Тестовете спазват конвенцията AAA – Arrange, Act, Assert. Благодарение на това, че е използван Dependency Injection през конструкторите на обектите и всички слоеве комуникират само с интерфейсите на другите проекти, то мокването, необходимо за тестване на отделните компоненти, е изключително лесно и приятно за разработка.

По следния начин се прави фалшив сървис, който да подадем на нашия тестван обект:

[Fact]  
public async Task CalculateAsync\_GivenInvalidAgeGroup\_ShouldThrowException()  
{  
 Mock<IVehicleInfoRepository> vehicleRepository = new();  
 vehicleRepository.Setup(x => x.GetTotalPaidAmountAsync())  
 .ReturnsAsync(200\_00M);  
 vehicleRepository.Setup(x => x.GetTotalPendingAmountAsync())  
 .ReturnsAsync(100\_000M);  
 vehicleRepository.Setup(x => x.GetPolicyCountAsync())  
 .ReturnsAsync(200);  
  
 Mock<ITariffTypeRepository> tariffTypeRepository = new();  
 tariffTypeRepository.Setup(x => x.GetByIdAsync(2))  
 .ReturnsAsync(new VehicleTariffType  
 {  
 Id = 2,  
 Description = "Mock description"  
 });  
 …  
  
 CalculationService calculationService = new CalculationService(  
 vehicleRepository.Object,  
 tariffTypeRepository.Object,  
 engineVolumeRepository.Object,  
 ageGroupRepository.Object,  
 municipalityRepository.Object,  
 calculationRepository.Object,  
 purposeRepository.Object,  
 logger.Object,  
 CreateMapper()  
 );  
  
 var calculationRequest = new CalculationRequest  
 {  
 VehicleTariffTypeId = 2,  
 MunicipalityId = 2,  
 VehiclePurposeId = 3,  
 VehicleTypeId = 2,  
 OwnerAgeId = 1,  
 };  
  
 EntityNotFoundException exception = await Assert.*ThrowsAsync*<EntityNotFoundException>(()  
 => calculationService.CalculateAsync(calculationRequest));  
  
 var expectedMessage = "Not results found";  
  
 Assert.*NotNull*(exception);  
 Assert.*Equal*(expectedMessage, exception.Message);  
}

*Фигура 10. Примерен тест*

## 3.4. Внедряване (deployment)

Нашето приложение ще работи на локалния хост. Нужни са ни няколко параметъра за това. Първо, базата данни работи със следния connection string: "Server=.\\SQLExpress;Database=Elto;Trusted\_Connection=True;".

След това трябва да знаем на кой порт работи нашето REST Api. В случая това е на 8080, а протокола е https.

От тук, при пускане на приложението през IDE-то спокойно можем да го консумираме през нашия фронтенд. Последният отваряме от index.html файлът. Благодарение на това, че CORS е разрешен за всички domain-и, нямаме проблем да достъпим API-то.

A picture containing text, indoor, screenshot

Description automatically generated

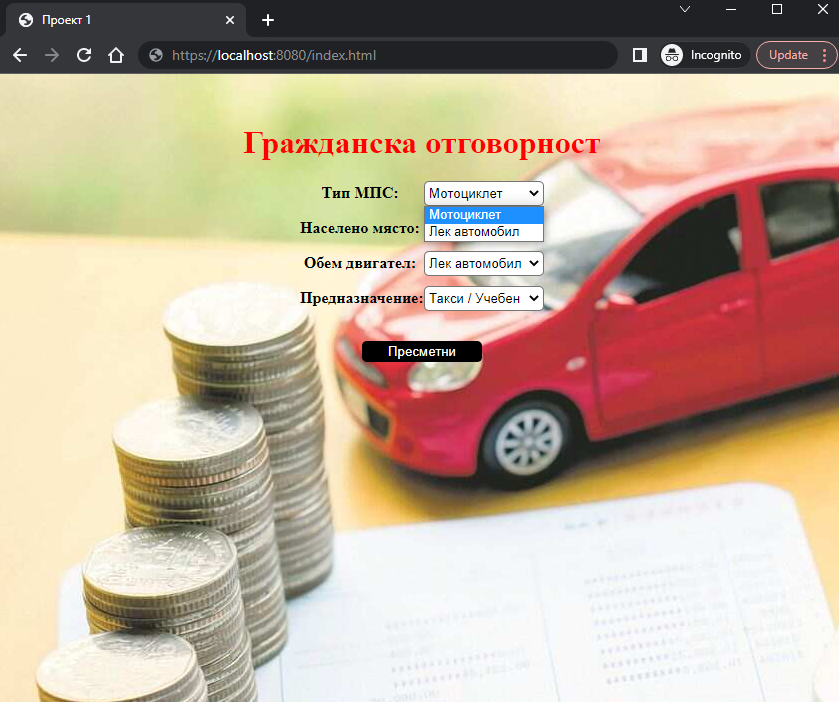
*Фигура 11. Пример на внедрено приложение*

# Глава 4. Ръководство за потребителя

Приложението „Калкулатор за изчисление на премията по застраховка Гражданска отговорност на автомобили“ може да бъде достъпено чрез index.html.

Отваря се следната страница с падащи менюта, в които се въвеждат данни за моторното превозно средство, което ще се застрахова и неговия собственик.

Проекта е достъпен през гитхъб: https://github.com/eltimirstoilkov

**

*Фигура 12. Пример за използване и селектиране на входните параметри за дадена калкулация*

За да се изчисли премията, по която може да се сключи застраховка Гражданска отговорност за определено превозно средство, потребителят трябва да въведе:

- населено място – от дефиниран с падащото меню списък;

- обем на двигателя - според предварително зададена скала;

- предназначение на автомобила – лични нужди; производствена или търговска дейност; такси или учебен автомобил;

- тип на автомобила – лек автомобил или мотоциклет и АТВ;

- възраст на собственика - при предназначение „лични нужди“ и тип на МПС „лек автомобил“ е необходимо да се въведат данни за възрастта на собственика, като се избере от дефиниран диапазон на възрастови групи.

След подаване на тези параметри, при калкулацията се селектира информация от базата данни за полиците, отговарящи на съответните критерии, като въз основа на историческа информация за щетите по съответните полици, се изчислява рисковата / базовата премия. След добавяне на заложените комисионни и административни разходи на застрахователя към базовата премия, се получава крайната премия, която се връща като отговор на клиента.

Всички параметри, които са използвани при калкулацията и нейния резултат се записват в базата, с цел да може да бъде проверена коректността на изчислението.

Graphical user interface, website

Description automatically generated

*Фигура 13. Пример за успешно калкулирана премия за застраховка Гражданска отговорност на автомобили*

# Заключение

Използвайки технологичния стек на Micrisoft ASP NET Core, Entity Framework Core и SQL Server за бекенд и JavaScript с jQuery за фронтенд е създадено приложение, с което да се изчислява премията за застраховка Гражданска отговорност на автомобили.

Калкулаторът може да се използва както от застрахователни компании и посредници, предлагащи застраховка Гражданска отговорност на автомобили, така и от всеки собственик, желаещ да провери на каква цена може да сключи полицата.

При калкулиране на премията се отчитат рискови характеристики като: вид и предназначение на превозното следство; възраст на собственика; населено място, където преимуществено се предполага, че ще се използва автомобила. Тъй като приложението е уеб базирано, то може да достигне широка аудитория. Благодарение на добрия подбор на технологии, при бъдеща нужда ще може да се разшири функционалността и да се включат допълнителни параметри за определяне на премията.

# Информационни източници

*1. За основната част от работата бяха използвани добре разработената документация на Microsoft https://learn.microsoft.com/*

*2. Презентациите предоставени по време на учебния процес*

*3. Кодекс за застраховане*

*4. Интернет сайтове на застрахователи и застрахователни брокери*

*5 „Актюерство“, доц. д-р Йото Йотов, Академично издателство „Ценов“ Свищов, 2000 г.*

|  |  |  |  |
| --- | --- | --- | --- |
| **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 |  | | |

# Рецензия на дипломен проект

**ЗАКЛЮЧЕНИЕ:**

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

.05.2023г. Рецензент:................................

град София