МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
«КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАДИМА ГЕТЬМАНА»
навчально-науковий інститут
«Інститут інформаційних технологій в економіці»

Кафедра інформаційних систем в економіці

галузь знань 12 Інформаційні технології
спеціальність 122 Комп'ютерні науки
спеціалізація Інформаційні управляючі системи та технології

Освітньо-професійна програма
«Інформаційні управляючі системи та технології»

Форма навчання: денна

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

РОЗРОБЛЕННЯ ІНФОРМАЦІЙНОЇ УПРАВЛЯЮЧОЇ СИСТЕМИ ДЛЯ СТРАХОВОЇ КОМПАНІЇ

студентки Рибнікової Анни Іллівни _____

Науковий керівник: д.е.н., проф.
_____ Устенко С.В.

Кваліфікаційна магістерська робота допущена до захисту в Екзаменаційній комісії з атестації здобувачів вищої освіти
Завідувач кафедри: к.е.н., доц.
_____ Тішков Б.О.

Київ 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
«КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВАДИМА ГЕТЬМАНА»
навчально-науковий інститут
«Інститут інформаційних технологій в економіці»

Кафедра інформаційних систем в економіці

галузь знань 12 «Інформаційні технології»
спеціальність 122 «Комп'ютерні науки»
спеціалізація «Інформаційні управляючі системи та технології»

Освітньо-професійна програма
«Інформаційні управляючі системи та технології»

Затверджую:
В.о. завідувача кафедри
_____ Тішков Б.О.
"____" _____2022 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

студентці Рибніковій Анні Іллівні

денної форми навчання

на підготовку кваліфікаційної магістерської роботи
на тему: «Розроблення інформаційної управляючої системи для страхової компанії»
Тему затверджено наказом ректора Університету від «___» _____ 2022 р. №_____.

Кваліфікаційна магістерська робота виконується на матеріалах
Навчально-наукова лабораторія «Інформаційні управляючі системи і технології» ДВНЗ «КНЕУ імені Вадима Гетьмана»

План кваліфікаційної магістерської роботи

Розділ І	*Теоретичний розділ. Дослідження та аналіз підходів до створення інформаційних управляючих систем страхової компанії*
(назва розділу)	
Розділ ІІ	*Аналітичний розділ. Характеристика інформаційних управляючих систем та методи і моделі*
(назва розділу)	
Розділ ІІІ	*Конструктивний розділ. Розробка проектних рішень*
(назва розділу)	

Об'єкт дослідження:	*Діяльність страхової компанії*
Предмет дослідження:	*Автоматизація надання послуг зі страхування страховими компаніями*
Мета кваліфікаційної магістерської роботи:	*Розробка та впровадження інформаційної управляючої системи для страхової компанії*

Конкретні завдання, які студент повинен виконати для досягнення поставленої мети:

У розділі 1	*Проаналізувати існуючі інформаційні управляючі системи страхових компаній, дослідити предметну область та ринок ІУС страхових компаній; обґрунтувати вибір підходів і технологій для створення інформаційних управляючих систем.*

У розділі 2	*Проаналізувати доступні структури побудови інформаційних систем та обґрунтувати вибір конкретної для розробки власної системи, надати її характеристику; окреслити методи та моделі в інформаційних управляючих системах, доцільність їх використання для розробки ІУС обраної предметної області*

У розділі 3	*Надати схематичне зображення побудови бази даних та знань, охарактеризувати їх; надати чіткі вимоги до розроблюваного програмного забезпечення, інструментальних засобів розробки, розробити архітектуру системи та навести чітку інструкцію для користувача системи; охарактеризувати технічне забезпечення та провести аналіз результатів реалізації інформаціної системи.*

Завдання підготував науковий керівник _____ **Устенко Станіслав Веніамінович**
(підпис) *(ініціали, прізвище)*

«_____» _____ 2022 р.

Завдання одержав здобувач _____ **Рибнікова Анна Іллівна**
(підпис) *(ініціали, прізвище)*

«_____» _____ 2022 р.

АНОТАЦІЯ

Кваліфікаційної магістерської роботи студентки 6 курсу ННІ «Інститут інформаційних технологій в економіці»

Рибнікової Анни Іллівни

виконану на тему:

«Розроблення інформаційної управляючої системи для страхової компанії»

Київ: кафедра інформаційних систем в економіці, 2022 р.

Магістерська кваліфікаційна робота присвячена актуальній проблемі автоматизації діяльності страхових компаній, яку пропонується розв'язувати з використанням сучасних інформаційних технологій.

Дана робота складається з трьох розділів. Перший розділ являється теоретичним. В ньому наведено дослідження предметної області, наводиться перелік існуючих програмних засобів автоматизації бізнес-процесів у даній сфері, а також обґрунтовується вибір підходів і технологій для створення ІУС.

Другий розділ є аналітичним і присвячений структурі та характеристиці системи, а також обґрунтуванню й розробці відповідної моделі інформаційної управляючої системи.

Третій розділ – конструктивний. Наведено програмне та технічне забезпечення створення програмного забезпечення, деталізовано вимоги та інструментальні засоби до системи, її архітектура та чітку інструкцію користування. Висвітлені питання побудови бази даних та знань: розроблена структура, описані та наведені форми вхідних та вихідних даних та взаємозв'язки. Обґрунтований комплекс технічних засобів, а також програмне забезпечення, що використовується для створення системи інформаційної управляючої системи страхової компанії.

Висновки містять рекомендації щодо доцільності розробки та впровадження інформаційної управляючої системи страхової компанії.

РЕФЕРАТ

Пояснювальна записка: 83 с., 27 рис., 6 табл., 2 ф-ли, 30 джерел.

Об’єкт дослідження: діяльність страхової компанії.

Предмет дослідження: автоматизація надання послуг зі страхування страховими компаніями.

В роботі проаналізовано існуючі інформаційні управляючі системи страхових компаній, виявлено їх недоліки та запропоновано комплексну систему, що пришвидшить роботу та перекриє недоліки можливих конкурентів .

Практичне значення роботи полягає у використані розробленої інформаційної управляючої системи у реальних компаніях для пришвидшення здійснення процесів для подальшої максимізації прибутку.

Результати здійснених у дипломному проекті досліджень можуть бути використані компаніями для впровадження у існуючу систему як надбудови для покриття недоліків існуючої системи.

Ключові слова: інформаційна управляюча система, система прийняття рішень, штучний інтелект, автоматизація бізнес-процесів, .NET, програмування, розробка програмного забезпечення.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ ВЕЛИЧИН І ТЕРМІНІВ

АСУ – автоматизована система управління

ІС – інформаційна система

ТП – технологія проектування

ІУС – інтелектуальна управляюча система

БД – база даних

ПЗ – програмне забезпечення

СКБД – система керування базою даних

СКБМ – система керування базою моделей

ОПР – особи, що приймають рішення

СКП – система керування повідомленнями

ШНМ – штучна нейронна мережа

МН – машинне навчання

ВСТУП

На сьогодні, ефективне керування підприємством неможливе без використання комп’ютерних технологій. Тож, із стрімким розвитком страхових компаній зростає чисельність їх клієнтури, загострюється конкуренція, з'являється необхідність введення нових послуг – все це вимагає вдосконалення управління і забезпечення прозорості бізнесу, що в сучасних умовах недосяжно без опори на інформаційні технології.

Зауважимо, що страхування є досить трудомістким та складним процесом, у якому вкрай необхідно враховувати безліч чинників. Саме тому, усім страховим компаніям необхідно автоматизувати збір та обробку інформації, тобто впровадити і розвивати інформаційну систему.

Актуальність роботи полягає у тому, що на ринку недостатня кількість програмного забезпечення, яке б відповідало усім необхідним потребам бізнесу. Розробка комплексної системи допоможе перекрити недоліки існуючих та зберегти фінанси компаній й забезпечити зручність використання.

Мета роботи: розробка та впровадження інформаційної управляючої системи для страхової компанії.

Для досягнення поставленої мети були поставлені наступні **завдання:**

- Проаналізувати існуючі ІУС страхових компаній, дослідивши предметну область та ринок ІУС;
- Обґрунтувати вибір підходів і технологій для створення ІУС;
- Навести структуру системи та охарактеризувати її;
- Дослідити методи і моделі в ІУСіТ;
- Спроектувати та охарактеризувати базу даних для ІУС;
- Провести інтелектуальний аналіз даних, спроектувати та реалізувати базу знань;
- Описати вимоги до створення ІУС;
- Навести та описати засоби розробки для створення ПЗ ІУС;
- Побудувати архітектуру програмного забезпечення;

– Написати керівництво (інструкцію) користувача;
– Описати технічне забезпечення;
– Навести та проаналізувати результати реалізації інформаційної системи.

Об’єктом дослідження є діяльність страхової компанії.

Предметом дослідження є автоматизація надання послуг зі страхування страховими компаніями.

Методи та інструментарій дослідження: методи наукових узагальнень, порівняльного і системного аналізу наукової літератури, статистичні та емпіричні методи дослідження; для обробки, аналізу та подання інформації будуть застосовані такі програмні засоби, як ERWin Proccess Modeler, Lucid Chart, Microsoft Excel.

Проект складається з трьох розділів:

1. Перший розділ є теоретичним, у якому проводиться дослідження предметної області, вже існуючих програм та обґрунтування вибору підходів для створення ІУС.
2. Другий розділ спрямований на аналітику. Тут проводимо характеристику інформаційних управляючих систем, її структури та описуємо методи та моделі і ІУС.
3. Третій розділ – конструктивний. У цьому розділі надаємо вичерпну інформацію про базу даних та знань, а також програмне та технічне забезпечення ІУС, з описом вимог розробки, інструментальних засобів реалізації, архітектуру та інструкцію користувача.

І. ТЕОРЕТИЧНИЙ РОЗДІЛ. ДОСЛІДЖЕННЯ АНАЛІЗІВ ТА ПІДХОДІВ ДО СТВОРЕННЯ ІНФОРМАЦІЙНИХ УПРАВЛЯЮЧИХ СИСТЕМ СТРАХОВОЇ КОМПАНІЇ

1.1. Аналіз існуючих інформаційних управляючих систем страхових компаній

1.1.1 Дослідження предметної області

Страхування – це система відносин щодо захисту майнових інтересів фізичних і юридичних осіб при настанні страхових випадків за рахунок грошових фондів, що формуються шляхом сплати ними страхових премій [1].

Страховий ринок – частина фінансово-кредитної сфери, яка регулюється державою. Основною ланкою страхового ринку є страхова компанія. Страхова компанія – певна форма функціонування страхового фонду [2].

Страхова компанія – це організація, яка надає послуги із здійснення різноманітних страхових операцій. Зазвичай це юридична особа, створена для реалізації страхової діяльності, яка законно отримала ліцензію на цю діяльність [3].

Мета страхування є комплексною:

1. Першою метою страхування є захист. Основну мету страхової діяльності можна визначити як задоволення суспільної потреби в надійному страховому захисті від випадкових небезпек, що відповідає загально прийнятим вимогам по фінансовій надійності.

2. Другою, і не менш важливою метою страхування, з макроекономічної позиції, є акумуляція грошових коштів, сплачених безліччю страхувальників, і інвестування їх в економіку. Інвестиційна ефективність страхової діяльності вище банківської, оскільки страхування забезпечує довготривалі інвестиції.

Ступінь досягнення основної мети і буде визначати ефективність страхової діяльності. Повна технологія страхування передбачає обробку великих і взаємопов'язаних масивів даних [4]:

– договорів страхування і перестрахування;
– страхових полісів;
– брокерських договорів;
– документів із зарплати страхових представників;
– платіжних доручень;
– касових ордерів і бухгалтерських проводок;
– заяв на виплату страхового відшкодування;
– актів про страхові випадки;
– тощо.

Застосування інформаційних систем значно спрощує процес роботи страхової компанії. Тобто найбільшого успіху зможуть досягти ті страхові компанії, діяльність яких чітко організована й автоматизована.

У страховому бізнесі основою всього є страховий продукт – послуга з різних видів страхування. Страхові продукти по одному і тому ж виду страхування можуть бути унікальними для різних страхових компаній, які формують і модифікують свій набір умов, що пропонуються клієнтам. Для того, щоб підтримувати відмінності і можливість постійних змін страхових продуктів, інформаційна система повинна мати гнучкі засоби налаштування параметрів договорів страхування.

Інформаційно-управляюча система (ІУС) – це сукупність засобів, методів, виконавців, що забезпечують необхідною і достатньою інформацією реалізацію всіх заходів процесу управління. ІУС є інтегрованою звітною системою, спеціально призначеною для допомоги керівникам у плануванні, здійсненні та контролі діяльності своєї установи.

Основними складовими ІУС є [5]:

– персонал – члени колективу, які беруть участь у функціонуванні ІУС;
– інформаційні ресурси – конкретний зміст інформації, яка використовується в управлінській діяльності;
– матеріальні ресурси – носії інформації, технічні засоби збору, обробки, зберігання, передачі інформації;
– канали циркуляції інформації – конкретні рівні комунікації, призначені для постійного поповнення і отримання інформації.

Manager Information System (MIS) – системи, які в потрібний момент часу в найбільш зручній формі представляють керівнику необхідну інформацію про минуле, сьогодення і майбутнє керованої системи.

Сучасна інформаційна управляюча система для страхової компанії повинна забезпечувати вирішення таких завдань [6]:

Задачі стратегічного рівня управління СК:

1. Розробка тарифної політики;
2. Розробка політики управління страховим портфелем;
3. Розробка політики управління інвестиційним портфелем;
4. Розробка політики управління власним капіталом;
5. Розробка політики управління страховими резервами;
6. Розробка політики перестрахування;
7. Розробка політики управління витратами;
8. Розробка методології аналізу ефективності управління СК;
9. Розробка стратегічного плану діяльності СК.

Задачі тактичного рівня управління СК:

1. Оцінка, аналіз та коригування методології розрахунку страхових тарифів;
2. Оцінка, аналіз та коригування методології управління страховим портфелем;
3. Оцінка, аналіз та коригування методології управління інвестиційним портфелем;

4. Оцінка, аналіз та коригування методології управління власним капіталом;
5. Оцінка, аналіз та коригування методології управління страховими резервами;
6. Оцінка, аналіз та коригування методології управління операціями перестрахування;
7. Оцінка, аналіз та коригування методології управління витратами;
8. Аналіз відповідності фактичних результатів плановим;
9. Прогнозування фінансового стану СК;
10. Розробка плану діяльності СК на недалеке майбутнє.

Задачі оперативного рівня:

1. Аналіз потенційного страхувальника: виявлення шахрайств на повторне страхування, аналіз страхової та кредитної історії;
2. Андеррайтинг потенційного об’єкта страхування: аналіз об’єкта страхування (страхова сума, страхові ризики, умови страхування), можливість та доцільність його страхування; аналіз можливості перестрахування ризику;
3. Розрахунок страхового тарифу, що враховує всі умови договору страхування і втілює принцип еквівалентності зобов’язань страхувальника і страховика;
4. Виявлення шахрайств при розгляді заяви на виплату страхового відшкодування;
5. Вибір контрагентів, що дозволяє ефективно управляти витратами на ведення справи.

1.1.2. Дослідження ринку інформаційних управляючих систем

На поточний момент існує велика кількість страхових інформаційних продуктів, однак я б хотіла охарактеризувати найбільш популярні серед провідних страхових компаній України:

1. BlackWater (від української компанії InCore) [7]

Використовується страховими компаніями: Оранта, ІНГО, НАСТА.

На ринку: 11 років.

Цінова категорія: висока.

BlackWater – це комплексна система для автоматизації діяльності страхової компанії. Впроваджуючи систему BlackWater страхова компанія вирішує завдання:

- роботи фронт-офісу та віддаленого вводу договорів;
- страхового обліку та бек-офісу;
- автоматизації процесу врегулювання збитків.

Систему BlackWater умовно можна розділити на два шари «інфраструктурний» і «функціональний»:

- Інфраструктурний шар – складається з базових модулів і сервісів необхідних для роботи системи. Наприклад, модуль авторизації, модуль управління структурою організації тощо;
- Функціональний шар – складається з модулів, обраних замовником, які відповідають за автоматизацію бізнес-завдань. Наприклад, фінансовий модуль, модуль продажів, модуль регулювання збитків тощо.

Система BlackWater розроблена таким чином, що замовник може вибирати і купувати тільки ті модулі функціонального шару, які йому необхідні.

Вартість і терміни впровадження системи залежать від конфігурації системи та узгоджуються індивідуально.

2. InsCom (від української компанії UIIS) [8]

Використовується страховими компаніями: Allianz, ПРОСТО-страхування.

На ринку: 21 рік.

Цінова категорія: середня.

InsCom – інформаційна система комплексної автоматизації страхових компаній. Її функції полягають у:

– повному обліку всієї страхової інформації,
– зв'язку з бухгалтерією,
– контролем платежів,
– контролі CRM-системи,
– менеджменті клієнтів,
– колективній роботі над управлінням завданнями.

Технології системи:

– Гнучкі налаштування на будь-які страхові продукти;
– Працює в мережі філій з установкою повноцінних робочих місць і обміном даними в режимах on-line та off-line;
– Практичні обмеження за рівнем деталізації інформації, обсягом даних та кількості користувачів відсутні.

Варіанти пропонованих рішень системи:

1. Власний сервер з відкритим кодом по виписці електронних полісів. Сервер може бути розташований на ресурсі страхової компанії фізично, або перенесений у дата-центр, причому всі доробки системи можна робити самостійно за рахунок відкритого коду.
2. Власний сайт-агрегатор продажів полісів.

   Агрегатор може продавати поліси багатьох компаній, а може бути використаний як продажний сайт для страхової компанії з використанням фірмового стилю.
3. Онлайн система для агентів з відкритим кодом, що може налаштовуватися фахівцями страхової компанії для крос-продажів, розширюючи продаж інших продуктів за рахунок клієнтів ОСАГО. Її використання не вимагає ніяких платежів за ліцензії та підтримку.

Тож, дані пропозиції дозволять використовувати конкурентну перевагу онлайн-продажів, клієнтських сервісів, крос-продажів з повністю

контрольованими витратами і без ризику "підсаджування на голку" розробника за рахунок відкритого коду.

Також в ході роботи із системою фахівці компанії вивчають найпередовіші інтернет-технології, системи розробки, ведення репозиторіїв коду, управління проектами, контролю завдань і складання документації.

3. INSIS (від болгарської компанії Fadata) [9]

Використовується страховими компаніями: Оранта, Провідна, Аска.

На ринку: 26 років.

Цінова категорія: висока.

Клієнти компанії Fadata, що займаються страхуванням життя, пенсійного страхування, медичного страхування, використовують платформу страхових процесів INSIS та пов’язані рішення для оптимізації основних процесів та процесів, спрямованих на клієнтів, що у комплексі сприяють успішному залученню клієнтів.

Починаючи від фронт-офісних продажів і закінчуючи управлінням життєвим циклом продуктів, INSIS дозволяє налаштовувати процеси страхової компанії на базі потреб сучасного цифрового споживача, що швидко змінюються.

4. Fort: Управління страховим бізнесом (від української групи компаній «Форт») [10]

Використовується страховими компаніями: Наста, Кардиф, Арма.

На ринку: 14 років.

Цінова категорія: середня

Система дозволяє вести облік об'єктів страхування різної природи, контролювати всі події, що відбуваються з окремим об'єктом страхування або класом об'єктів за встановленими критеріями відбору. Налаштування включає цілісний комплекс статистичної, управлінської та фінансової звітності для всебічного аналізу діяльності компанії та підтримки прийняття рішень.

При використанні відповідної комплектації система може використовуватися як в центральному офісі страхової компанії, так і у всіх

віддалених підрозділах, з можливістю автоматичної консолідації даних і поділу прав доступу користувачів до різних розділів обліку.

Так, охарактеризуємо основні функціональності надбудови, що пропонує у своєму рішенню компанія Fort на систему 1С:Підприємство:

– Ведення класифікатора правил страхування та історії його зміни;
– Ведення класифікатора страхових ризиків для кожного з правил страхування;
– Конструктор об'єктів страхування:
  - Визначення довільної кількості типів об'єктів страхування,
  - Визначення довільного набору реквізитів для кожного типу об'єктів страхування;
– Конструктор страхових продуктів:
  - Визначення довільного набору страхових ризиків для кожного страхового продукту,
  - Угруповання ризиків в програми страхування в рамках страхового продукту,
  - Можливість визначення коефіцієнтів пайової участі кожного ризику в страховому тарифі,
  - Визначення довільної кількості додаткових умов страхування для кожного страхового продукту,
  - Можливість визначення набору типових умов страхування для страхового продукту: типові франшизи тощо,
  - Визначення показників та формул для розрахунку сум страхового відшкодування по страховому продукту,
  - Налаштування шаблонів друкарських форм документів і правил їх заповнення,
  - Можливість призначення довільного набору додаткових, специфічних для страхового продукту, реквізитів договору страхування;

– Модуль андеррайтингу:
  - Можливість настройки механізмів автоматизованого розрахунку тарифів за обраним продукту,
  - Визначення тарифоутворюючих коефіцієнтів і формул розрахунку,
  - Можливість застосування різних тарифних сіток для різних підрозділів, агентів, категорій клієнта тощо,
  - Використання процедури електронного підтвердження умов страхування, можливість спецакценту на нестандартних умов договору.

1.2. Обґрунтування вибору підходів і технологій для створення інформаційних управляючих систем

У теорії та практиці створення інформаційних систем виділяють три підходи [11]:

1. Локальний.

Сутність локального підходу полягає в тому, що інформаційна система створюється послідовним нарощуванням задач, що призводить до збільшення функціональних можливостей. Він передбачає необмежений розвиток інформаційних систем, а тому кожну із них неможливо пізнати в цілому.

Недоліки: неможливість забезпечення раціональної організації комплексів задач, дублювання, постійна перебудова програм та організації задач, що призводить до дискредитації самої ідеї створення інформаційної системи.

2. Глобальний.

При глобальному підході створюється проект системи у повному обсязі, а потім здійснюється її розробка та впровадження. Як правило, цей підхід призводить до морального старіння проекту ще до його впровадження, оскільки час його розробки може перевищувати період оновлення технічних, програмних та інших засобів, використаних у ньому.

3. Системний.

Системний підхід до створення інформаційної системи – це комплексне вивчення об'єкта автоматизації як одного цілого з представленням частин його як цілеспрямованих систем і вивчення цих систем та взаємозв'язків між ними. При системному підході об'єкт розглядається як сукупність взаємопов'язаних елементів однієї складної динамічної системи, яка перебуває в стані постійних змін під впливом багатьох внутрішніх і зовнішніх факторів, пов'язаних процесами перетворення вхідної інформації у вихідну внаслідок вирішення задач автоматизації.

Системний підхід заснований на принципах:

- абсолютного пріоритету кінцевої мети або цільового спрямування;
- розгляду системи як цілого, так і сукупності елементів;
- розгляду будь-якої частини разом з її зв'язками з оточенням;
- модульності (при побудові виділяються модулі, що є складовими елементами системи, а система розглядається як сукупність модулів);
- введення ієрархії складових частин або елементів і їх ранжування;
- функціональності (сукупний розгляд структури і функцій з пріоритетом функцій над структурою);
- врахуванням змін системи, її здатності до розвитку, розширення, заміни частин, нагромадження інформації;
- поєднання рішень, які приймаються, та керування централізацією і децентралізацією;
- врахуванні невизначеностей та випадковостей у системі.

Задачею системного підходу до створення інформаційної системи є розробка всієї сукупності методологічних і соціально – наукових засобів обстеження (опис, аналіз, синтез, реалізація) систем різного типу.

Тож, саме останній підхід буде застосовано при розробці ІУС у даному проекті.

Технологією проектування (ТП) ІС сукупність методології та засобів проектування ІС, а також методів і засобів організації проектування [12].

Технологія проектування задається регламентованою послідовністю технологічних операцій, що виконуються в процесі створення проекту на основі конкретного методу, в результаті чого стало б ясно, не тільки ЩО повинно бути зроблено для створення проекту, але і ЯК, КОМУ і в ЯКІЙ ПОСЛІДОВНОСТІ це повинно бути зроблено.

Основу технології проектування ІС становить методологія, яка визначає сутність, основні відмінні технологічні особливості. В процесі створення проекту ІС організація проектування передбачає визначення методів взаємодії проектувальників між собою та із замовником.

Методи проектування ІС можна класифікувати за ступенем: використання засобів автоматизації, використання типових проектних рішень, адаптивності до передбачуваних змін [12].

1. За ступенем автоматизації: мануальне та комп’ютерне проектування,
2. За ступенем використання типових проектних рішень: індивідуальне та типове проектування,
3. За ступенем адаптивності проектних рішень: методи реконструкції, параметризації та реструктуризації моделі.

Поєднання різних ознак класифікації методів проектування обумовлює характер використовуваної ТП ІС, серед яких виділяються два основні класи: канонічна та індустріальна технології.

Для проекту розробки інформаційної управляючої системи у страховій компанії були обрано індустріальне автоматизоване проектування, що передбачає:

– Комп’ютерне проектування. Воно передбачає генерацію та налаштування проектних рішень на основі використання спеціальних інструментальних програмних засобів;
– Оригінальне проектування, тобто проектне рішення буде розроблене «з нуля»;

– Реструктуризація моделі (генерація ІС). Буде змінено модель проблемної області на основі якої автоматично генеруються проектні рішення.

ІІ. АНАЛІТИЧНИЙ РОЗДІЛ. ХАРАКТЕРИСТИКА ІНФОРМАЦІЙНИХ УПРАВЛЯЮЧТХ СИСТЕМ ТА МОДЕЛІ І МЕТОДИ

2.1. Структура і характеристика системи

Система ІУС складається із забезпечувальної та функціональної частини.

Призначення системи: пришвидшення процесу розгляду заявок на страхування.

Загально, сучасну структуру інформаційної управляючої системи можна побудувати таким чином. (рис. 2.1.)

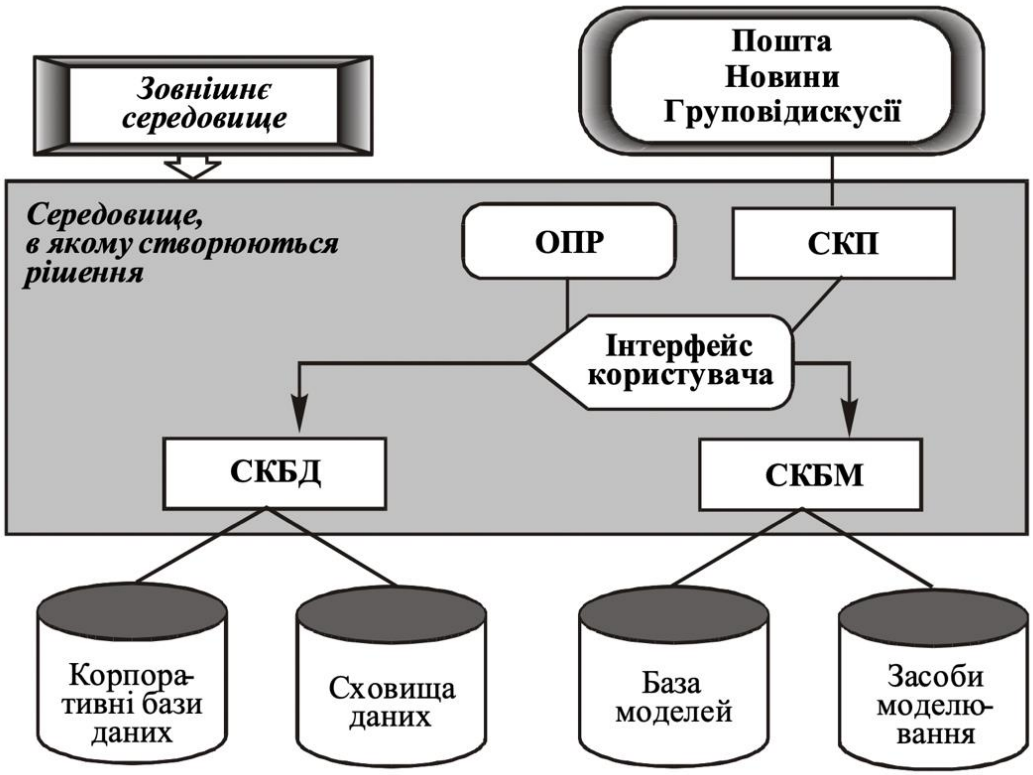


Рисунок 2.1 – Сучасна структура ІУС [13, стор. 348]

Тож, саме таким чином будуватимемо і нашу ІУС щодо надання страхування.

Загальна архітектура ІУС поєднує інтерфейс користувача, базу даних, моделі й аналітичні інструментальні засоби та мережеву структуру. (Рис. 2.2.)

Рисунок 2.2 – Загальна архітектура ІУС [13, стор. 397]

Інтерфейс користувача (засіб зручної взаємодії користувача з інформаційною системою) є головним компонентом.

БД ІУС є сукупністю даних, які організовуються для легкого доступу до них і аналізу. Архітектура для БД ІУС включає декілька серверів.

Конкретним кейсом для наглядності роботи системи було обрано автострахування.

Страхування транспортних засобів (також автострахування) — вид страхування для легкових автомобілів, вантажівок та інших транспортних засобів, що повинен захистити майнові інтереси автовласника, пов'язані з витратами на відновлення транспортного засобу після аварії, поломки чи купівлю нового авто після угону [14].

На оформлення договору страхування та виплату страхової суми впливають [15]:

1. Історія водіння

Якщо у особи є послужний список безпечної їзди, то є велика ймовірність того, що вона становитиме менше ризику нещасного випадку, ніж той, хто має низку попередніх претензій або судимостей.

Саме тому страховики враховують історію людини за кермом при виданні страховки. Вони враховують історію заявок, оскільки водії, які подали заявки в минулому, частіше мають шанс зробити інші заявки у майбутньому. В рамках

цього страховики нададуть перевагу особам без претензій, яку ви нарахували за роки безпригодних автомобільних перевезень.

2. Сімейне положення

Люди з сім'єю та дітьми схильні до спокійного водіння транспортним засобом, що є менш ризикованим для

3. Кредитна історія

Людині, що не має кредитної історії, або ж має гарну – можна більше довіряти, тож надання страховки не буде проблемою.

Суттєвими умовами договору страхування є страхова сума і страхова виплата. Страхова сума — грошова сума, в межах якої страховик відповідно до умов договору страхування зобов´язаний здійснити виплату в разі настання страхового випадку. Страхова виплата — грошова сума, яка виплачується страховиком відповідно до умов договору страхування при настанні страхового випадку.

Наразі зберігання інформації ведеться із застосуванням різних технічних засобів. Так, маємо контекстну діаграму (рис. 2.3) процесу страхування та її декомпозиція (рис. 2.4).

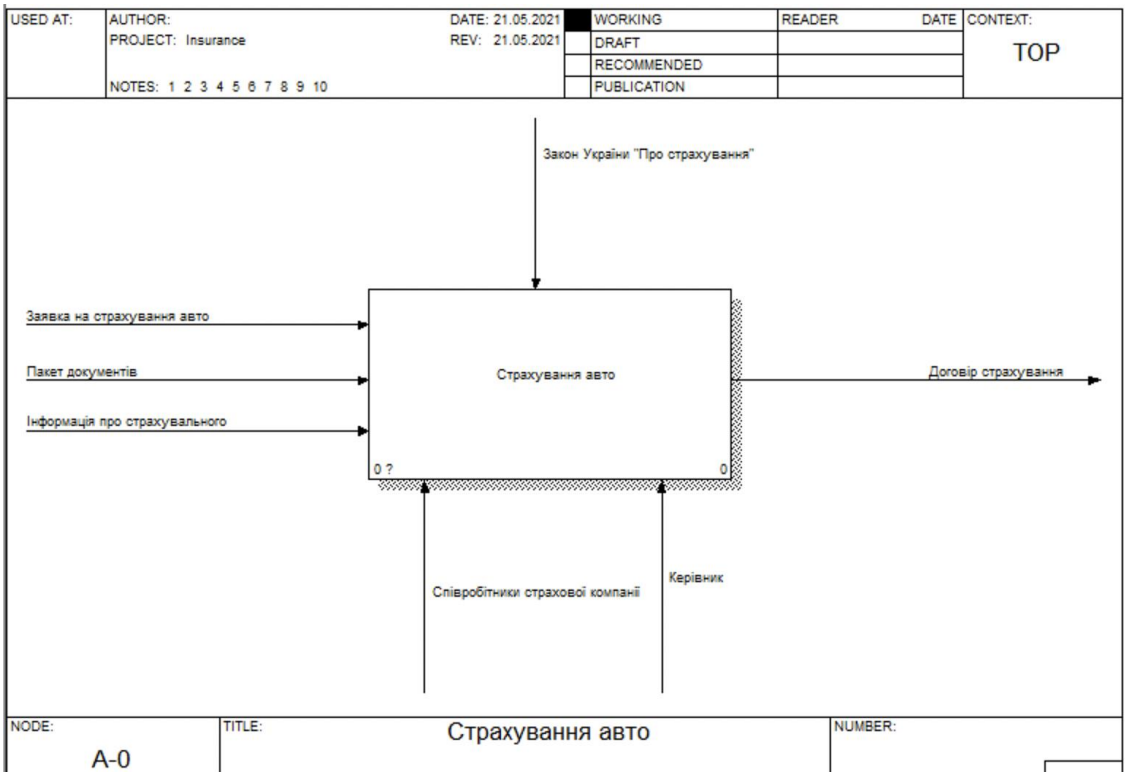


Рисунок 2.3 – Контекстна діаграма процесу страхування авто [Власна розробка]

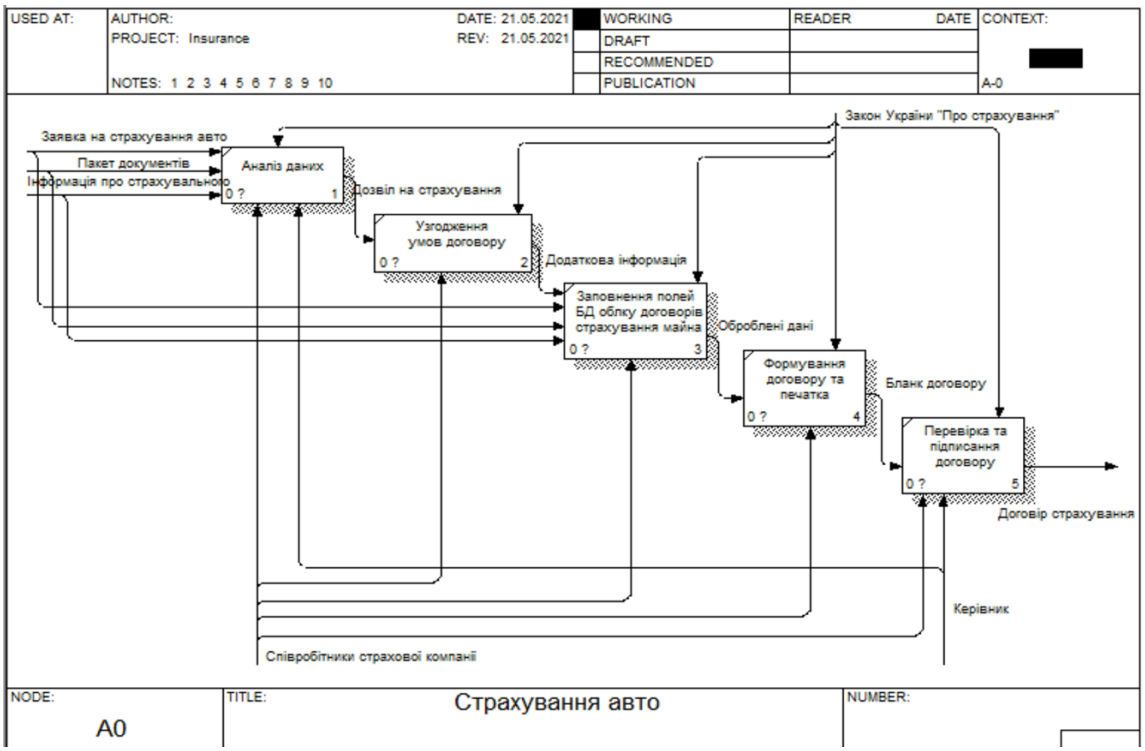


Рисунок 2.4 – IDEF0-декомпозиція процесу страхування автомобіля [Власна розробка]

Також, маємо BPMN-діаграму процесу прийняття страхового рішення (рис.2.5).

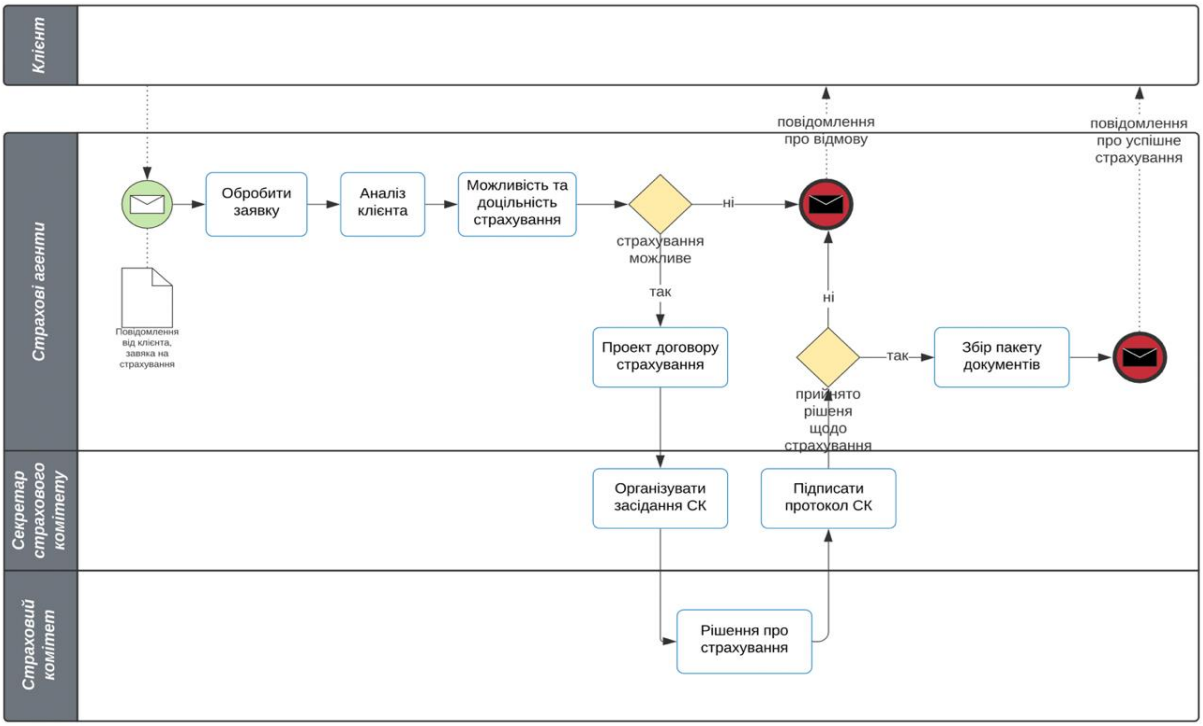


Рисунок 2.5 – BPMN-діаграма процесу прийняття рішення щодо надання страхового рішення. [Власна розробка]

2.2. Методи та моделі в інформаційних управляючих системах і технологіях

Для розробки інформаційної управляючої системи було обрано використання об’єктно-орієнтованого програмування.

Об'єктно-орієнтоване програмування — це метод програмування, заснований на поданні програми у вигляді сукупності взаємодіючих об'єктів, кожен з яких є екземпляром певного класу, а класи є членами певної ієрархії наслідування. Програмісти спочатку пишуть клас, а на його основі при виконанні програми створюються конкретні об'єкти (екземпляри класів). На основі класів можна створювати нові, які розширюють базовий клас і таким чином створюється ієрархія класів [16].

Об'єктно-орієнтований підхід полягає в наступному наборі основних принципів [16]:

- Все є об'єктами;
- Всі дії та розрахунки виконуються шляхом взаємодії (обміну даними) між об'єктами, при якій один об'єкт потребує, щоб інший об'єкт виконав деяку дію. Об'єкти взаємодіють, надсилаючи і отримуючи повідомлення. Повідомлення — це запит на виконання дії, доповнений набором аргументів, які можуть знадобитися при виконанні дії;
- Кожен об'єкт має незалежну пам'ять, яка складається з інших об'єктів.
- Кожен об'єкт є представником (екземпляром, примірником) класу, який виражає загальні властивості об'єктів;
- У класі задається поведінка (функціональність) об'єкта. Таким чином усі об'єкти, які є екземплярами одного класу, можуть виконувати одні й ті ж самі дії;
- Класи організовані у єдину деревоподібну структуру з загальним корінням, яка називається ієрархією успадкування. Пам'ять та поведінка, зв'язані з екземплярами деякого класу, автоматично

доступні будь-якому класу, розташованому нижче в ієрархічному дереві;

Процес розрахунку страхової суми було вирішено запровадити за допомогою систем штучного інтелекту. Так, як і у випадку з більшістю нейронних мереж, моя мета полягає в навчанні мережі таким чином, щоб досягти балансу між здатністю мережі давати вірний відгук на вхідні дані, що використовувалися в процесі навчання (запам'ятовування), і здатністю видавати правильні результати у відповідь на вхідні дані, схожі, але неідентичні тим, що були використані під час навчання (принцип узагальнення).

В загальному вигляді алгоритм зворотного поширення помилки, що вирішено застосувати, являє собою наступну послідовність кроків [17]:

– Крок 1: Ініціювати ваги малими випадковими величинами;
– Крок 2: Якщо умова зупинки не виконується, виконати кроки 3-10%;
– Крок 3: Для кожної навчальної пари виконати кроки 4-9;

Прямий прохід:

– Крок 4: Кожен вхідний нейрон $x_i = 1 \dots n$ приймає вхідний сигнал і поширює його до всіх нейронів прихованого шару;
– Крок 5: Кожен нейрон прихованого шару $v_j = 1 \dots q$ підсумовує свої зважені вхідні сигнали: $h_j = \sum_i^n w_{ij} x_i$, застосовує до одержаної суми функцію активації, формуючи вихідний сигнал: $v_j = f(h_j)$, котрий надсилається до всіх нейронів вихідного шару;
– Крок 6: Кожен вихідний нейрон $y_k = 1 \dots m$ підсумовує зважені сигнали: $h_k = \sum_j^q w_{jk} v_j$ формуючи після застосування функції активації вихідний сигнал мережі: $y_k = f'(h_k)$;

Зворотне поширення помилки:

– Крок 7: Кожен вихідний нейрон співставляє своє значення виходу з потрібною цільовою функцією і вираховує $\delta_k = (t_k - y_k) f'(h_k)$, після чого визначається корегуючий член ваг: $\Delta w_{jk} = \eta \delta_k v_j$, а параметри δ_k надсилаються в нейрони прихованого шару;

– Крок 8: Кожен нейрон прихованого шару v_j підсумовує свої δ-входи від нейронів вихідного шару: $h_k = \sum_k^m \delta_k w_{jk}$, результат помножують на похідну від функції активації для визначення δ_j: $\delta_j = f'(h_j) \sum_k^m \delta_k w_{jk}$, та вираховує коригуючий член: $\Delta w_{ij} = \eta \delta_k w_{jk}$.

За допомогою блок-схеми поданий алгоритм можна зобразити наступним чином (Рис. 2.6).

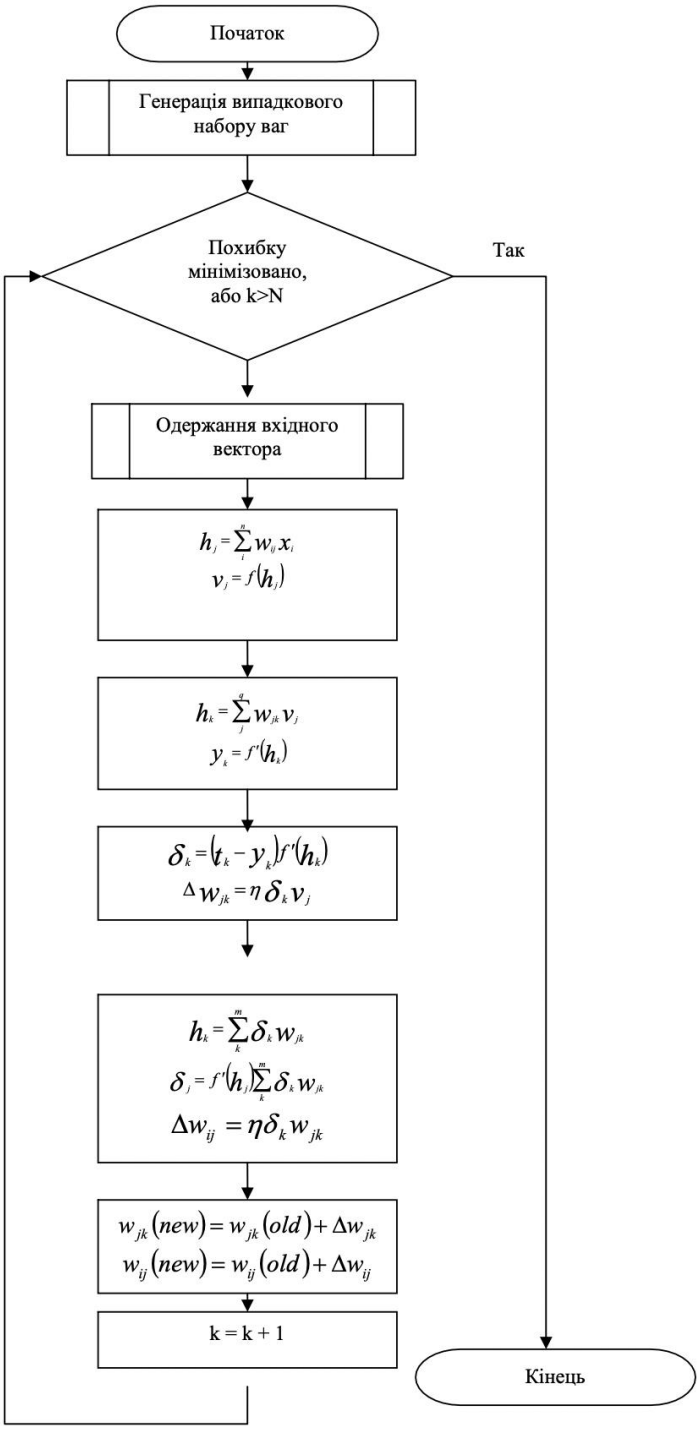


Рисунок 2.6 – Блок-схема алгоритму зворотного розповсюдження помилки [17]

Коригування ваг:

– Крок 9: Ваги між прихованим та вихідним шарами модифікуються так:

$$w_{jk}(new) = w_{jk}(old) + \Delta w_{jk} \quad (2.1)$$

Аналогічно корегуються ваги між вхідним та прихованим шарами:

$$w_{ij}(new) = w_{ij}(old) + \Delta w_{ij} \quad (2.2)$$

– Крок 10: Перевіряється умова зупинки: мінімізація похибки між потрібним та реальним виходом мережі.

ІІІ. КОНСТРУКТИВНИЙ РОЗДІЛ.
РОЗРОБКА ПРОЕКТНИХ РІШЕНЬ.

3.1. Проектування бази даних для інформаційних управляючих систем

База даних – це певний набір даних, які пов'язані між собою спільною ознакою або властивістю, та впорядковані, наприклад, за алфавітом [18].

Об'єднання великої кількості даних в єдину базу дає змогу для формування безлічі варіації групування інформації – особисті дані клієнта, історія замовлень, каталог товарів та будь-що інше.

Головною перевагою БД є швидкість внесення та використання потрібної інформації. Завдяки спеціальним алгоритмам, які використовуються для баз даних, можна легко знаходити необхідні дані всього за декілька секунд.

Щоб створити запит до бази даних часто використовують Structured Query Language. SQL дає змогу додавати, редагувати та видаляти інформацію, що міститься у таблицях [19].

В якості БД розробленої ІУС використовується реляційна база даних SQLit., Таблиці БД нормалізовані до третьої нормальної форми. (Рис. 3.1 – 3.2)

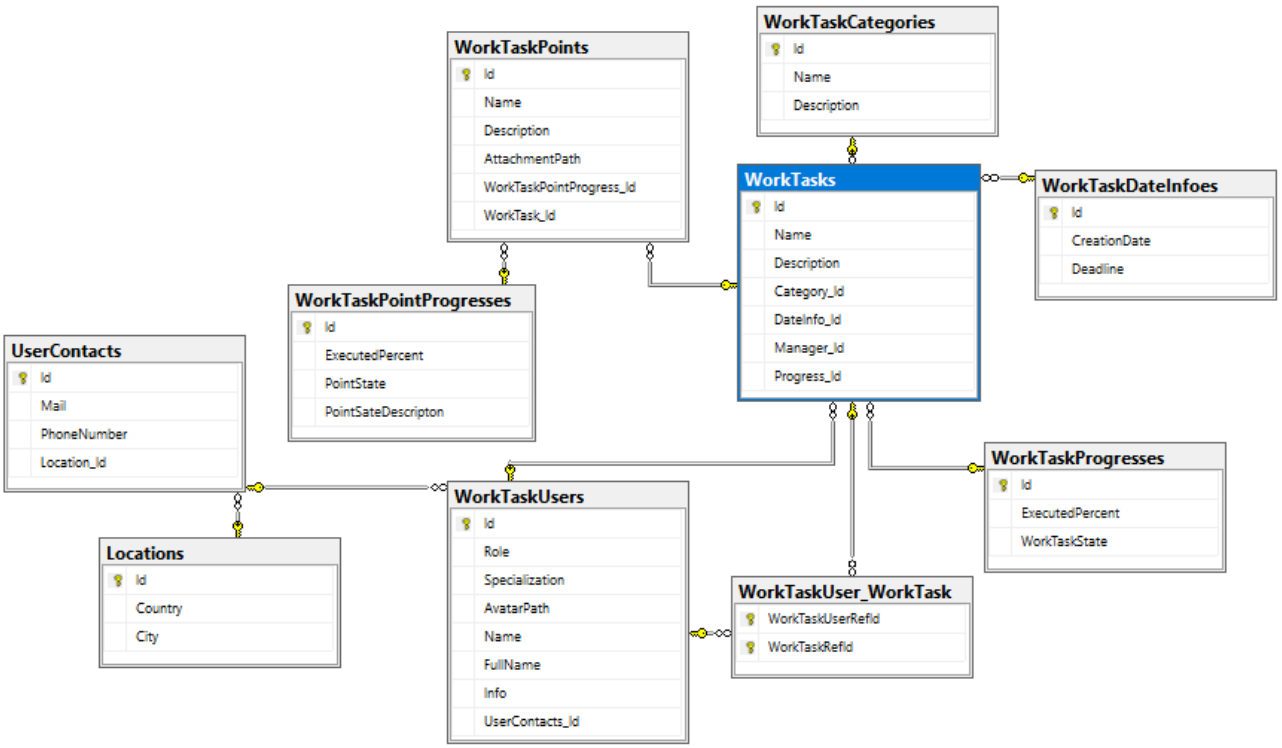


Рисунок 3.1 – Схема бази даних ІУС [Власна розробка]

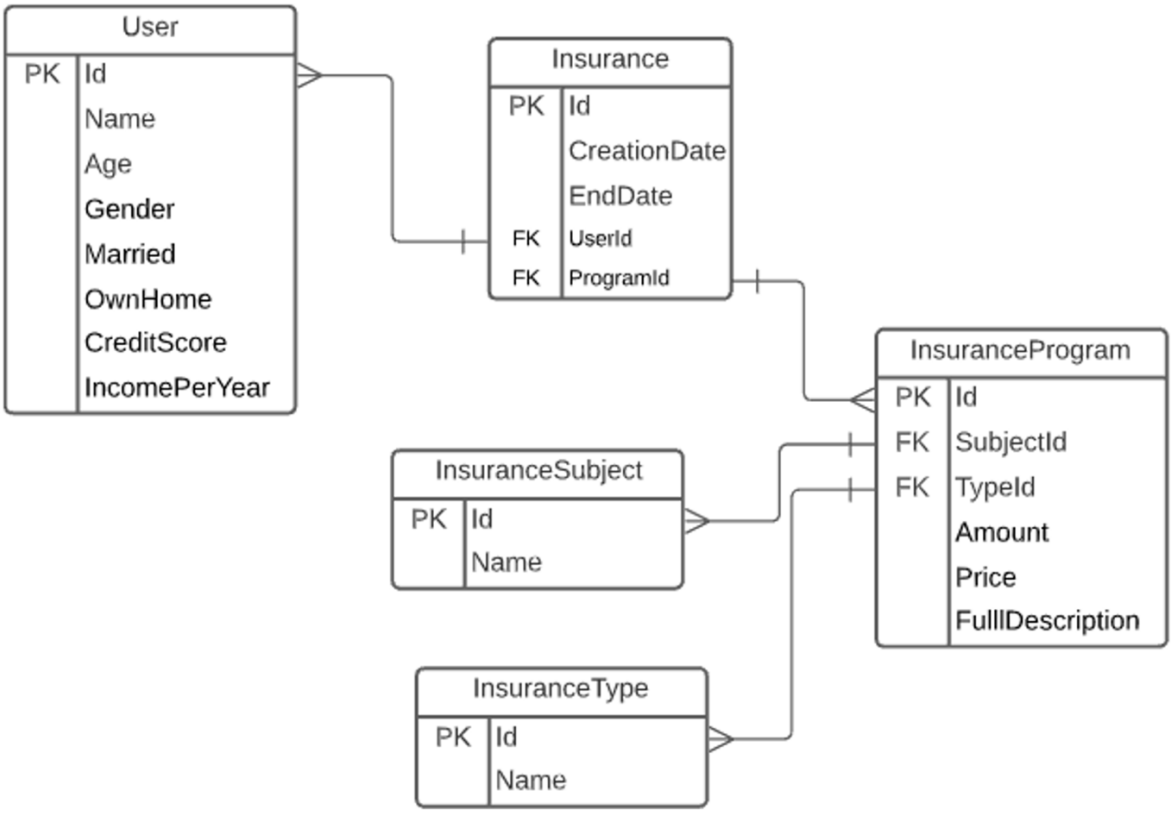


Рисунок 3.2 – Схема бази даних підсистеми ІУС з розрахунку суми кредитування [Власна розробка]

SQLite – це бібліотека на мові C, яка реалізує невеликий, швидкий, автономний, високонадійний, повнофункціональний механізм баз даних SQL.

SQLite – є найбільш використовуваним механізмом баз даних у світі. SQLite вбудований у всі мобільні телефони та більшість комп’ютерів і входить до складу безлічі інших програм, якими люди користуються щодня.

Нормалізація схеми бази даних — покроковий процес розбиття одного відношення (на практиці: таблиці) відповідно до алгоритму нормалізації на декілька відношень на базі функціональних залежностей.

Нормальна форма — властивість відношення в реляційній моделі даних, що характеризує його з точки зору надмірності, яка потенційно може призвести до логічно помилкових результатів вибірки або зміни даних. Нормальна форма визначається як сукупність вимог, яким має задовольняти відношення.

Третя нормальна форма (3НФ, 3NF) вимагає, аби дані в таблиці залежали винятково від основного ключа:

– Кожна таблиця повинна мати основний ключ: мінімальний набір колонок, які ідентифікують запис.
– Уникнення повторень груп (категорії даних, що можуть зустрічатись різну кількість разів в різних записах) правильно визначаючи неключові атрибути.
– Атомарність: кожен атрибут повинен мати лише одне значення, а не множину значень.
– Будь-яке поле, що залежить від основного ключа та від будь-якого іншого поля, має виноситись в окрему таблицю.

3.2 Інтелектуальний аналіз даних. Проектування та реалізація бази знань.

Існуюча проблема: забагато часу втрачається на обробку кожної заявки на страхування, адже, потрібно задіяти багатьох людей для отримання необхідної інформації по кожній заявці.

При розв’язанні здійснюється керування об’єктами: відділ продажів, менеджер, клієнт. Суб’єктом рішення виступає менеджер.

Вихідна інформація використовується відділом продажів для ефективної роботи та менеджером – для аналізу та моніторингу показників по наданим страховкам.

Цілі при прийнятті рішення: максимізація прибутку шляхом ефективного використання робочого часу співробітників.

Надамо інформацію щодо вихідних повідомлень (табл. 3.1).

Таблиця 3.1 – Перелік і опис вихідних повідомлень

Назва вихідного повідомлення	**Ідентифікатор**	**Форма подання і вимоги до неї**	**Термін видання і допустимий час затримки**	**Користувач інформації**
Звіт про обсяги страхування	ZVIT_1	Електронний документ	Щодня до 18:00	Аналітики, Відділ маркетингу
Заява щодо страхування	FORMA_1	Електронний документ	По мірі надходження	Бухгалтерія

Надамо інформацію щодо вхідних повідомлень (табл. 3.2).

Таблиця 3.2 – Перелік і опис вхідних повідомлень

Назва вхідного повідомлення	**Ідентифікатор**	**Форма подання**	**Термін і частота надходження**	**Джерело**
Довідник клієнтів	CLIENT	Електронний документ	По мірі надходження	Масив
Довідник страхових програм	IN_PR	Електронний документ	По мірі надходження	Масив
Довідник заявлень	IN	Електронний документ	По мірі надходження	Масив

Перелік та опис структурних одиниць інформації вхідних повідомлень (табл 3.3 – 3.5).

Таблиця 3.3 – Перелік і опис довідника клієнтів

Назва атрибута	**Ідентифікатор**	**Умовне позначення**	**Характеристика атрибута**
Ідентифікатор клієнта	CLIENT_ID	cid	Числовий
ФІО	CLIENT_NAME	cfn	Якісний, довідковий
Вік	CLIENT_AGE	cag	Числовий
Стать	CLIENT_G	cgd	Якісний, довідковий
Кредитна характеристика	CREDIT_S	ccd	Числовий
Дохід на рік	INCOME	cin	Числовий

Таблиця 3.4 – Перелік і опис довідника страхових програм

Назва атрибута	**Ідентифікатор**	**Умовне позначення**	**Характеристика атрибута**
Ідентифікатор страхової програми	IN_ID	inid	Числовий
Призначення	IN_SUBJ	insubj	Якісний, довідковий
Тип	IN_TYPE	intp	Якісний, довідковий
Сума страхування	IN_SUM	insum	Числовий
Ціна	IN_PRICE	inpr	Числовий
Опис	IN_DESC	indscr	Довідковий

Таблиця 3.5 – Перелік і опис страхових заявлень

Назва атрибута	**Ідентифікатор**	**Умовне позначення**	**Характеристика атрибута**
Ідентифікатор заявки	INZ_ID	inzid	Числовий
Дата створення	INZ_DATE	inzd	Якісний, довідковий
Ідентифікатор користувача	INZ_CLIENT	inzc	Числовий
Ідентифікатор програми	INZ_PR	inzpr	Числовий

3.3. Програмне забезпечення

Програмне забезпечення (ПЗ) є сукупністю програм, які реалізують мету й завдання інформаційної системи та забезпечують функціонування технічних засобів системи [19].

3.3.1 Вимоги до створення інформаційної управляючої системи

Основні вимоги до страхової інформаційної системи можна визначити так:

- облік всіх виданих бланків полісів;
- облік кожного полісу по всім страховим продуктам;
- зв’язок кожного полісу з бухгалтерськими платежами;
- централізований в межах компанії облік всіх клієнтів та застрахованих об’єктів;
- облік процесу врегулювання збитків;

- облік інформації в розрізі страхових ризиків;
- робота територіально віддалених офісів з загальною базою.

Для цього потрібно зробити систему з простим та зрозумілим інтерфейсом. Також, вона має працювати з різними рівнями доступу та надавати авторизованим користувачам швидкий та повний доступ до потрібної інформації в залежності від рівню користувача.

Крім того, інформаційна управляюча система повинна задовольняти ряду технічних вимог:

- бути надійно захищеною від несанкціонованого доступу;
- забезпечувати швидкість введення, обробки, пошуку інформації;
- мати зручний користувацький інтерфейс;
- мати можливість розвитку та оновлень системи, додання модулів;
- мати можливість вести аудит дій користувачів, тобто реєстрацію дій.

Системні вимоги, які були висунуті до інформаційної системи страхової компанії:

- доменне ім’я, символьне ім’я, що ідентифікує область в мережі інтернет, або локальної мережі;
- сервер хостингу, забезпечений швидким та якісним каналом зв’язку;
- програмне забезпечення серверу – веб-сервер, сервер баз даних, системи захисту даних від несанкціонованого доступу.

Основні вимоги до комплексної інформаційної системи страхової компанії:

1. Вся інформація знаходиться в єдиному інформаційному середовищі.
2. Інформація має бути доступною з комп'ютеру співробітнику незалежно від місцезнаходження.
3. Максимальний термін отримання інформації повинен бути визначеним.
4. Обмеження на інформацію визначаються лише правами доступу.

До інформаційної системи страхової компанії операторами заносяться первинні дані лише один раз. Все інше робить інформаційна система – створення

звітів, пошук інформації, контроль даних. При цьому можливості використання методик обробки та контролю практично необмежені.

3.3.2 Інструментальні засоби розробки для створення прикладного програмного забезпечення інформаційних управляючих систем

Для створення інформаційної системи у даній роботі було використано наступне програмне забезпечення:

1. ERWin Process Modeler

CA ERwin Data Modeler (раніше називався AllFusion Process Modeler) – програмний продукт в області реалізації засобів CASE технологій.

Інструмент для моделювання, аналізу, документування та оптимізації бізнес-процесів ERwin Process Modeler можна використовувати для графічного представлення бізнес-процесів. Він дозволяє проводити опис, аналіз і моделювання моделі даних – будівник мета-моделей даних. Він займає одне з лідируючих місць в своєму сегменті ринку.

Він включає три стандартні методології:

– IDEF0 (функціональне моделювання),
– DFD (моделювання потоків даних),
– IDEF3 (моделювання потоків робіт).

Методологія IDEF0, що є офіційним федеральним стандартом США, являє собою сукупність методів, правил і процедур, призначених для побудови функціональної моделі об'єкта якої-небудь предметної області. Функціональна модель IDEF0 відображає функціональну структуру об'єкта, тобто вироблені їм дії й зв'язки між цими діями. Методологія IDEF може використатися для моделювання широкого кола систем і визначення вимог і функцій, а потім для розробки системи, що задовольняє цим вимогам і реалізує ці функції. Для вже існуючих систем IDEF може бути використана для аналізу функцій, виконуваних системою, а також для вказівки механізмів, за допомогою яких вони здійснюються.

Ці методології по-своєму унікальні. Кожна з них може бути виконана окремо за допомогою BPwin, але їх сукупність укладена в модель дає аналітику повну картину предметної області клієнта [20].

ERwin є засобом концептуального моделювання БД, що використає стандарт IDEF1X. ERwin реалізує проектування схеми БД, генерацію її опису мовою цільовий СУБД і реінжиніринг існуючої БД. Методологія IDEF0, що є офіційним федеральним стандартом США, являє собою сукупність методів, правил і процедур, призначених для побудови функціональної моделі об'єкта якої-небудь предметної області.

2. LucidChart — це орієнтоване на веб програмне забезпечення, призначене для створення та редагування діаграм. LucidChart дозволяє користувачам об’єднуватися та працювати разом над створенням блоксхем, організаційних діаграм, шаблонів веб-сайтів, UML-діаграм, ментальних карт, прототипів програмного забезпечення та багатьох інших типів діаграм. Програма побудована на веб-стандартах HTML5 та Javascript, що підтримуються усіма новими браузерами, такими як Google Chrome, Firefox, Safari та Internet Explorer.

3. Visual Studio – інтегроване середовище розробки програмного забезпечення від фірми Microsoft. Він використовується для розробки комп'ютерних програм, а також веб- сайтів, веб-програм, веб-сервісів та мобільних додатків. Visual Studio використовує платформи для розробки програмного забезпечення Microsoft, такі як Windows API, Windows Forms, Windows Presentation Foundation, Windows Store і Microsoft Silverlight [21].

Visual Studio дає змогу розробнику написати єдину програму для роботи на кількох платформах Windows, таких як мобільний, настільний і навіть експериментальний середовище Microsoft HoloLens. Він також забезпечує спосіб створення додатків, які взагалі не працюють на комп'ютерах Windows, але замість цього вони працюють на пристроях iOS або у веб-додатках у хмарі.

Для розробки програмного веб-додатку використовуються наступні технології:

1. Мова серверної частини – C#.

Мова програмування C#, у якій поєднуються потужність і гнучкість універсальних мов програмування з високою ефективністю виконавчого коду й можливістю безпосереднього доступу до апаратних ресурсів комп’ютера – одна з найкращих мов програмування.

C# це об'єктно-орієнтована мова програмування з безпечною системою типізації для платформи .NET [22].

Незважаючи на те, що C # — самодостатня комп'ютерна мова, у неї особливі взаємозв’язки з середовищем .NET Framework. І на це є дві причини. По-перше, C# спочатку був розроблений компанією Microsoft для створення коду, що виконується в середовищі .NET Framework. По-друге, в цьому середовищі визначені бібліотеки, використовувані мовою C#. І хоча можна відокремити C# від .NET Framework, ці два середовища тісно пов'язані, тому дуже важливо мати загальне уявлення про .NET Framework і розуміти, чому це середовище настільки важливе для C#.

Оболонка .NET Framework визначає середовище для розробки і виконання сильно розподілених додатків, заснованих на використанні компонентних об'єктів. Вона дозволяє "мирно співіснувати" різним мовам програмування і забезпечує безпеку, переносимість програм і загальну модель програмування для платформи Windows. Важливо при цьому розуміти, що .NET Framework за своєю сутністю не обмежена застосуванням в Windows, тобто програми, написані для неї, можна потім переносити в середовища, відмінні від Windows. Зв'язок середовища .NET Framework з C# обумовлений наявністю двох дуже важливих засобів.

Одне з них, Common Language Runtime (CLR), являє собою систему, яка управляє виконанням призначених для користувача програм. CLR — це складова частина .NET Framework, яка робить програми багатоплатформними, підтримує багатомовне програмування і забезпечує безпеку. Другий засіб, бібліотека класів

.NET-оболонки, надає програмам доступ до середовища виконання. Наприклад, якщо вам потрібно виконати операцію вводу-виводу: відобразити що-небудь на екрані, то для цього необхідно використовувати .NET.

2. Фреймворк – ASP .NET Core MVC.

Одним з характерних моментів платформи ASP.NET Core є застосування патерну MVC. Невірно було б ототожнювати ASP.NET Core цілком з фреймворком ASP.NET Core MVC. Фреймворк ASP.NET Core MVC працює поверх платформи ASP.NET Core, і призначений для того, щоб спростити створення програми [23].

Концепція паттерна MVC передбачає поділ додатка на три компоненти:

– Модель (model): описує використовувані в додатку дані, а також логіку, яка пов'язана безпосередньо з даними, наприклад, логіку валідації даних. Як правило, об'єкти моделей зберігаються в базі даних. У MVC моделі представлені двома основними типами: моделі уявлень, які використовуються уявленнями для відображення і передачі даних, і моделі домену, які описують логіку управління даними. Модель може містити дані, зберігати логіку управління цими даними. У той же час модель не повинна містити логіку взаємодії з користувачем і не має визначати механізм обробки запиту. Крім того, модель не повинна містити логіку відображення даних в поданні.
– Подання (view): відповідають за візуальну частину або призначений для користувача інтерфейс, нерідко html-сторінка, через який користувач взаємодіє з додатком. Також уявлення може містити логіку, пов'язану з відображенням даних. У той же час уявлення не повинно містити логіку обробки запиту користувача або управління даними.
– Контролер (controller): представляє центральний компонент MVC, який забезпечує зв'язок між користувачем та програмою, поданням і сховищем даних. Він містить логіку обробки запиту користувача. Контролер отримує вводяться користувачем дані і обробляє їх. І в

залежності від результатів обробки відправляє користувачеві певний висновок, наприклад, у вигляді подання, наповненого даними моделей.

3. Розмітка – HTML, CSS.

HTML – (Hypertext Markup Language — Мова гіпертекстової розмітки) — це мова опису структури сторінок документів, яка дозволяє звичайний текст форматувати в абзаци, заголовки, списки та інші структури, створювати посилання на інші сторінки. Це текстова мова, в якій інструкції з форматування, що називаються тегами, вбудовані в розділи документа, які містять конкретну інформацію. Теги повідомляють браузерам, як форматувати і представляти інформацію на екрані.

CSS (Cascading Style Sheets - каскадні таблиці стилів) – одна з базових технологій у сучасному Інтернеті. Нечасто можна зустріти сайт, зверстаний без примінення CSS.

CSS-код – це список інструкцій для браузера, - як і де відображати елементи веб-сторінки, написаний особливим чином. Під «елементами» зазвичай маються на увазі теги XHTML / HTML і їх вміст.

4. Бібліотеки – EntityFramework, AutoMapper, Newtonsoft.Json

Entity Framework – це рішення для роботи з базами даних, яке використовується в програмуванні на мовах семейства.NET. Воно дозволяє взаємодіяти з СУБД за допомогою сутностей (entity), а не таблиць. Також код з використанням EF пишеться набагато швидше [24].

Entity Framework передбачає три можливі способи взаємодії з базою даних:

Database first: Entity Framework створює набір класів, які відображають модель конкретної бази даних

Model first: спочатку розробник створює модель бази даних, по якій потім Entity Framework створює реальну базу даних на сервері.

Code first: розробник створює клас моделі даних, які будуть зберігатися в бд, а потім Entity Framework за цією моделлю генерує базу даних і її таблиці

AutoMapper – це об'єкт-об'єктний картограф. Об'єктно-об'єктне відображення працює, перетворюючи вхідний об'єкт одного типу в вихідний

об'єкт іншого типу. Що робить AutoMapper цікавим, так це те, що він пропонує кілька цікавих конвенцій, щоб усунути брудну роботу, не з’ясовуючи, як зіставити тип A з типом B. Поки тип B відповідає встановленим правилам AutoMapper, для зіставлення двох типів потрібна майже нульова к онфігурація [25].

Простір імен Newtonsoft.Json надає класи, які використовуються для реалізації основних служб фреймворку. Перетворює об'єкт в JSON і з нього. JsonConverterAttribute. Вказує JsonSerializer використовувати вказаний JsonConverter під час серіалізації члена або класу.

Newtonsoft. Json використовує відображення для отримання параметрів конструктора, а потім намагається знайти найбільш близьке за назвою ці параметри конструктора до властивостей об'єкта. Він також перевіряє тип властивості та параметрів, які збігаються. Якщо збігу не знайдено, значення цього типового конструктора буде передано за замовчуванням [26].

ML.NET надає можливість додавати машинне навчання до програм .NET як в режимі онлайн, так і в режимі офлайн. За допомогою цієї можливості ви можете робити автоматичні прогнози, використовуючи дані, доступні для вашої програми. Програми машинного навчання використовують шаблони в даних для прогнозування, а не потребують явного програмування [27].

Центральним для ML.NET є модель машинного навчання. Модель визначає кроки, необхідні для перетворення вхідних даних у передбачення. За допомогою ML.NET ви можете навчити власну модель, вказавши алгоритм, або імпортувати попередньо навчені моделі TensorFlow та ONNX.

3.3.3 Архітектура програмного забезпечення

Так як C# високорівнева мова програмування загального застосування (в тому числі ООП), код написано з дотримання усіх принципів даного підходу:

1. Інкапсуляція – один з трьох основних механізмів об'єктно-орієнтованого програмування. Йдеться про те, що об’єкт вміщує не тільки дані, але і правила їх обробки, оформлені в вигляді виконуваних

фрагментів (методів). А також про те, що доступ до стану об'єкта напряму заборонено, і ззовні з ним можна взаємодіяти виключно через заданий інтерфейс (відкриті поля та методи), що дозволяє знизити зв'язність.

2. В об'єктно-орієнтованому програмуванні, успадкування (наслідування) — це механізм утворення нових класів на основі використання вже існуючих. При цьому властивості та функціональність батьківського класу переходять до класу нащадку (дочірнього).
3. Поліморфізм – властивість, яка дозволяє одне і те саме ім’я використовувати для вирішення декількох технічно різних задач, тобто основною метою поліморфізму є використання одного імені для задання загальних класу дій. На практиці це значить спроможність об’єктів вибирати внутрішній метод або процедуру, виходячи із типу даних, прийнятих в повідомленні.

Шаблони проектування програмного забезпечення — ефектні способи вирішення задач проектування програмного забезпечення. Шаблон не є закінченим зразком, який можна безпосередньо транслювати в програмний код. Об'єктно-орієнтований шаблон найчастіше є зразком вирішення проблеми і відображає відношення між класами та об'єктами, без вказівки на те, як буде зрештою реалізоване це відношення.

Застосування шаблонів (патернів):

1. Singleton — шаблон проектування, відноситься до класу твірних шаблонів. (рис. 3.3) Гарантує, що клас матиме тільки один екземпляр, і забезпечує глобальну точку доступу до цього екземпляру. (Приклад застосування: зберігання токену авторизації, див. додаток Б).

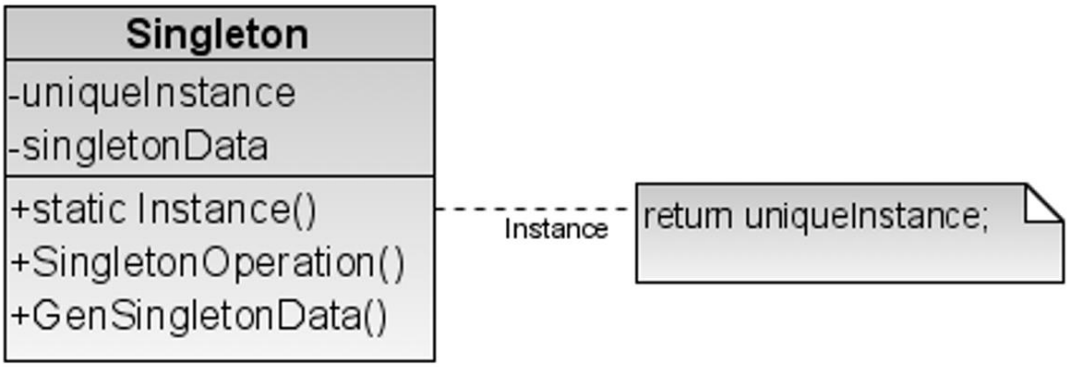


Рисунок 3.3 – Діаграма класів, що описує структуру шаблону singleton [28]

2. Builder — шаблон проектування, відноситься до класу твірних шаблонів. (рис 3.4) Відокремлює конструювання складного об'єкта від його подання, таким чином у результаті одного й того ж процесу конструювання можуть бути отримані різні подання. (Приклад застосування: «будівельник» email повідомлень, див. додаток В).

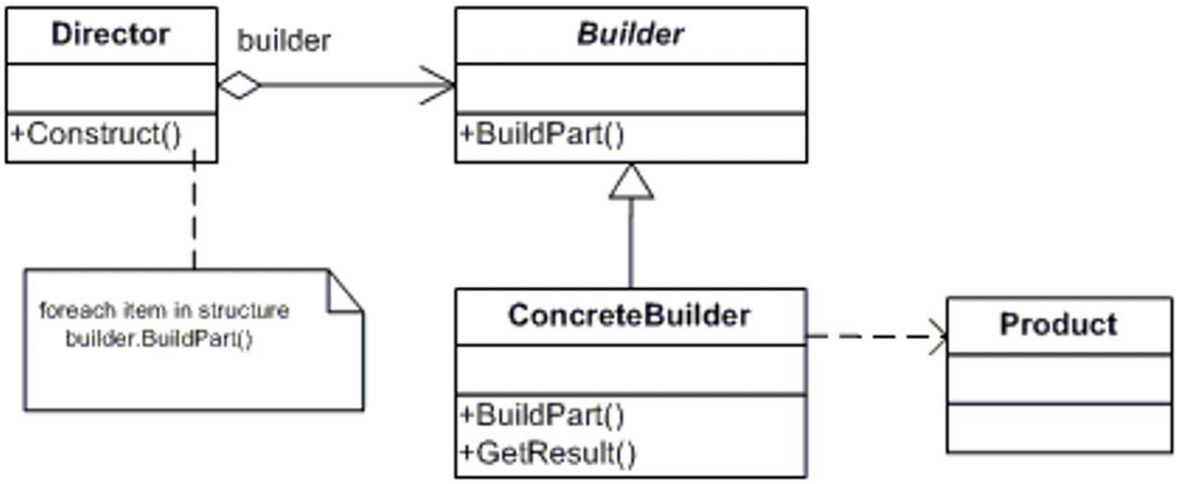


Рисунок 3.4 – UML діаграма, що описує структуру шаблону проектування Builder [Власна розробка]

3. Factory Method — шаблон проектування, відноситься до класу твірних шаблонів. (рис. 3.5) Визначає інтерфейс для створення об'єкта, але залишає підкласам рішення про те, який саме клас інстанціювати. Фабричний метод дозволяє класу делегувати інстанціювання підкласам. (Приклад застосування: «фабрика» темплейтів email повідомлень, див. додаток Г).

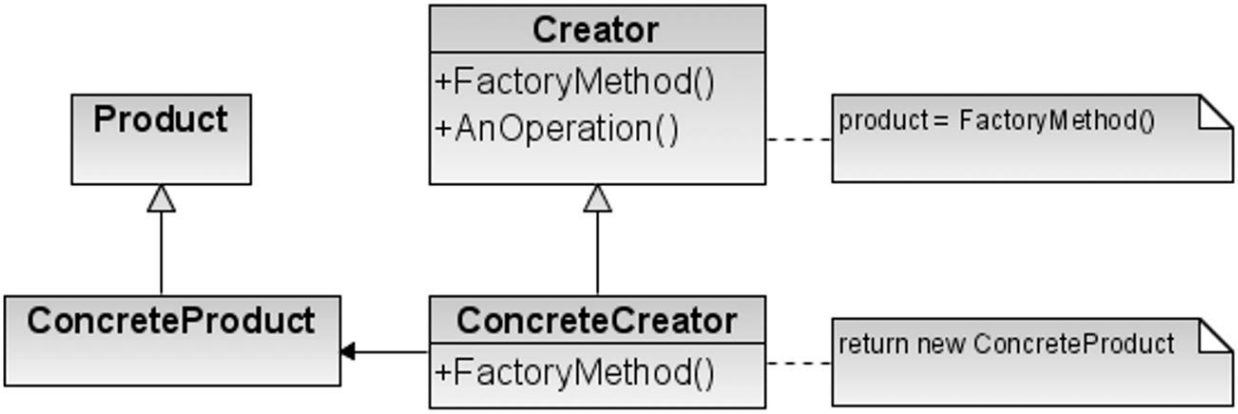


Рисунок 3.5 – UML діаграма, що описує структуру шаблону проектування Factory Method [28]

4. Facade — це структурний патерн проектування (рис. 3.6), який надає простий інтерфейс до складної системи класів, бібліотеки або фреймворку. (Приклад застосування: ApiClient для обмеження типів запитів, див. додаток Д).

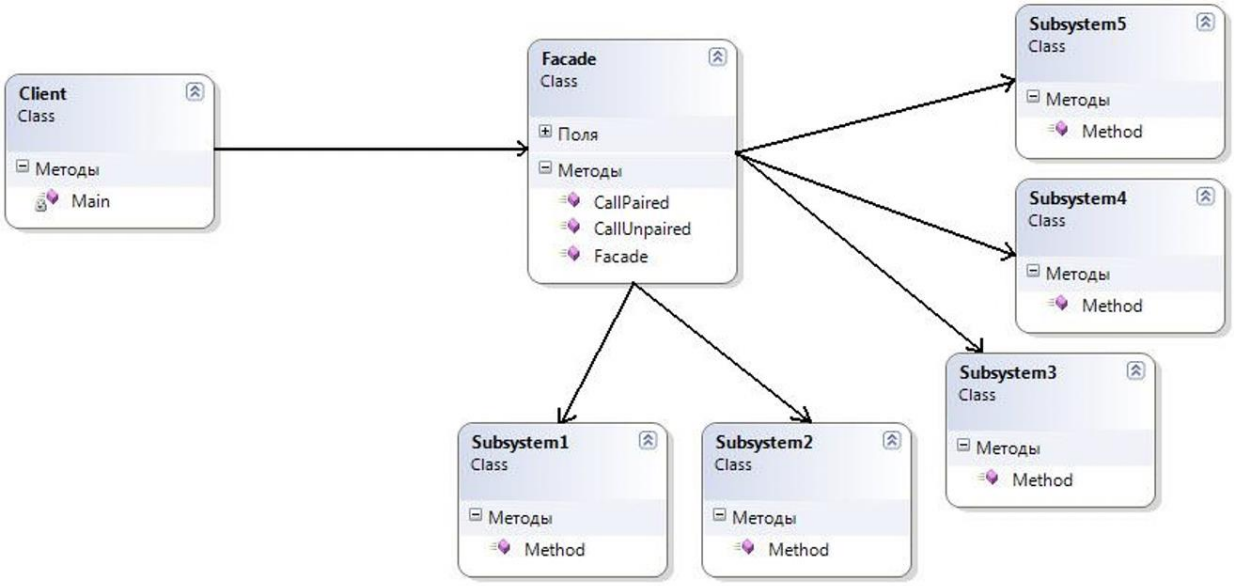


Рисунок 3.6 – UML діаграма, що описує структуру шаблону проектування Facade [Власна розробка]

5. Interprener для заданої мови визначає представлення її граматики (рис. 3.7), а також інтерпретатор речень цієї мови. (Приклад застосування: даний патерн використовується коли потрібна робота з AST (абстрактними синтаксичними деревами), реалізацією яких в середовищі .Net є Expression., див. додаток Е).

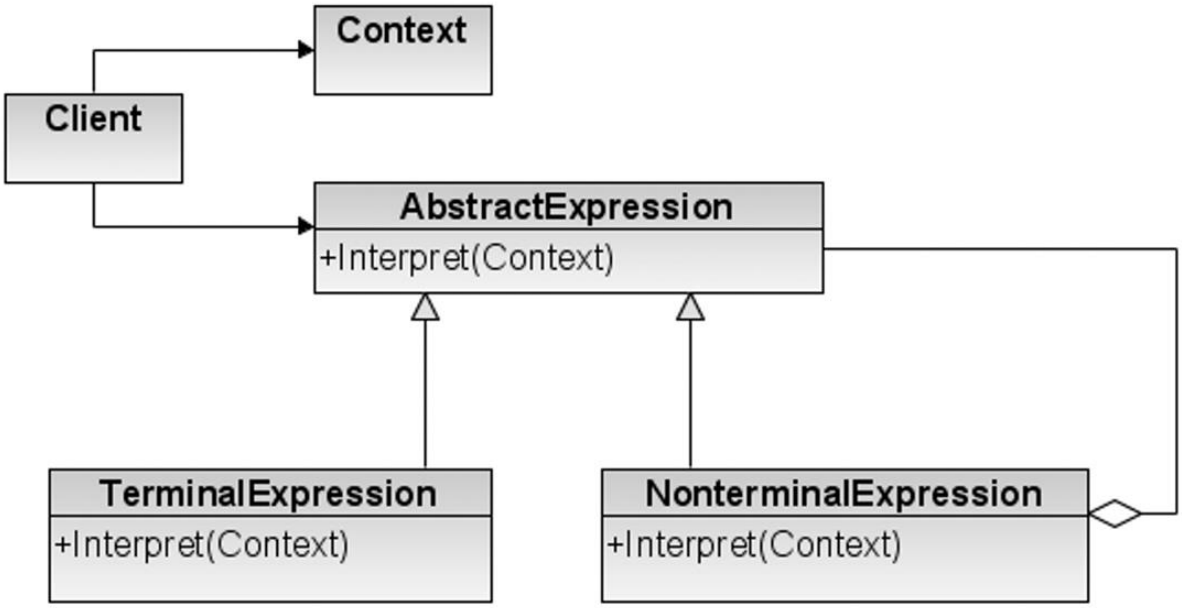


Рисунок 3.7 – UML діаграма, що описує структуру шаблону проектування Інтерпретатор [28]

Структурою проекту було обрано трирівневу Onion Architecture. (рис. 3.8)

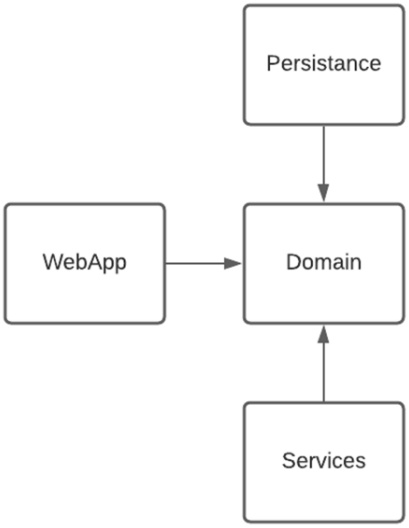


Рисунок 3.8 – Структура проекту [Власна розробка]

Onion Architecture заснована на принципі інверсії управління. Головна ідея цієї архітектури полягає у тому, що вона складається з кількох концентричних шарів, що взаємодіють один з одним у напрямку до ядра, що представляє домен (WebApp) [29].

Onion Architecture використовує концепцію рівнів:

– Рівень домену (Domain) знаходиться центральній частині архітектури, та представляє бізнес і об'єкти поведінки, він містить усі об’єкти домену програми;
– Рівень сховища (Persistance) створює абстракцію між сутностями домену та бізнес-логікою програми. На цьому рівні додаються інтерфейси, які забезпечують збереження об’єктів та поведінку отримання даних, як правило, за допомогою БД. На цьому рівні створюється загальний репозиторій, додаються ендпоінти та запити до БД для роботи із даними;
– Рівень служб (Services) містить інтерфейси зі звичайними операціями та використовується, переважно, саме для зв’язку між рівнем UI та рівнем сховища;
– Рівень інтерфейсу користувача (UI) зберігає периферійні проблеми, такі як інтерфейс користувача та веб-тести. Для веб-додатків він представляє веб-API або проект модульного тестування. Цей рівень має реалізацію принципу ін’єкції залежностей, тому програма створює слабко пов’язану структуру і може спілкуватися з внутрішнім рівнем через інтерфейси.

Також, надамо переваги та недоліки вибору саме такої архітектури (табл. 3.6).

Таблиця 3.6 – Переваги та недоліки Onion архітектури [30]

Переваги	**Недоліки**
– Гнучкість, – Зовнішні залежності знаходяться на зовнішніх рівнях, – Стійкість системи, – Відсутні залежності між внутрішнім та зовнішнім рівнями, – Швидкість тестування, адже ядро програми незалежне.	– Інтенсивне використання інтерфейсів, – Складність для початківців у встановленні зв’язків між рівнями.

Не дивлячись на недоліки, ця структура є найбільш привабливою для c#-розробників, тому саме вона полягла в основу розробки ІУС.

Основні сутності ІУС, у відповідності із поставленими вимогами, наведено на рис. 3.9.

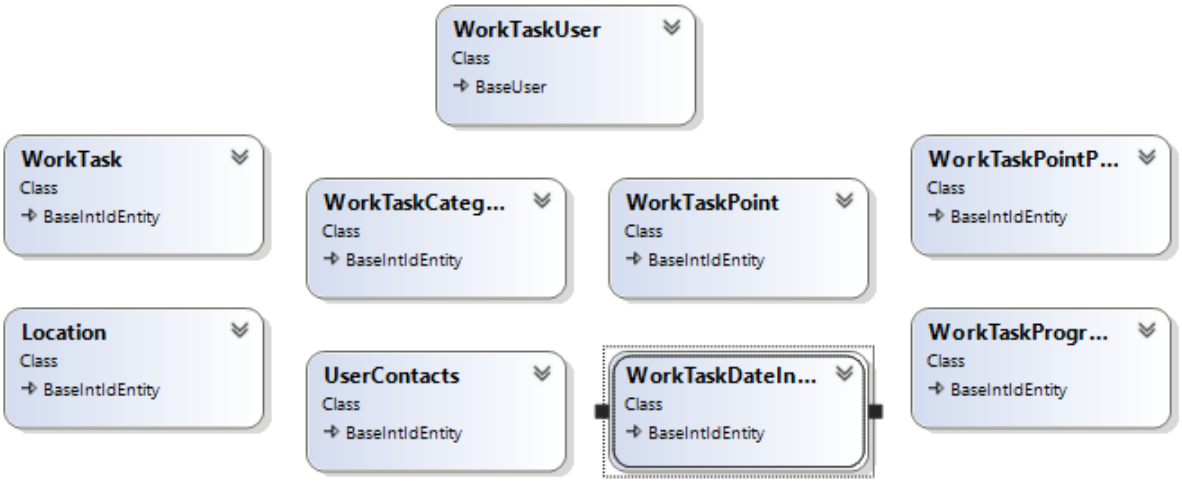


Рисунок 3.9 – Діаграма основних сутностей для ІУС [Власна розробка]

Підсистему прийняття рішення щодо розрахунку вартості страхування (страхової суми) було вирішено побудувати за такою схемою (рис. 3.10), а алгоритм наведено у додатку Ж.

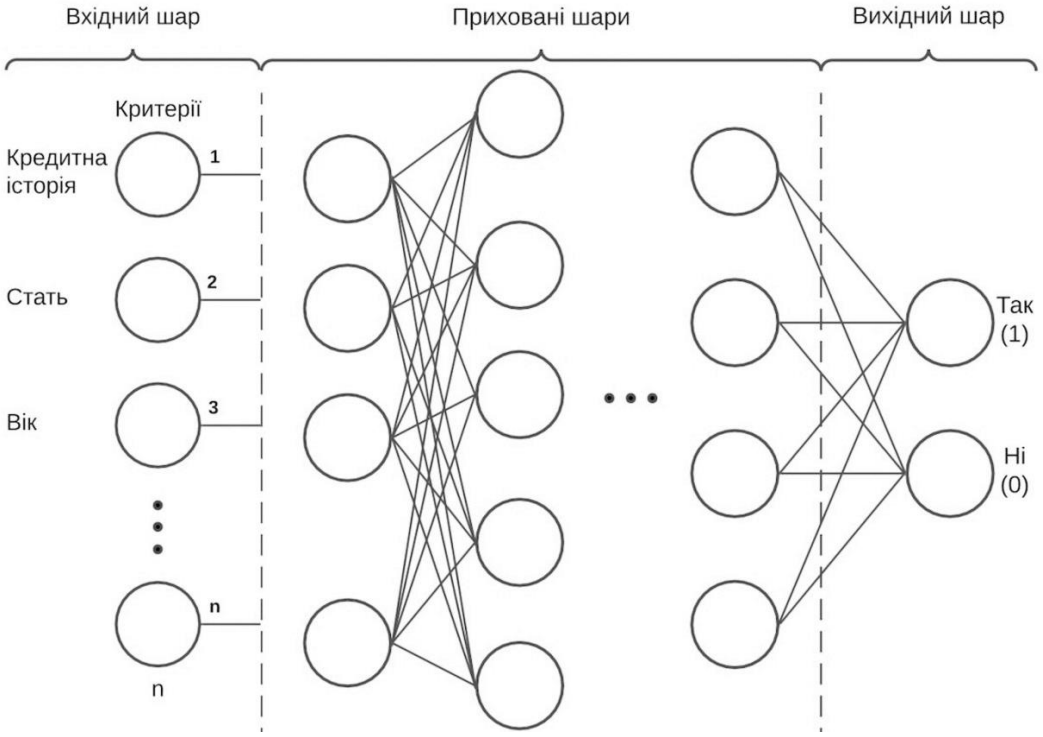


Рисунок 3.10 – Схема алгоритму машинного навчання підсистеми ІУС з визначення ціни страхування [Власна розробка]

3.3.4 Керівництво (інструкція) користувача

Відкриваючи програму у браузері, опинимось на головній сторінці додатку. (рис. 3.11)

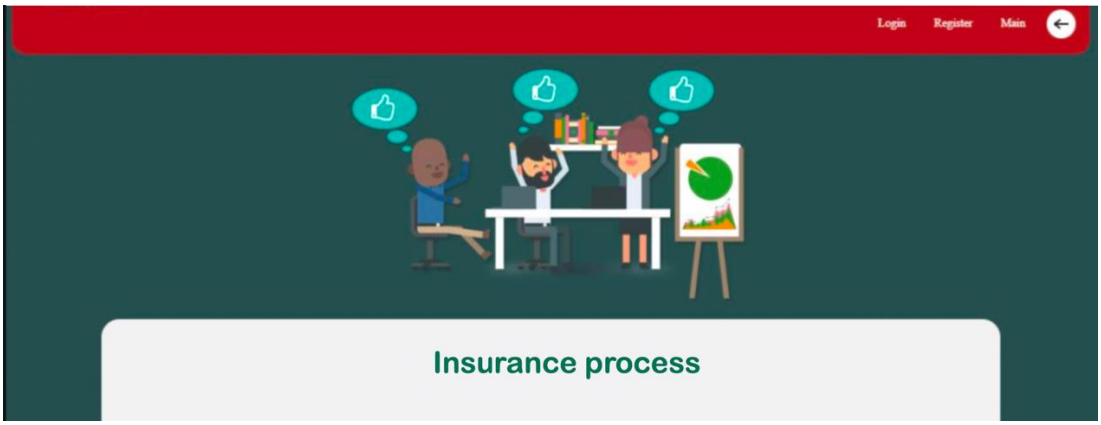


Рисунок 3.11 – Головна сторінка додатку

Для реєстрації нового користувача необхідно натиснути кнопку Register у верхньому меню, після чого опинимося на сторінці реєстрації користувача (рис. 3.12). Після введення даних користувача та вибору ролі, користувача успішно зареєстровано.

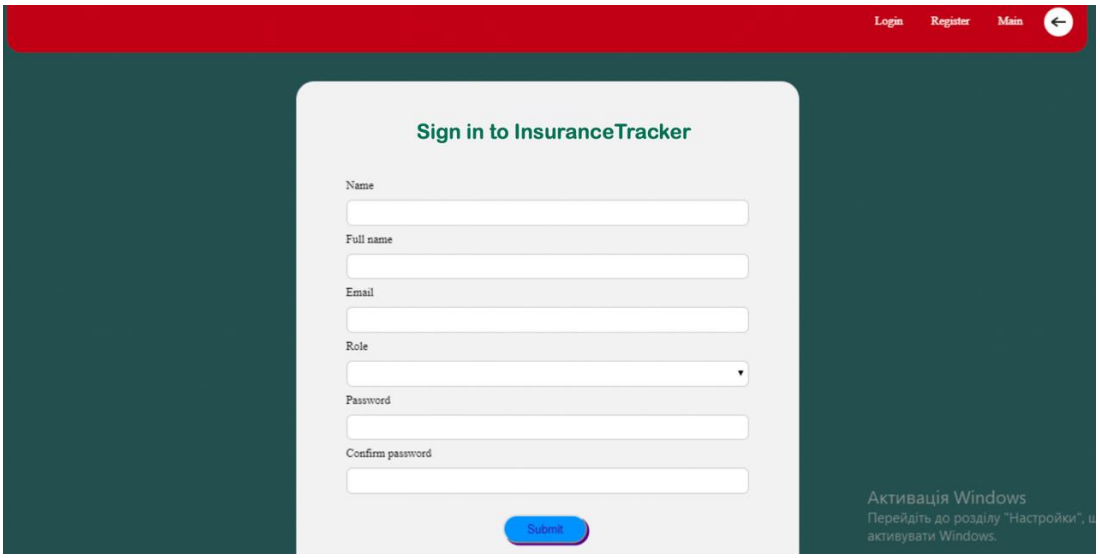


Рисунок 3.12 – Сторінка реєстрації користувача

Далі, опиняємось на сторінці входу в особистий кабінет (рис. 3.13) (на цю сторінку можливо дістатись з головної сторінки за натисканням на кнопку Login, якщо не потрібно реєструвати нового користувача).

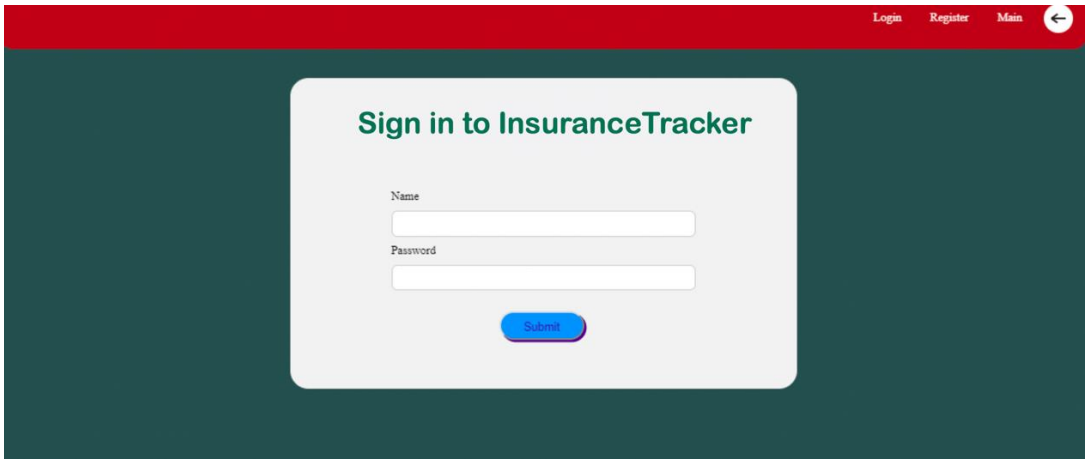


Рисунок 3.13 – Сторінка для входу в особистий профіль

На цій сторінці необхідно заповнити коректні дані існуючого користувача та натиснути на кнопку Submit. Після цього відбувається вхід у систему та користувач опиняється на основній сторінці особистого профілю (рис. 3.14).

У власному кабінеті користувач має можливість подивитись свою статистику виконання різних заявок, змінити свою інформацію (відправити заявку на погодження до менеджера), змінити налаштування.

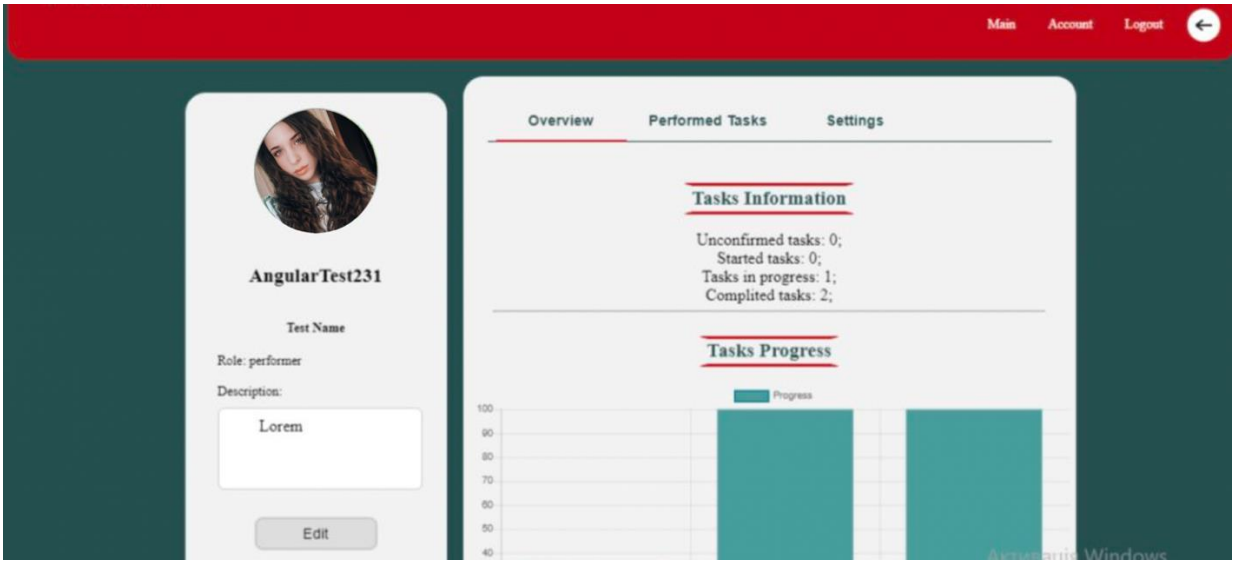


Рисунок 3.14 – Основна сторінка профілю

Так, при переході на Performed Tasks переходимо на сторінку профілю з завданнями конкретного страховика (рис. 3.15). Так, на прикладі бачимо (зеленим) заявку «Hello World», яка була виконана, та дві заявки (синім), що були назначені менеджером саме цьому працівнику страхової компанії.

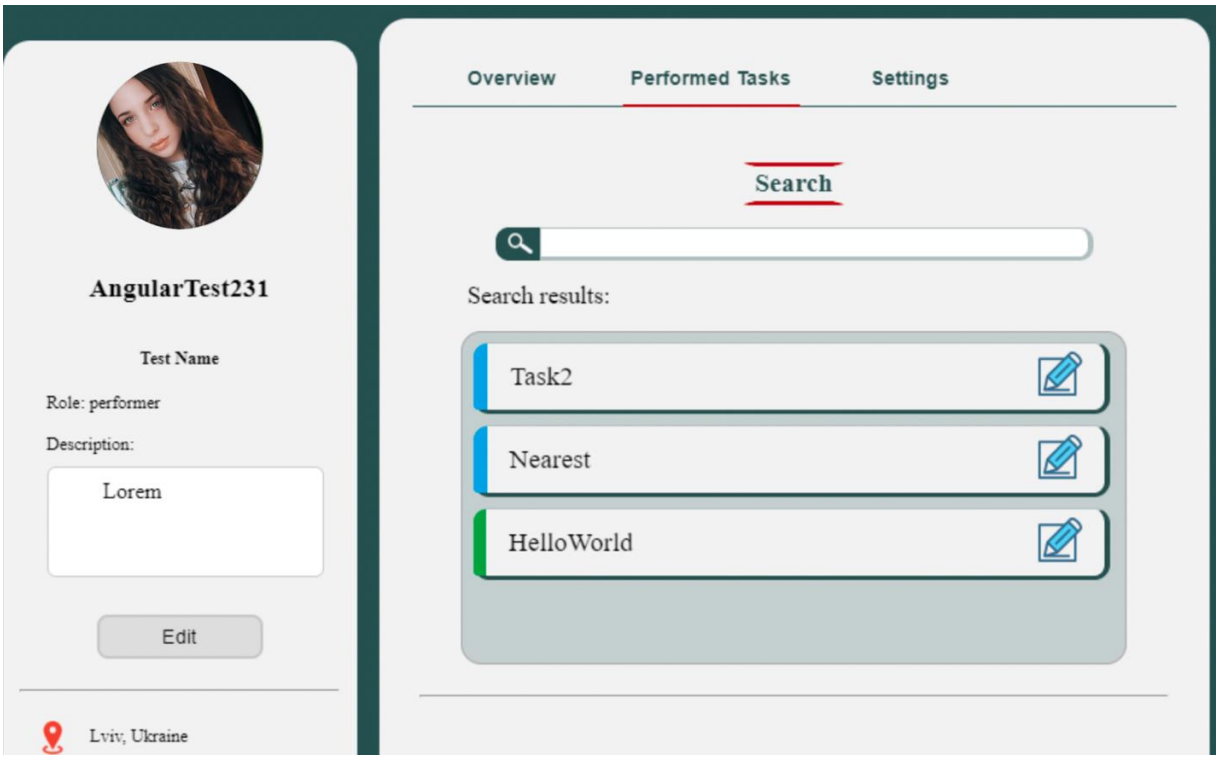
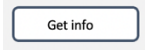


Рисунок 3.15 – Сторінка назначених страхових заявок менеджером працівнику

Для відкриття додаткової інформації щодо заявки (рис. 3.16) та можливості зміни її статусу натиснемо на кнопку  навпроти необхідної заявки. Тут бачимо дані, які можемо змінити та статус, який керівник матиме змогу побачити у власному кабінеті (рис. 3.18). За натисканням на кнопку 

переходимо на головну систему розрахунку страхового полісу (рис. 3.19).

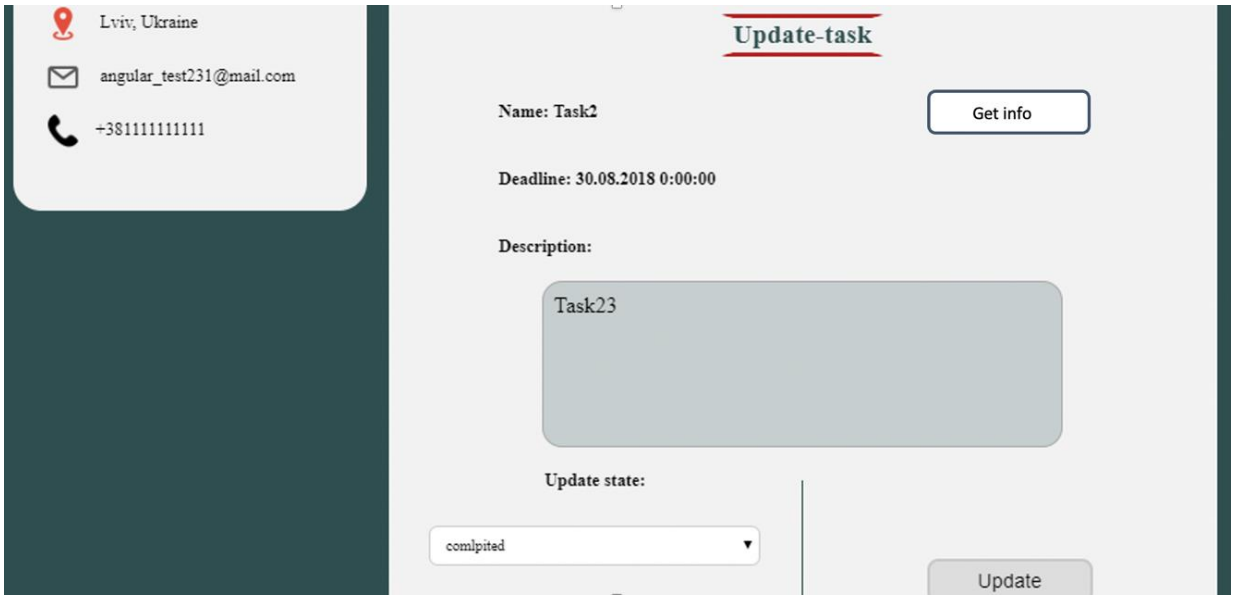


Рисунок 3.16 – Секція можливості змінити статус заявки

Далі переглянемо сторінку зміни особистих даних за натисканням на Settings (рис. 3.17), де можна відправити заявку до менеджера на зміну даних та завантажити нове фото профілю.

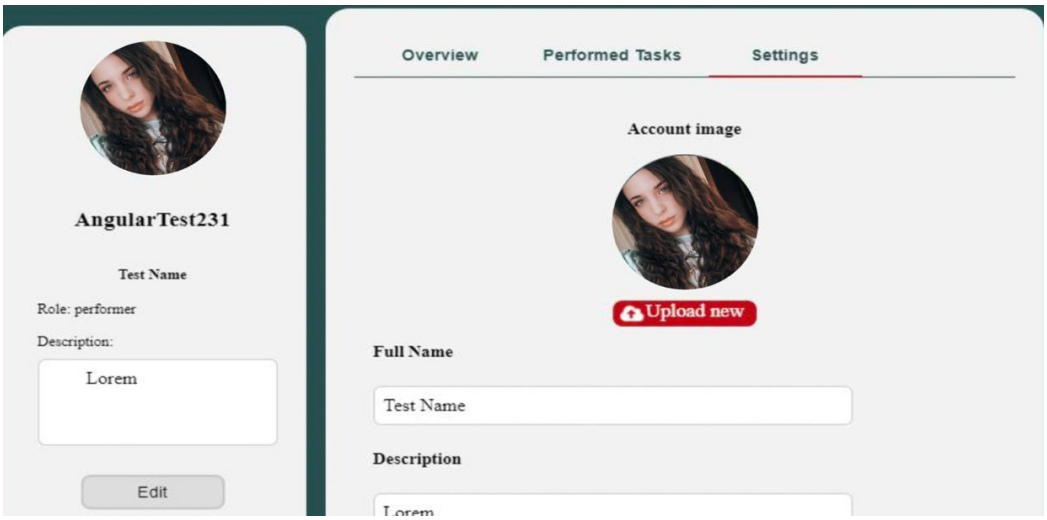


Рисунок 3.17 – Сторінка зміни особистих даних у профілі

Далі, розглянемо особисту сторінку користувача, з роллю «менеджер» (рис. 3.18), який має додатковий функціонал у вигляді перегляду інформації підлеглих, створення завдань для виконання ними та розподіл заявок від клієнтів між працівниками страхової компанії.

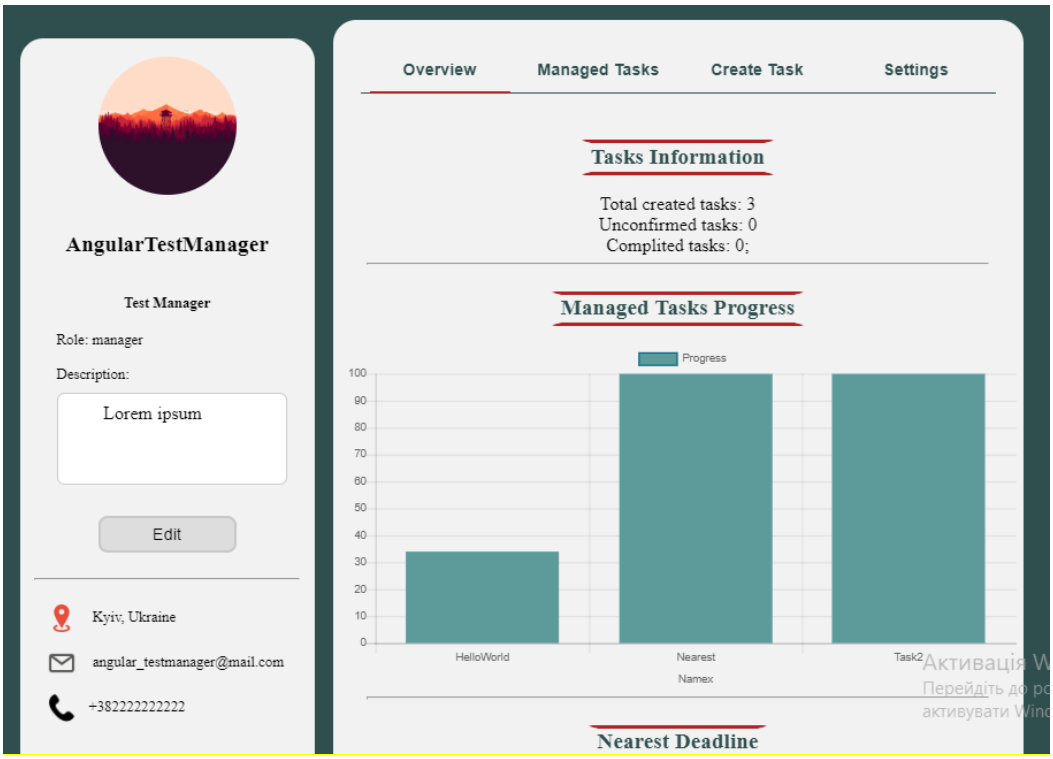
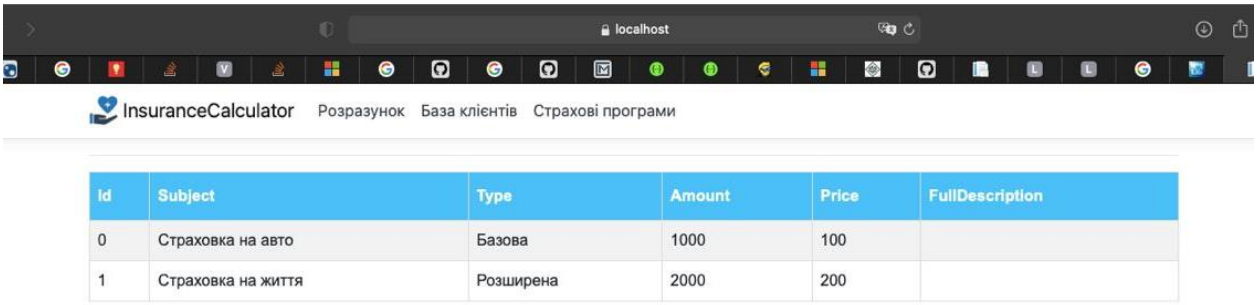


Рисунок 3.18 – Основна сторінка користувача з роллю «менеджер» для директора та відповідальних осіб

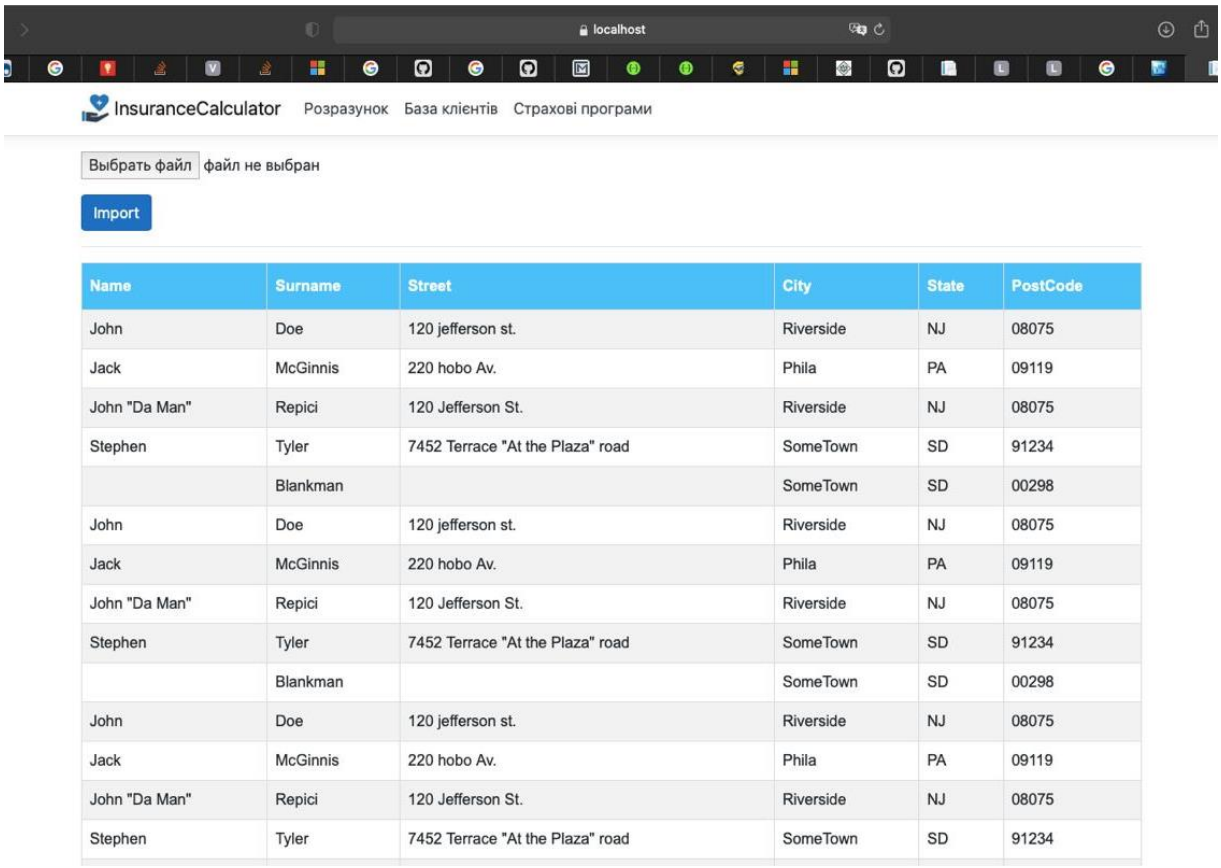
Тепер, опишемо підсистему визначення можливості надання страхування. Так, зі сторінки задачі менеджера відбувається перехід на головну сторінку «калькулятору страхування» (“InsuranceCalculator”). У пункті меню «Страхові програми» можемо побачити доступні на поточний момент програми страхування (рис. 3.19).



Id	Subject	Type	Amount	Price	FullDescription
0	Страховка на авто	Базова	1000	100	
1	Страховка на життя	Розширена	2000	200	

Рисунок 3.19 – Демонстрація завантажених страхових програм з БД

У вкладці «База клієнтів» при завантаженні необхідних даних з БД маємо можливість перевірити їх коректність (рис. 3.20).



Name	Surname	Street	City	State	PostCode
John	Doe	120 jefferson st.	Riverside	NJ	08075
Jack	McGinnis	220 hobo Av.	Phila	PA	09119
John "Da Man"	Repici	120 Jefferson St.	Riverside	NJ	08075
Stephen	Tyler	7452 Terrace "At the Plaza" road	SomeTown	SD	91234
	Blankman		SomeTown	SD	00298
John	Doe	120 jefferson st.	Riverside	NJ	08075
Jack	McGinnis	220 hobo Av.	Phila	PA	09119
John "Da Man"	Repici	120 Jefferson St.	Riverside	NJ	08075
Stephen	Tyler	7452 Terrace "At the Plaza" road	SomeTown	SD	91234
	Blankman		SomeTown	SD	00298
John	Doe	120 jefferson st.	Riverside	NJ	08075
Jack	McGinnis	220 hobo Av.	Phila	PA	09119
John "Da Man"	Repici	120 Jefferson St.	Riverside	NJ	08075
Stephen	Tyler	7452 Terrace "At the Plaza" road	SomeTown	SD	91234

Рисунок 3.20 – Головна сторінка підсистеми InsuranceCalculator із завантаженою БД

Для розрахунку суми страхування, переходимо у вкладку «Розрахунок» та натискаємо на кнопку «Розрахувати», маємо суму страхування (рис. 3.21), яка розрахована з врахуванням основних параметрів та особистості клієнту, що бажає оформити страхування майна чи особи.

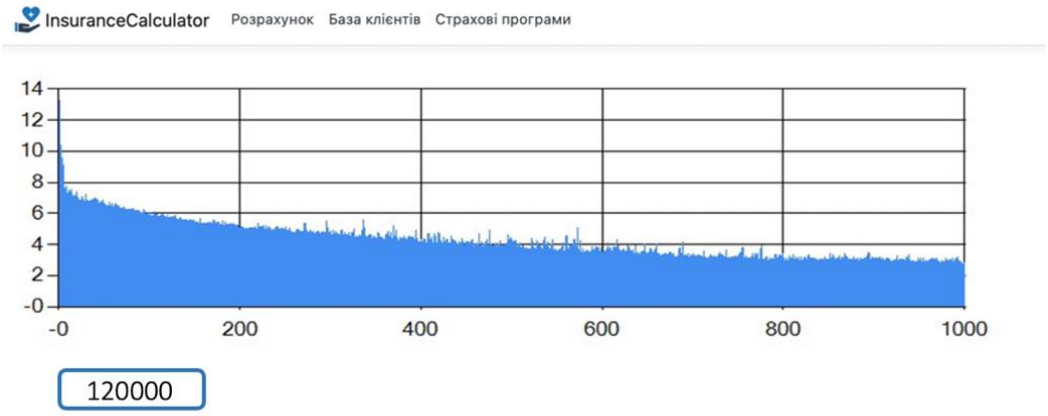


Рисунок 3.21 – Розрахунок страхової суми

3.4. Технічне забезпечення

Технічні засоби служать основою побудови ІС. Потужність засобів значною мірою визначає склад задач, що розв’язуються в даній предметній сфері. До технічного забезпечення ІС належать комп’ютерна техніка, засоби комунікації та оргтехніка. Іншими словами, технічне забезпечення — це комплекс взаємопов’язаних технічних засобів, призначених для збирання, нагромадження, оброблення, передавання, обміну та відображення інформації, потрібної для управління системою [17].

Інформаційна управляюча система для страхової компанії реалізовується за допомогою трирівневої клієнт-серверної архітектури, що передбачає наявність клієнтської частини, серверу та БД-серверу.

Комплекс технічних засобів забезпечує роботу автоматизованої системи, а його основним призначенням є:

– Забезпечення стійкості системи (її функціонування);
– Резервне копіювання (та, за необхідності, відновлення) даних;
– Ведення бази даних;
– Забезпечення необхідного рівня безпеки.

Тож, для розробки виконавчої системи необхідно мати персональний комп’ютер (з монітором, мишкою та клавіатурою) чи ноутбук з будь-якою операційною системою та мінімальними характеристиками:

– Процесором Intel Core i5, з частотою 1,6 ГГц;
– Оперативною пам’яттю 8 ГБ;
– Типом накопичувача SSD на 64 ГБ.

Для стабільної роботи системи необхідно мати сервери (фізичний чи віртуальний) з будь-якою операційною системою та мінімальними характеристиками:

– Процесором Intel Core i3, з частотою 1,6 ГГц;
– Оперативною пам’яттю 4 ГБ;
– Типом накопичувача HDD на 64 ГБ.

Для користування розробленою ІУС необхідно мати будь-який персональний комп’ютер (з монітором, мишкою та клавіатурою) чи ноутбук з будь-якою операційною системою, стабільним з’єднанням до інтернету та встановленим (хоча б одним) браузером.

3.5 Результати реалізації інформаційної системи

Програмне забезпечення ІУС було вирішено побудувати за допомогою Onion архітекури, так і результати реалізації можна з її допомогою, де ядро – 1 розділ, теоретичний, а наступні – «надбудови».

У першому – теоретичному – розділі проведено дослідження предметної області, визначено поняття, що стосуються процесу страхування, його мету, наведено основні складові ІУС та задачі стратегічного, тактичного та оперативного рівнів, які ІУС має виконувати. Описано існуючі підходи та технології у практиці створення ІУС та обрано найбільш релевантні для ІУС досліджуваної предметної області. Результатом другого підрозділу є чітке розуміння існуючих ІУС ринку, їх недоліків, для формулювання чіткого базиси для початку аналізу та проектування власної інформаційної системи.

У розділі ІІ – аналітичному – маємо чітке формулювання структури ІУС, на основі інформації розділу І, характеристику системи, визначаємо конкретну задачу, яку будемо вирішувати, із детальним описом та побудовою діаграм, та описуємо методи та моделі для реалізації.

Результатом останнього – конструктивного – розділу вже і є програмна реалізація ІУС, спроектовано та побудовано БД та БД, чітко описано вхідні та вихідні повідомлення, описано вимоги до системи та загальне програмне та технічне забезпечення при розробці ІУС, для зручності – наведено інструкцію користувача.

Спроектована інформаційна управляюча система для страхової компанії здатна покрити усі процеси страхової компанії, реалізоване програмне забезпечення є лише частиною системи і для повноцінного впровадження у роботу компанії необхідно ще багато доопрацювань. Наразі, розроблена інформаційна система може слугувати надбудовою на існуючу систему певної страхової компанії, адже вперше було запропоновано вирішення проблеми розрахунку страхової суми за допомогою штучного інтелекту.

Втім, на жаль, наразі навчання алгоритму відбувається за допомогою заздалегідь внесених даних, що не може на 100% відповідати запитам бізнесу, адже наразі, у рамках навчального проекту не можливо отримати доступ до баз даних банків та державних установ, але саме це, після впровадження системи у повсякденну роботу страхової компанії, слугуватиме надійним джерелом інформації для прийняття рішення щодо страхування та розрахунку суми страхової виплати.

ВИСНОВОК

В результаті проведених досліджень за темою магістерської кваліфікаційної роботи «Розроблення інформаційної управляючої системи для страхової компанії», можемо зробити наступні висновки:

1. охарактеризовано поняття страхування, страхового ринку, компанії, визначено мету страхової діяльності та задачі управління страховою діяльністю;
2. охарактеризовано чотири провідні системи для страхових компаній та з’ясовано, що ні одна не може повністю задовільнити потреби компаній;
3. проаналізовано існуючі підходи та технології до розробки ІУС та обґрунтовано вибір саме системного підходу та індустріальної автоматизованої технології проектування ІС;
4. визначено структуру ІУС, охарактеризовано її та надано деталізовану схему загальної архітектури ІС, надано схеми контекстної діаграми, її декомпозиції у IDEF0-нотації та BPMN-діаграму;
5. детерміновано алгоритм рішення проблеми визначення кредитної суми, визначено кроки та побудовано блок-схему алгоритму зворотного розповсюдження помилки;
6. описано патерни для загальної побудови ІУС;
7. надано схематичне зображення бази даних ІС та її підсистеми прийняття рішення щодо кредитної суми, діаграму основних сутностей ІУС та надано характеристику використаних технології проектування;
8. поставлено задачу, детерміновано та охарактеризовано вхідні та вихідні повідомлення у табличному вигляді для наглядності;
9. охарактеризовано програмне забезпечення та технології, які було застосовано при розробці ПЗ, висунуто конкретні вимоги до системи, побудовано та описано чітку архітектуру ПЗ з шаблонами проектування;

10. охарактеризовано технічне забезпечення та описано необхідні програмні засоби для розробки системи, її стабільної роботи та користування нею кінцевими користувачами;

Отже, як показують дослідження, розробка та впровадження прикладної інтелектуальної системи для страхової компанії є доцільним та необхідним, оскільки дозволить зменшити ризики для страхової компанії, чим збільшить ефективність робочого часу співробітників та, як наслідок, збільшить прибуток компанії, що і є ціллю, адже мета діяльності будь-якого бізнесу – максимізація прибутку .

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Investopedia: JULIA KAGAN. Insurance. [Електронне джерело] – режим доступу: https://www.investopedia.com/terms/i/insurance.asp
2. Юридический портал Протокол: Законы. [Електронне джерело] – режим доступу: https://protocol.ua/ru/pro_strahuvannya_stattya_1/
3. Pidru4niki: Визначення страхової компанії. [Електронне джерело] – режим доступу: https://pidru4niki.com/19170417/strahova_sprava/viznachennya_strahovoyi_kompaniyi
4. Букліб: Страхування. Конспект Лекцій. [Електронне джерело] – режим доступу: https://buklib.net/books/22752/
5. О.Гуйда. ІНФОРМАЦІЙНО-УПРАВЛЯЮЧІ СИСТЕМИ [Електронне джерело] – режим доступу: http://elartu.tntu.edu.ua/bitstream/123456789/12645/2/Conf_2015v1_Huida_O_I-Information_and_control_systems_73.pdf
6. Інтелектуальні системи підтримки прийняття рішень у страхуванні: потреби українських страхових компаній та їх задоволення / В. М. Гужва, О. С. Скрипова //Бізнес-інформ. – 2012. – № 3. – С. 183-187. [Електронне джерело] – режим доступу: http://nbuv.gov.ua/UJRN/binf_2012_3_51
7. InCore: BlackWater. [Електронне джерело] – режим доступу: http://in-core.com.ua/blackwater
8. Офіційний сайт компанії InsCom. [Електронне джерело] – режим доступу: https://www.ins.com.ua/products/InsCom
9. Офіційний сайт компанії fadata. [Електронне джерело] – режим доступу: https://www.fadata.eu/insurance-solutions
10. Офіційний сайт компанії fort.group. [Електронне джерело] – режим доступу: https://crsu.kiev.ua/strakhovanie/upravlenie-strakhovym-biznesom.html

11. Інформаційні системи і технології у фінансових установах. А.В.Олійник, В.М.Шацька - Навчальний посібник - Львів: "Новий Світ-2000", 2006 – 436 с. – Букліб. [Електронне джерело] – режим доступу: https://buklib.net/books/23562/#:~:text=%D0%A3%20%D1%82%D0%B5%D0%BE%D1%80%D1%96%D1%97%20%D1%82%D0%B0%20%D0%BF%D1%80%D0%B0%D0%BA%D1%82%D0%B8%D1%86%D1%96%20%D1%81%D1%82%D0%B2%D0%BE%D1%80%D0%B5%D0%BD%D0%BD%D1%8F,%D0%BF%D1%80%D0%B8%D0%B7%D0%B2%D0%BE%D0%B4%D0%B8%D1%82%D1%8C%20%D0%B4%D0%BE%20%D0%B7%D0%B1%D1%96%D0%BB%D1%8C%D1%88%D0%B5%D0%BD%D0%BD%D1%8F%20%D1%84%D1%83%D0%BD%D0%BA%D1%86%D1%96%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D0%B8%D1%85%20%D0%BC%D0%BE%D0%B6%D0%BB%D0%B8%D0%B2%D0%BE%D1%81%D1%82%D0%B5%D0%B9.
12. Studfile: Технологии проектирования ИС. [Електронне джерело] – режим доступу: https://studfile.net/preview/3997729/page:5/
13. Ситник В.Ф. Системи підтримки прийняття рішень: Навч. посіб. – К.: КНЕУ, 2009. – 614 с. [Електронне джерело] – режим доступу: http://kist.ntu.edu.ua/textPhD/sppr1.pdf
14. Вікіпедія: Страхування транспортних засобів [Електронне джерело] – режим доступу: https://uk.wikipedia.org/wiki/Страхування_транспортних_засобів
15. Tim Gibson. How is your car insurance calculated? – The telegraph. [Електронне джерело] – режим доступу: https://www.telegraph.co.uk/cars/road-safety/how-is-car-insurance-calculated/
16. Отримання знань: Об’єктно-орієнтовне програмування. [Електронне джерело] – режим доступу: https://disted.edu.vn.ua/courses/learn/4461
17. Калініна І.О., ДОСЛІДЖЕННЯ АЛГОРИТМІВ НАВЧАННЯ НЕЙРОННИХ МЕРЕЖ У ЗАДАЧАХ ПРОГНОЗУВАННЯ.

[Електронне джерело] – режим доступу: https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwiLgrXhydjwAhUOCRAIHYkICDAQFjAFegQIBRAD&url=http%3A%2F%2Fwww.irbis-nbuv.gov.ua%2Fcgi-bin%2Firbis_nbuv%2Fcgiirbis_64.exe%3FC21COM%3D2%26I21DBN%3DUJRN%26P21DBN%3DUJRN%26IMAGE_FILE_DOWNLOAD%3D1%26Image_file_name%3DPDF%2FNpchduct_2009_117_104_18.pdf&usg=AOvVaw2LhMCOiTTnF9NqAEb-_rHi

18. Кафедра АПАПЕС ТЕФ КПІ ім. І. Сікорського: Герасимик Іван. Що таке база даних? [Електронне джерело] – режим доступу: http://apeps.kpi.ua/shco-take-basa-danykh
19. Інформаційні системи і технології в обліку. Терещенко Л.О., Матієнко-Зубенко І.І. Навч. посіб. – К.: КНЕУ, 2004. – 187 с. – Букліб. [Електронне джерело] – режим доступу: https://buklib.net/books/22551/
20. Pro-Spo: ERwin Process Modeler (ранее BPwin) – моделирование,анализ и оптимизации бизнес-процессов [Електронне джерело] – режим доступу: https://pro-spo.ru/information-required-to-install/1702-erwin
21. Вікіпедія: LucidChart. [Електронне джерело] – режим доступу: https://uk.wikipedia.org/wiki/LucidChart
22. Борисюк О.Б., Переваги і недоліки мови програмування C#. [Електронне джерело] – режим доступу: http://www.rusnauka.com/21_NPN_2017/Informatica/2_226669.doc.htm
23. Metanit: ASP.NET Core MVC. Введение в MVC. [Електронне джерело] – режим доступу: https://metanit.com/sharp/aspnet5/3.1.php
24. Metanit: Введение в Entity Framework. Что такое Entity Framework. [Електронне джерело] – режим доступу: https://metanit.com/sharp/entityframework/1.1.php
25. Офіційний сайт бібліотеки AutoMapper [Електронне джерело] – режим доступу: https://docs.automapper.org/en/stable/Getting-started.html

26. StackOverflow: How does JSON deserialization in C# work [Електронне джерело] – режим доступу: https://stackoverflow.com/questions/41870132/how-does-json-deserialization-in-c-sharp-work
27. Microsoft: What is ML.NET and how does it work? [Електронне джерело] – режим доступу: https://docs.microsoft.com/en-us/dotnet/machine-learning/how-does-mldotnet-work
28. Вікіпедія: Шаблони проектування програмного забезпечення. [Електронне джерело] – режим доступу: https://uk.wikipedia.org/wiki/Шаблони_проєктування_програмного_забезпечення
29. Codeguru: Understanding Onion Architecture [Електронне джерело] – режим доступу: https://www.codeguru.com/csharp/understanding-onion-architecture/
30. Code-Maze: Onion Architecture in ASP.NET Core [Електронне джерело] – режим доступу: https://code-maze.com/onion-architecture-in-aspnetcore/

ДОДАТОК А

Програмна реалізація ІУС

FrontEnd кабінету користувача

```
div style="display: flex;">
  <div class="account-info-form">
      <img class="avatar" [src]="imageToShow" alt="logo image">
      <h2 style="text-align: center">{{currentUser['Name']}}</h2>
      <h4 style="text-align: center">{{currentUser['FullName']}}</h4>
      <div style="margin-left: 40px;">Role: {{currentUser['Role']}}</div>
      <br>
      <div style="margin-left: 40px;">
        Description: <br><textarea readonly *ngIf="hasDescription"
        class="form-input" style="height: 100px;
        width: 250px;
        font-size: 20px;
        resize: none;">
        {{currentUser['Info']}}
      </textarea>
      <span *ngIf="currentUser['Description']"><br>None<br><br></span>
      </div><br>
      <button class="settings-btn"
       (click)="this.changeTab('settings')">Edit</button>
      <br>
      <hr>
      <br>
      <div style="margin-left: 30px">
        <img style="vertical-align: middle" src="../assets/images/location-icon.png"
width="30px" height="30px" alt="">
        <span *ngIf="!currentUser['Country']" style="margin-left: 15px">None</span>
        <span *ngIf="currentUser['Country']" style="margin-left: 15px">
          {{currentUser['City']}}, {{currentUser['Country']}}</span>
      </div>
      <br>
      <div style="margin-left: 30px;">
          <img style="vertical-align: middle" src="../assets/images/mail.png" width="30px"
height="30px" alt="">
          <span style="margin-left: 15px;"> {{currentUser['Mail']}} </span>
      </div>
      <br>
      <div style="margin-left: 30px;">
          <img style="vertical-align: middle" src="../assets/images/phone.png" width="30px"
height="30px" alt="">
          <span *ngIf="!currentUser['PhoneNumber']" style="margin-left: 15px">None</span>
          <span *ngIf="currentUser['PhoneNumber']" style="margin-left:
15px">{{currentUser['PhoneNumber']}}</span>
      </div>
      <br>
  </div>

  <div class="account-form">
    <div class="tab-container">

      <button
       [ngClass]="mainTabClass"
       (click)="this.changeTab('main')">Overview
      </button>
       <button [ngClass]="performedTabClass"
       *ngIf="this.currentUser['Role'] == 'performer'"
        (click)="this.changeTab('performed')">Performed Tasks</button>
      <button *ngIf="this.currentUser['Role'] == 'manager'"
```

```
        [ngClass]="managedTabClass"
        (click)="this.changeTab('managed')">Managed Tasks</button>
      <button *ngIf="this.currentUser['Role'] == 'manager'"
        [ngClass]="createTabClass"
        (click)="this.changeTab('create-task')">Create Task</button>
      <button
      [ngClass]="settingsTabClass"
      (click)="this.changeTab('settings')">Settings</button>
    </div>

    <div *ngIf="selectedTab=='performed'">
      <app-performed-tasks></app-performed-tasks>
    </div>

    <div *ngIf="selectedTab=='managed'">
        <app-managed-tasks></app-managed-tasks>
      </div>

    <div *ngIf="selectedTab=='settings'">
        <app-user-settings [user]="currentUser"></app-user-settings>
    </div>

    <div *ngIf="selectedTab=='create-task'">
        <app-create-task></app-create-task>
    </div>

    <div *ngIf="selectedTab=='main'">
      <app-performer-profile *ngIf="this.currentUser['Role'] == 'performer'">
      </app-performer-profile>

      <app-manager-profile *ngIf="this.currentUser['Role'] == 'manager'">
        </app-manager-profile>
    </div>

  </div>
</div>


import { async, ComponentFixture, TestBed } from '@angular/core/testing';

import { UserProfileComponent } from './user-profile.component';

describe('UserProfileComponent', () => {
  let component: UserProfileComponent;
  let fixture: ComponentFixture<UserProfileComponent>;

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [ UserProfileComponent ]
    })
    .compileComponents();
  }));

  beforeEach(() => {
    fixture = TestBed.createComponent(UserProfileComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });
});

import { Component, OnInit } from '@angular/core';
```

```
import { TokenManagerService } from '../../core/services/token-manager.service';
import { WorkTaskUserService } from '../../core/services/work-task-user.service';
import { WorkTaskUser } from '../../core/models/work-task-user';
import { WorkTask } from '../../core/models/work-task';

@Component({
  selector: 'app-user-profile',
  templateUrl: './user-profile.component.html',
  styleUrls: ['../../../assets/scss/styles.scss']
})
export class UserProfileComponent implements OnInit {

  constructor(private userService: WorkTaskUserService) { }

  loading: boolean = false;
  currentUser: WorkTaskUser = {} as WorkTaskUser;
  hasDescription;
  hasLocation;
  hasPhone;
  userRole;

  tasks : WorkTask[];

  selectedTab = 'main';

  imageToShow: any = "../../../assets/images/Ajax-loader.gif";

  createImageFromBlob(image: Blob) {
    let reader = new FileReader();
    reader.addEventListener("load", () => {
      this.imageToShow = reader.result;
    }, false);

    console.log(image);

    if (image) {
      reader.readAsDataURL(image);
    }
  }

  ngOnInit() {
    this.userService.getCurrentUser().subscribe(x => {
      this.currentUser = x;
      this.hasLocation = x.Country;
      this.hasDescription = x.Info;
      this.hasPhone = x.PhoneNumber;
      this.userRole = x.Role;
    });

    this.userService.getCurrentImage().subscribe
    (
      x => {console.log("blob");
        this.createImageFromBlob(x);},
      err => {console.log(err);}
    );

  }

  mainTabClass = "tab-button-selected";
  managedTabClass = "tab-button";
  performedTabClass = "tab-button";
  settingsTabClass = "tab-button";
  createTabClass = "tab-button";

  changeTab(name: string)
```

```
  {
    this.mainTabClass = "tab-button";
    this.managedTabClass = "tab-button";
    this.performedTabClass = "tab-button";
    this.settingsTabClass = "tab-button";
    this.createTabClass = "tab-button";

    if(name == 'main')
    {
      this.mainTabClass = "tab-button-selected";
    }
    else if(name == 'create-task')
    {
      this.createTabClass = "tab-button-selected";
    }
    else if(name == 'performed')
    {
      this.performedTabClass = "tab-button-selected";
    }
    else if(name == 'managed')
    {
      this.managedTabClass = "tab-button-selected";
    }
    else if(name == 'settings')
    {
      this.settingsTabClass = "tab-button-selected";
    }
    this.selectedTab = name;
  }

}
```

Валідація полів

```
import { Component, Input, OnInit }  from '@angular/core';
import { FormGroup, FormControl, Validators }    from '@angular/forms';

import { QuestionBase }  from '../../core/models/questions/question-base';
import { AccountManagerService } from '../../core/services/account-manager.service';
import { RegistrationModel} from '../../core/models/registration-model';
import { TokenManagerService } from '../../core/services/token-manager.service';
import { Router } from '../../../../node_modules/@angular/router';

@Component({
  selector: 'app-registration-form',
  templateUrl: './registration-form.component.html',
  styleUrls: ['../../../assets/scss/styles.scss']
})
export class RegistrationFormComponent implements OnInit {

  @Input() questions: QuestionBase<any>[] = [];
  form: FormGroup;
  payLoad = '';
  private model: RegistrationModel;

  constructor(private accountService: AccountManagerService,
     private tokenManager: TokenManagerService,
     private router: Router) {  }

  public state: string = '';
  private message: string = '';

  ngOnInit() {
    this.form = this.toFormGroup(this.questions);
  }
```

```
  onSubmit() {
    this.state = "loading";
    if(!this.form.valid)
    {
      this.state="validationError";
      return;
    }

    if(this.form.value['Password'] != this.form.value['ConfirmPassword'])
    {
      this.state="passwordMatchingError";
      return;
    }

    if(this.form.value['Password'].length < 8)
    {
      this.state="passwordValidationError";
      return;
    }

    this.payLoad = JSON.stringify(this.form.value);

    this.model = this.form.value;

    this.accountService.register(this.model).subscribe(res => {
      this.tokenManager.setToken(res);
      this.state = 'success';
    },
    err => {this.state = 'apiError';},
    () => {this.router.navigate(['/user-profile']);});


  }

  result;

  private toFormGroup(questions: QuestionBase<any>[] ) {
    let group: any = {};

    questions.forEach(question => {
      group[question.key] = question.required ? new FormControl
      (question.value || '', Validators.required)
      : new FormControl(question.value || '');
    });

    return new FormGroup(group);
  }
}
```

Реалізація сервісу щодо закріплення за страхуванням менеджеру та усі операції над страхуванням

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using TaskTracker.Bll.Abstract.Services;
using TaskTracker.Bll.Impl.Services.Base;
using TaskTracker.Common.Enums;
using TaskTracker.Common.Results;
using TaskTracker.Dal.Abstract.Uof;
using TaskTracker.Dal.Impl.Ef.Specifications;
using TaskTracker.Dto;
```

```
using TaskTracker.Entities;
using TaskTracker.Mapping.Base;

namespace TaskTracker.Bll.Impl.Services
{
    /// <summary>
    /// Service that provides access to WorkTask's data.
    /// CRUD operations.
    /// </summary>
    public class WorkTaskService : UnitOfWorkBasedService, IWorkTaskService
    {
        private readonly IMapper<WorkTaskUser, WorkTaskUserDto>
            _workTaskUserDtoMapper;
        private readonly IMapper<WorkTask, WorkTaskDto> _workTaskDtoMapper;
        private readonly IMapper<WorkTaskPoint, WorkTaskPointDto>
            _workTaskPointDtoMapper;


        public WorkTaskService
            (
            IUnitOfWork unitOfWork,
            IMapper<WorkTaskUser, WorkTaskUserDto> workTaskUserDtoMapper,
            IMapper<WorkTask, WorkTaskDto> workTaskDtoMapper,
            IMapper<WorkTaskPoint, WorkTaskPointDto> workTaskPointDtoMapper
            ) : base(unitOfWork)
        {
            _workTaskUserDtoMapper = workTaskUserDtoMapper;
            _workTaskPointDtoMapper = workTaskPointDtoMapper;
            _workTaskDtoMapper = workTaskDtoMapper;
        }

        /// <summary>
        /// Add performer to the task.
        /// </summary>
        /// <param name="workTaskDto">WorkTask</param>
        /// <param name="performer"> Performer</param>
        /// <returns>Return Result of adding performer.</returns>
        public async Task<Result> AddPerformerAsync(WorkTaskDto workTaskDto,
            WorkTaskUserDto performer)
        {
            Result methodResult = new Result();

            var findedTask = await _unitOfWork.WorkTaskRepository
                .FirstAsync(x => x.Name == workTaskDto.Name);

            if (findedTask != null)
            {
                var findedUser = await _unitOfWork.WorkTaskUserRepository
                    .GetByMailAsync(performer.Mail);

                bool ifUserExist = findedUser != null;

                if (ifUserExist)
                {
                    findedUser.WorkTasks.Add(findedTask);
                    findedTask.WorkTaskUsers.Add(findedUser);

                    _unitOfWork.WorkTaskUserRepository.Update(findedUser);
                    _unitOfWork.WorkTaskRepository.Update(findedTask);
                }
                else
                {
                    methodResult.Success = false;
                    methodResult.Message = "User not found.";
```

```
        }
        methodResult.Success = true;
        methodResult.Message = $"Added performer(Name: {performer.Name})" +
            $" to work task(Name: {workTaskDto.Name}).";

        await _unitOfWork.SaveChangesAsync();
    }
    else
    {
        methodResult.Message = "Task not found.";
    }

    return methodResult;
}

/// <summary>
/// Add WorkTaskPoint to the WorkTask.
/// </summary>
/// <param name="workTaskDto"> WorkTask information</param>
/// <param name="workTaskPointDto"> WorkTaskPoint information</param>
/// <returns></returns>
public async Task<Result> AddWorkTaskPointAsync(WorkTaskDto workTaskDto,
    WorkTaskPointDto workTaskPointDto)
{
    Result methodResult = new Result();

    var findedTask = await _unitOfWork.WorkTaskRepository
        .FirstAsync(x => x.Name == workTaskDto.Name);

    if (findedTask != null)
    {
        var pointEntity = _workTaskPointDtoMapper.Map(workTaskPointDto);

        var findedTaskPoint = await _unitOfWork.WorkTaskPointRepository
            .FirstAsync(x => x.Name == pointEntity.Name
            && x.Description == pointEntity.Description);

        if (findedTask != null)
        {
            findedTask.WorkTaskPoints.Add(pointEntity);

            _unitOfWork.WorkTaskRepository.Update(findedTask);

            methodResult.Message = $"Added new work " +
                $"task point({pointEntity.Name})" +
                $"to work task({findedTask.Name})";
            methodResult.Success = true;
        }
        else
        {
            methodResult.Message = "Task not found.";
        }

        await _unitOfWork.SaveChangesAsync();
    }
    else
    {
        methodResult.Message = "Task not found.";
    }

    return methodResult;
}

/// <summary>
/// Change WorkTask Description.
```

```
/// </summary>
/// <param name="workTaskDto"> WorkTask information</param>
/// <param name="description"> Description</param>
/// <returns></returns>
public async Task<Result> ChangeDescriptionAsync
    (WorkTaskDto workTaskDto, string description)
{
    Result methodResult = new Result();

    var findedTask = await _unitOfWork.WorkTaskRepository
       .FirstAsync(x => x.Name == workTaskDto.Name);

    if (findedTask != null)
    {
        findedTask.Description = description;

        _unitOfWork.WorkTaskRepository.Update(findedTask);

        methodResult.Success = true;
        methodResult.Message = $"Description updated " +
            $"in Task({findedTask.Name})";

        await _unitOfWork.SaveChangesAsync();
    }
    else
    {
        methodResult.Message = "Task not fount.";
    }

    return methodResult;
}

/// <summary>
/// Change manager of the task.
/// </summary>
/// <param name="workTaskDto"> WorkTask information</param>
/// <param name="manager">Manager information</param>
/// <returns></returns>
public async Task<Result> ChangeManagerAsync(WorkTaskDto workTaskDto,
    WorkTaskUserDto manager)
{
    Result methodResult = new Result();

    var taskEntity = _workTaskDtoMapper.Map(workTaskDto);

    var findedTask = await _unitOfWork.WorkTaskRepository
        .FirstAsync(x => x.Name == taskEntity.Name);

    if (findedTask != null)
    {
        var findedUser = await _unitOfWork.WorkTaskUserRepository
            .GetByMailAsync(manager.Mail);

        bool userExist = findedUser != null;

        if (userExist)
        {
            if (findedUser.Role != (int)WorkTaskUserRoles.TaskManager)
            {
                methodResult.Message = "User not a manager.";
                return methodResult;
            }

            var allManagerTask = await _unitOfWork.WorkTaskRepository
                .GetAllTasksByManagerIdAsync(findedUser.Id);
```

```
            bool taskManagedByUser = allManagerTask.FirstOrDefault
                (x => x.Id == findedTask.Id) != null;

            if (!taskManagedByUser)
            {
                methodResult.Message = "Task not managed by current User.";
                return methodResult;
            }

            findedUser.WorkTasks.Add(findedTask);
            findedTask.Manager = findedUser;

            _unitOfWork.WorkTaskUserRepository.Update(findedUser);
            _unitOfWork.WorkTaskRepository.Update(findedTask);
        }
        else
        {
            methodResult.Success = false;
            methodResult.Message = "User not found.";

            return methodResult;
        }
        methodResult.Success = true;
        methodResult.Message = $"Added manager: {manager.Name}" +
            $" to Task: {taskEntity.Name}.";

        await _unitOfWork.SaveChangesAsync();
    }
    else
    {
        methodResult.Message = "Task not found.";
    }

    return methodResult;
}

/// <summary>
/// Create WorkTask.
/// </summary>
/// <param name="workTaskDto">WorkTask information</param>
/// <returns></returns>
public async Task<Result> CreateWorkTaskAsync(WorkTaskDto workTaskDto,
    WorkTaskUserDto manager)
{
    Result methodResult = new Result();

    var findedUser =
        await _unitOfWork.WorkTaskUserRepository
        .GetByMailAsync(manager.Mail);

    if (findedUser != null)
    {
        if (findedUser.Role != (int)WorkTaskUserRoles.TaskManager)
        {
            methodResult.Message = "User are not a manager.";
            return methodResult;
        }

        var allManagerTasks = await _unitOfWork.WorkTaskRepository
            .GetAllTasksByManagerIdAsync(findedUser.Id);


        var taskExist = allManagerTasks
            .FirstOrDefault(x => x.Name == workTaskDto.Name)
```

```
            != null;

        if (taskExist)
        {
            methodResult.Message
                = "Task with such name already exist.";
            return methodResult;
        }

        var taskEntity = _workTaskDtoMapper.Map(workTaskDto);

        await Task.Run(() =>
        {
            try
            {
                _unitOfWork.WorkTaskRepository.Add(taskEntity);
            }
            catch (Exception ex)
            {
                methodResult.Success = false;
                methodResult.Message = ex.Message;
            }
        });

        await _unitOfWork.SaveChangesAsync();

        methodResult.Success = true;
        methodResult.Message = $"Manager({manager.Name}) " +
            $"created Task({workTaskDto.Name})";
        return methodResult;
    }
    else
    {
        methodResult.Message = "User not found.";
        return methodResult;
    }

}

/// <summary>
/// Delete WorkTask by WorkTaskManager.
/// </summary>
/// <param name="workTaskDto"> WorkTask information</param>
/// <param name="manager"> WorkTaskUser information</param>
/// <returns></returns>
public async Task<Result> DeleteWorkTaskAsync(WorkTaskDto workTaskDto,
    WorkTaskUserDto manager)
{
    var result = new Result();

    var findedUser = await _unitOfWork
        .WorkTaskUserRepository.GetByMailAsync(manager.Mail);

    var findedTask = await _unitOfWork.WorkTaskRepository
        .FirstAsync(x => x.Name == workTaskDto.Name);

    if(findedUser == null)
    {
        result.Message = "User not found";
        return result;
    }

    if(findedUser.Role != (int)WorkTaskUserRoles.TaskManager)
    {
        result.Message = $"User({findedUser.Name})" +
```

```
                $"- is not a manager.";
            return result;
        }

        var managedTasks = await _unitOfWork.WorkTaskRepository
               .GetAllTasksByManagerIdAsync(findedUser.Id);

        bool isManagerOfCurrentTask = managedTasks
            .FirstOrDefault(x => x.Id == findedTask.Id) != null;

        if (isManagerOfCurrentTask)
        {
            _unitOfWork.WorkTaskRepository.Delete(findedTask);
            await _unitOfWork.SaveChangesAsync();

            result.Message = $"Work task({findedTask.Name}), " +
                $"deleted by user({findedUser.Name})";
            result.Success = true;
        }
        else
        {
            result.Message = result.Message = $"User({findedUser.Name})" +
                $"- is not a manager of work task({findedTask.Name}).";
        }

        return result;
    }

    public async Task<DataResult<IEnumerable<WorkTaskDto>>> GetByManagerName(string
name)
    {
        var result = new DataResult<IEnumerable<WorkTaskDto>>();

        var findedUser = await _unitOfWork.WorkTaskUserRepository
            .GetByNameAsync(name);

        if(findedUser == null)
        {
            result.Message = "User not found.";
            return result;
        }

        var workTasks = await _unitOfWork.WorkTaskRepository
            .GetAllTasksByManagerIdAsync(findedUser.Id);

        if(workTasks == null || workTasks.Count() == 0)
        {
            result.Message = "User has not managed tasks.";
            return result;
        }

        result.Success = true;
        result.Message = $"Work Tasks. Manager: {name}";
        result.Data = _workTaskDtoMapper.Map(workTasks);

        return result;
    }

    public async Task<DataResult<IEnumerable<WorkTaskDto>>>
        GetByPerformerName(string name)
    {
        var result = new DataResult<IEnumerable<WorkTaskDto>>();

        var findedUser = await _unitOfWork.WorkTaskUserRepository
            .GetByNameAsync(name);
```

```
    if (findedUser == null)
    {
        result.Message = "User not found.";
        return result;
    }

    var workTasks = await _unitOfWork.WorkTaskRepository
        .GetAllTasksByPerformerIdAsync(findedUser.Id);

    if (workTasks == null || workTasks.Count() == 0)
    {
        result.Message = "User has not performed tasks.";
        return result;
    }

    result.Success = true;
    result.Message = $"Work Tasks. Manager: {name}";
    result.Data = _workTaskDtoMapper.Map(workTasks);

    return result;
}

public async Task<DataResult<WorkTaskDto>> GetWorkTaskByIdAsync(int id)
{
    DataResult<WorkTaskDto> methodResult
       = new DataResult<WorkTaskDto>();

    WorkTask workTask = await _unitOfWork.WorkTaskRepository
        .FirstAsync(x => x.Id == id);

    if (workTask != null)
    {
        methodResult.Data = _workTaskDtoMapper.Map(workTask);
        methodResult.Success = true;

        methodResult.Message = $"WorkTask: ({workTask.Name})";
    }
    else
    {
        methodResult.Message = $"Not Found.";
    }

    return methodResult;
}

/// <summary>
/// Get WorkTaskiInformation.
/// </summary>
/// <param name="name"> WorkTask name </param>
/// <param name="manager"></param>
/// <returns></returns>
public async Task<DataResult<WorkTaskDto>> GetWorkTaskByNameAsync
    (string name)
{
    DataResult<WorkTaskDto> methodResult
        = new DataResult<WorkTaskDto>();

    WorkTask workTask = await _unitOfWork.WorkTaskRepository
        .FirstAsync(x => x.Name == name);

    if (workTask != null)
    {
        methodResult.Data = _workTaskDtoMapper.Map(workTask);
        methodResult.Success = true;
```

```
                methodResult.Message = $"WorkTask: ({name})";
            }
            else
            {
                methodResult.Message = $"Not Found.";
            }

            return methodResult;
        }

        public async Task<Result> UpdateWorkTaskAsync(WorkTaskDto workTaskDto,
            WorkTaskUserDto manager)
        {
            var result = new Result();

            var findedUser = await _unitOfWork
                .WorkTaskUserRepository.GetByMailAsync(manager.Mail);

            var findedTask = await _unitOfWork.WorkTaskRepository
                .FirstAsync(x => x.Name == workTaskDto.Name);

            if (findedUser == null)
            {
                result.Message = "User not found";
                return result;
            }

            if (findedUser.Role != (int)WorkTaskUserRoles.TaskManager)
            {
                result.Message = $"User({findedUser.Name})" +
                    $"- is not a manager.";
                return result;
            }

            var managedTasks = await _unitOfWork.WorkTaskRepository
                   .GetAllTasksByManagerIdAsync(findedUser.Id);

            bool isManagerOfCurrentTask = managedTasks
                .FirstOrDefault(x => x.Id == findedTask.Id) != null;

            if (isManagerOfCurrentTask)
            {
                if (!string.IsNullOrEmpty(workTaskDto.Name))
                    findedTask.Name = workTaskDto.Name;

                if (!string.IsNullOrEmpty(workTaskDto.Description))
                    findedTask.Description = workTaskDto.Description;

                _unitOfWork.WorkTaskRepository.Update(findedTask);
                await _unitOfWork.SaveChangesAsync();

                result.Message = $"Work task({findedTask.Name}), " +
                    $"deleted by user({findedUser.Name})";
                result.Success = true;
            }
            else
            {
                result.Message = result.Message = $"User({findedUser.Name})" +
                    $"- is not a manager of work task({findedTask.Name}).";
            }
            return result;
        }
    }
}
```

ДОДАТОК Б

Приклад реалізації Singletone для токену авторизації

```
using RestSharp;
using System;

namespace TaskTracker.Mobile.Infrastructure.Authorization
{
    public class AuthToken
    {
        private static string _endpoint;
        private static Lazy<AuthToken> _instance;
        private static IRestClient _restClient;

        public string Value { get; }

        private AuthToken(string value) { Value = value; }

        public static AuthToken Get => _instance?.Value ?? Update;

        public static AuthToken Update => (_instance = new
        Lazy<AuthToken>(GetTokenFromAPI)).Value;

        private static AuthToken GetTokenFromAPI()
        {
            var client = _restClient ?? (_restClient = new RestClient(_endpoint));
            var request = new RestRequest(Method.POST);
            var value = client.Execute(request).Content;

            return new AuthToken(value);
        }

        public static void ConfigureEndpoint(string endpoint) => _endpoint = endpoint;
    }
}
```

ДОДАТОК В

Приклад реалізації Builder для email повідомлень

```
using System.Collections.Generic;
using TaskTracker.Messaging.Entities;

namespace TaskTracker.Messaging.Builders
{
    /// <summary>
    /// Base mail entity builder implementation.`
    /// </summary>
    public class MailBuilder : IMailBuilder
    {
        private MailEntity _mail;

        public MailBuilder()
        {
            _mail = new MailEntity();
        }

        public IMailBuilder AddAttachment(string path)
        {
            _mail.Data.Attachments.Add(path);
            return this;
        }

        public IMailBuilder AddAttacments(IEnumerable<string> attachments)
        {
            _mail.Data.Attachments.AddRange(attachments);
            return this;
        }

        public IMailBuilder AddFromMail(string fromMail)
        {
            _mail.From = fromMail;
            return this;
        }

        public IMailBuilder AddFromName(string fromName)
        {
            _mail.FromName = fromName;
            return this;
        }

        public IMailBuilder AddHtml(string html)
        {
            _mail.Data.IsHtml = true;
            _mail.Data.Text = html;
            return this;
        }

        public IMailBuilder AddSubject(string subject)
        {
            _mail.Data.Subject = subject;
            return this;
        }

        public IMailBuilder AddText(string text)
        {
            _mail.Data.IsHtml = false;
            _mail.Data.Text = text;
            return this;
```

```
        }

        public IMailBuilder AddToMail(string toMail)
        {
            _mail.To = toMail;
            return this;
        }

        public IMailBuilder AddToName(string toName)
        {
            _mail.ToName = toName;
            return this;
        }

        public IMailBuilder AddSystemPart(SystemMailEntity systemPart)
        {
            _mail.From = systemPart.From;
            _mail.FromName = systemPart.FromName;
            _mail.To = systemPart.To;
            _mail.ToName = systemPart.ToName;
            return this;
        }

        public MailEntity Build()
        {
            return _mail;
        }

        public IMailBuilder Clear()
        {
            _mail = new MailEntity();
            return this;
        }

        public IMailBuilder AddAttacments(params string[] attachments)
        {
            return AddAttacments(attachments);
        }
    }
}
```

ДОДАТОК Г

Приклад реалізації Factory Method для email повідомлень

Базовим абстрактним класом є MessageTemplate в якому фабричним методом є GetMail()

```
using TaskTracker.Bll.Abstract.Messaging.Template;
using TaskTracker.Common.Enums;
using TaskTracker.Messaging.Builders;
using TaskTracker.Messaging.Entities;

namespace TaskTracker.Bll.Impl.Messaging.Templates.Base
{
    public abstract class MessageTemplate : IMessageTemplate
    {
        protected readonly IMailBuilder _builder;

        public MessageTemplate(IMailBuilder builder)
        {
            _builder = builder;
        }

        public abstract MessageTemplateType MessageType { get; }

        public abstract MailEntity GetMail(SystemMailEntity systemMail,
            object additionalTemplateData);

        protected abstract void SetAdditionalTemplateData
            (object additionalTemplateData);
    }
}
```

Його конкретні реалізації вирішують який сам об’єкт буде створюватись. (На відміну від звичної реалізації в даній не використовується поліморфізм для типу який повертається фабричним методом)

```
using System.Text;
using TaskTracker.Bll.Abstract.Messaging.Template;
using TaskTracker.Bll.Impl.Messaging.Templates.Base;
using TaskTracker.Common.Enums;
using TaskTracker.Messaging.Builders;
using TaskTracker.Messaging.Entities;

namespace TaskTracker.Bll.Impl.Messaging.Templates
{
    /// <summary>
    /// Template for confirm registration.
    /// </summary>
    public class RegistrationTemplate : MessageTemplate, IMessageTemplate
    {
        public RegistrationTemplate(IMailBuilder builder) : base(builder) { }

        public override MessageTemplateType MessageType
            => MessageTemplateType.RegistrationConfirm;

        /// <summary>
        /// Get message content.
        /// </summary>
        /// <param name="systemMail">System mail part.</param>
```

```
        /// <returns>MailEntity with all needed content for registration.</returns>
        public override MailEntity GetMail(SystemMailEntity systemMail,
            object additionalTemplateData = null)
        {
            StringBuilder htmlBuilder = new StringBuilder();

            var subject = "TaskTracker Registration.";

            var html = htmlBuilder.Append
                (@"<!DOCTYPE html>
                    <html>
                        <head>
                             <style> ")
                .Append(TemplateStyleHolder.ICloudSuportedStyle)
                .Append(@"</style>
                        </head>
                        <body>
                            <div class= 'message-header'>
                                <div class='title'>
                                    TaskTracker
                                </div>
                            </div>
                            <div class='message-content'>
                                <div>
                                    <h3>TaskTracker Team.</h3>
                                     <hr>")
                 .Append($"Hello {systemMail.ToName}.<br>")
                 .Append(@"Welcome to TaskTracker!<br>
                    Thanks for your registration.
                    Return to the site to update your information
                    and start work with your tasks!
                                </div>
                                <div class='base-button-container'>
                                    <a href='#...' class='base-button'>
                                        Go To TaskTracker!</a>
                                </div>
                            </div>
                            <div class='message-footer'>
                                <div class='text'>
                                    <h4>Contacts: </h4>
                                    Phone number: xxx-xxx-xxx<br>
                                    E-Mail: xxxx @mail.com
                            </div>
                        </div>
                    </body>
                </html>").ToString();

            return _builder
                .AddSystemPart(systemMail)
                .AddSubject(subject)
                .AddHtml(html)
                .Build();
        }

        protected override void SetAdditionalTemplateData
            (object additionalTemplateData)
        {
            //There are no aditioanle data for this template now.
        }
    }
}
```

ДОДАТОК Д

Приклад реалізації Facade для обмеження типів HTTP запитів

В нашому випадку фасадом буде ApiClient який абстрагується над бібліотекою RestSharp та дозволяє кінцевому користувачеві використовувати лише 4 типу Http запитів.

```
using Microsoft.Extensions.Options;
using RestSharp;
using System.Threading.Tasks;
using TaskTracker.Mobile.Common.Extenssions;
using TaskTracker.Mobile.Infrastructure.Authorization;

namespace TaskTracker.Mobile.Infrastructure.REST
{
    public class ApiCilent : IApiClient
    {
        private readonly IRestClient _restClient;

        public ApiCilent(IOptions<ApiConfiguration> apiConfig)
        {
            _restClient = new RestClient(apiConfig.Value.Url);
        }

        private IRestResponse TryExecute(IRestRequest request)
        {
            request.AddHeader("token", AuthToken.Get.Value);

            var response = _restClient.Execute(request.AddHeader("token",
AuthToken.Get.Value));

            return response.StatusCode != System.Net.HttpStatusCode.Unauthorized ? response
                : _restClient.Execute(request.AddHeader("token", AuthToken.Update.Value));
        }

        private async Task<IRestResponse> TryExecuteAsync(IRestRequest request)
        {
            request.AddHeader("token", AuthToken.Get.Value);

            var response = await _restClient.ExecuteTaskAsync(request.AddHeader("token",
AuthToken.Get.Value));

            return response.StatusCode != System.Net.HttpStatusCode.Unauthorized ? response
                : await _restClient.ExecuteTaskAsync(request.AddHeader("token",
AuthToken.Update.Value));
        }

        public void Delete(string endpoint) => TryExecute(new RestRequest(endpoint,
Method.DELETE));

        public Task DeleteAsync(string endpoint) => TryExecuteAsync(new
RestRequest(endpoint, Method.DELETE));

        public TResult Get<TResult>(string endpoint)
            => TryExecute(new RestRequest(endpoint,
Method.GET)).Content.CastAsJson<TResult>();

        public async Task<TResult> GetAsync<TResult>(string endpoint)
            => (await TryExecuteAsync(new RestRequest(endpoint,
Method.GET))).Content.CastAsJson<TResult>();
```

```
        public TResult Post<TData, TResult>(TData data, string endpoint)
            => TryExecute(new RestRequest(endpoint,
Method.POST).AddJsonBody(data)).Content.CastAsJson<TResult>();

        public async Task<TResult> PostAsync<TData, TResult>(TData data, string endpoint)
            => (await TryExecuteAsync(new RestRequest(endpoint,
Method.POST).AddJsonBody(data)))
            .Content.CastAsJson<TResult>();

        public void Put<TData>(TData data, string endpoint)
            => TryExecute(new RestRequest(endpoint, Method.PUT).AddJsonBody(data));

        public async Task PutAsync<TData>(TData data, string endpoint)
            => await TryExecuteAsync(new RestRequest(endpoint,
Method.PUT).AddJsonBody(data));
    }
```

ДОДАТОК Е

Приклад реалізації Interprener для SQL запитів

Бачимо, що інтерпритатор EntityFramework розбирає Expression та парсить їх в SQL запит.

```
using System;
using System.Collections.Generic;
using System.Linq.Expressions;
using TaskTracker.Dal.Abstract.Base;

namespace TaskTracker.Dal.Impl.Ef.Specifications
{
    public class Specification<T> : ISpecification<T>
    {
        public Expression<Func<T, bool>> Criteria { get; }
        public List<Expression<Func<T, object>>> Includes { get; }
        public List<string> IncludeStrings { get; }

        public Specification(Expression<Func<T, bool>> criteria)
        {
            Criteria = criteria;
            Includes = new List<Expression<Func<T, object>>>();
            IncludeStrings = new List<string>();
        }

        protected virtual void AddInclude(Expression<Func<T, object>> includeExpression)
        {
            Includes.Add(includeExpression);
        }

        protected virtual void AddInclude(string includeString)
        {
            IncludeStrings.Add(includeString);
        }

    }
}
```

ДОДАТОК Є

Приклад реалізації прийняття кредитного рішення

```
public readonly struct Decision
    {
        public readonly bool Positive;

        public readonly Percent Percent;

        public Decision(bool positive, Percent? percent = null)
        {
            Percent = percent ?? (positive ? Percent.Max : Percent.Min);
            Positive = positive;
        }

        public static Decision Yes(Percent? percent = null) => new(true, percent);

        public static Decision No(Percent? percent = null) => new(true, percent);
    }

    public interface IDecisionMaker
    {
        ValueTask<Decision> Decide(string userId, string insuranceId);
    }
public interface IInsuranceRepository
    {
        InsuranceProgram Get(string Id);
    }
public interface IInsurnaceCalculator
    {
        ValueTask<Percent> GetInsurancePossibility(string userId, string insuranceId);
    }
public interface IMapTo<T>
    {
        void Mapping(Profile profile) => profile.CreateMap(GetType(), typeof(T));
    }

    public interface IMapFrom<T>
    {
        void Mapping(Profile profile) => profile.CreateMap(typeof(T), GetType());
    }
public interface IUserRepository
    {
        Task<User> GetAsync(string userId);

        Task BulkInsertAsync(IEnumerable<User> userData);
    }
public class Insurance
    {
        public string Id { get; set; }

        public DateTime CreationDate { get; set; }

        public DateTime EndDate { get; set; }

        public User User { get; set; }

        public InsuranceProgram Program { get; set; }
    }
public enum InsuranceSubject
    {
        HealthCare,
```

```
        Life,
        Car,
        House
    }

    public enum InsuranceType
    {
        Standard,
        Extended
    }

    public record InsuranceProgram(string Id)
    {
        public InsuranceSubject Subject { get; init; }

        public InsuranceType Type { get; init; }

        public decimal Amount { get; set; }

        public decimal Price { get; set; }

        public string FullDescription { get; set; }
    }
public enum Gender { Male, Female }

    public readonly struct CreditScore
    {
        public enum CreditGroup { None, Poor, Average, Good, Excellent }

        public readonly int Value;

        public readonly CreditGroup Group;

        public CreditScore(int value)
        {
            Value = value;

            Group = value switch
            {
                >= 350 and < 630 => CreditGroup.Poor,
                >= 630 and < 690 => CreditGroup.Average,
                >= 690 and < 720 => CreditGroup.Good,
                >= 720 and < 850 => CreditGroup.Excellent,
                _ => CreditGroup.None
            };
        }
    }

    public record User(string Id)
    {
        public string Name { get; set; }

        public int Age { get; set; }

        public Gender Gender { get; set; }

        public bool Married { get; set; }

        public bool OwnHome { get; set; }

        public CreditScore CreditScore { get; set; }

        public decimal IncomePerYear { get; set; }
    }
public readonly struct Percent : IComparable<double>
```

```
    {
        public static Percent Max = new(100);

        public static Percent Min = new(0);

        public readonly double Value;

        public Percent(double value) => Value = value switch
        {
            <= 0 => 0,
            > 100 => 100,
            _ => value
        };

        public int CompareTo(double other) => Value.CompareTo(other);

        public override string ToString() => $"{Value}%";
    }
public class UserEntity : IMapFrom<User>
    {
        [Key]
        public string Id { get; set; }

        public string Name { get; set; }

        public int Age { get; set; }

        public int Gender { get; set; }

        public bool Married { get; set; }

        public bool OwnHome { get; set; }

        public int CreditScore { get; set; }

        public decimal IncomePerYear { get; set; }

        public void Mapping(Profile profile)
        {
            profile.CreateMap<User, UserEntity>()
                .ForMember(x => x.Gender, x => x.MapFrom(y => (int)y.Gender))
                .ForMember(x => x.CreditScore, x => x.MapFrom(y => y.CreditScore.Value))
                .ReverseMap()
                .ForMember(x => x.Gender, x => x.MapFrom(y => (Gender)y.Gender))
                .ForMember(x => x.CreditScore, x => x.MapFrom(y => new
CreditScore(y.CreditScore)));
        }
    }
public abstract class Repository
    {
        protected readonly InsuranceCalculatorContext Context;
        protected readonly IMapper Mapper;

        public Repository(InsuranceCalculatorContext db, IMapper mapper)
        {
            Context = db;
            Mapper = mapper;
        }

        protected Task<TEntity> Get<TEntity>(Expression<Func<TEntity, bool>> predicate)
            where TEntity : class
            => Context.Set<TEntity>().FirstOrDefaultAsync(predicate);

        protected T Map<T>(object value) => Mapper.Map<T>(value);
    }
```

```
    public class UserRepository : Repository, IUserRepository
    {
        public UserRepository(InsuranceCalculatorContext db, IMapper mapper)
            : base(db, mapper)
        {

        }

        public Task BulkInsertAsync(IEnumerable<User> userData)
        {
            throw new NotImplementedException();
        }

        public async Task<User> GetAsync(string userId)
        {
            var userEntity = await Get<UserEntity>(u => u.Id == userId);
            return Map<User>(userEntity);
        }
    }
public class DecisionMaker : IDecisionMaker
    {
        private readonly IInsurnaceCalculator _calculator;

        public DecisionMaker(IInsurnaceCalculator calculator)
        {
            _calculator = calculator;
        }

        public async ValueTask<Decision> Decide(string userId, string insuranceId)
        {
            return await _calculator.GetInsurancePossibility(userId, insuranceId) switch
            {
                { } percent when percent.Value > 50 => Decision.Yes(percent),
                { } percent => Decision.No(percent)
            };
        }
    }
```

ДОДАТОК Ж

Алгоритм роботи підсистеми прийняття рішення