МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМ. В. ДАЛЯ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ЕЛЕКТРОНІКИ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК ТА ІНЖЕНЕРІЇ

До захисту допускається
Завідувач кафедри
_____ Скарга-Бандурова І.С.
підпис
"____"_____ 2018 р.

ДИПЛОМНИЙ ПРОЕКТ (РОБОТА) БАКАЛАВРА

ПОЯСНЮВАЛЬНА ЗАПИСКА

НА ТЕМУ:

Система пошуку нерухомості на основі інтелектуального аналізу даних

Освітньо-кваліфікаційний рівень "бакалавр"
Напрям підготовки 6.050102 – "Комп'ютерна інженерія"

Керівник проекту:	_____ (підпис)	Лифар О.К. (ініціали, прізвище)
Консультант з охорони праці:	_____ (підпис)	Критська Я.О. (ініціали, прізвище)
Студент:	_____ (підпис)	Черкасов О.О. (ініціали, прізвище)
Група:		КН-14д

Сєвєродонецьк 2018

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Факультет Інформаційних технологій та електроніки
Кафедра Комп'ютерних наук та інженерії
Освітньо-кваліфікаційний рівень Бакалавр
Напрям підготовки 6.050102 – "Комп'ютерна інженерія"
(шифр і назва)
Спеціальність
(шифр і назва)

ЗАТВЕРДЖУЮ:

Завідувач кафедри
_____ І.С. Скарга-Бандурова
"_____" _____2018 р.

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) БАКАЛАВРА

Черкасов Олександр Олександрович
(прізвище, ім'я, по батькові)

1. Тема роботи Система пошуку нерухомості на основі інтелектуального аналізу даних

керівник проекту (роботи) Лифар О.К., ст. викладач
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від "14" 05 2018 р. № 117/48

2. Термін подання студентом роботи 17.06.2018

3. Вихідні дані до роботи матеріали переддипломної практики

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз предметної області та постановка задачі.
2. Вибір засобів для розробки.
3. Реалізація система пошуку нерухомості на основі інтелектуального аналізу даних.
4. Охорона праці та безпека в надзвичайних ситуаціях.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Електронні плакати

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці та безпека в надзвичайних ситуаціях	ст. викл. Критська Я.О.		

7. Дата видачі завдання 14.05.2018

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Ознайомлення з предметною галуззю	03.05 – 15.05	
2	Аналіз існуючих аналогів	16.05 – 18.05	
3	Вибір засобів для розробки	19.05 – 25.05	
4	Розробка основних модулів підсистеми	26.05 – 03.06	
5	Розробка розділу "Охорона праці та безпека в надзвичайних ситуаціях"	04.06 – 08.06	
6	Оформлення пояснювальної записки	09.06 – 15.06	

Студент _____ Черкасов О.О.
(підпис) (прізвище та ініціали)

Керівник _____ Лифар О.К.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломного проекту містить: 79 сторінок; 17 рисунків; 24 джерела; 7 таблиць; 17 формул.

Об’єкт: Система пошуку нерухомості на основі інтелектуального аналізу даних.

Мета роботи: створення додатку "Система пошуку нерухомості на основі інтелектуального аналізу даних".

В проекті виконано:

1. У розділі "Система пошуку нерухомості на основі інтелектуального аналізу даних на базі технологій parser" було виконано зрівняння алгоритмів, аналіз алгоритмів, були поставлені задачі щодо розробки системи.
2. У розділі "Обґрунтування інструментів для розробки системи пошуку нерухомості на основі інтелектуального аналізу даних" були розглянуті інструменти, за допомогою яких буде розроблятися система.
3. У розділі "Проектування системи пошуку нерухомості на основі parser та кластеризації" описано проектування системи.
4. У розділі "Охорона праці" був проведений аналіз шкідливих виробничих факторів. На основі цього аналізу запропоновані заходи усунення цих факторів.

Отримано наступні результати: розроблена система Parser на основі кластеризації.

АЛГОРИТМИ, PARSER, .NET FRAMEWORK, СИСТЕМА, МОДЕЛІ, C#, HTML AGILITY PACK, VISUAL STUDIO

Умови одержання дипломного проекту: СНУ ім. В. Даля, пр. Центральний 59-А, м. Сєвєродонецьк, 93400.

ЗМІСТ

ВСТУП

Parser – об'єктно-орієнтована скриптова мова програмування, створена для генерації HTML-сторінок на веб-сервері з підтримкою CGI. Розроблена Студією Артемія Лебедєва і випущена під ліцензією, подібна до GNU GPL. Мова спеціально спроектована і оптимізована для того, щоб було зручно створювати прості сайти. Робота з формами, cookies, табличними файлами, базами даних і XML – частина мови, а модульність мови дозволяє легко нарощувати функціональність.

Parser в даному разі – макромова, в якій не має оператора print. Весь текст, набраний у вихідному файлі, по суті, великий оператор print, а конструкції Parser є зануреними в текст. Виходить, що не створюється програма, яка виводить текст – навпаки, в наявний текст додається логіка і організація, блоки (методи), на які розбивається HTML-код.

У кожну директорію, з якої буде працювати Parser, можна класти файл auto.p, в якому будуть описані основні налаштування та методи. Особливістю є спадковість (наявність в декількох папках по шляху до скрипту) і безумовне підключення цього файлу. Таким чином, висновок меню можна описати лише в одному файлі і він автоматично буде застосований до всього сайту.

Деякі обмеження (наприклад, робота з зображеннями) легко усуваються використанням сторонніх консольних утиліт і shell-скриптів.

Тема дипломної роботи – "Система пошуку нерухомості на основі інтелектуального аналізу даних".

Метою дипломної роботи є створення додатку "Система пошуку нерухомості на основі інтелектуального аналізу даних". Досягнення поставленої мети дозволить прискорити пошук потрібної інформації.

Досягнення мети дипломної роботи було зведено до вирішення наступних завдань.

Створення додатку в якому містять:

– район (центр, околиця);

– загальна площа і корисна площа в $м^2$;

– призначення (магазин, офіс, склад, різне);

– наявність додаткових послуг – водопостачання, каналізації, додаткових телефонних ліній та ін. (є немає);

– купівля нерухомості:

а) кількість поверхів у будинку;

б) поверх, на якому знаходиться об'єкт (підвал, перший поверх, другий поверх і вище).

У зв'язку з кризою, яка останнім часом охопила всі сфери життя, для покупця нерухомості дуже актуально зберегти баланс: ціна-якість. Тому на сьогоднішній день на ринку нерухомість стала ще більш актуальною темою.

Ступінь новизни. Інформаційні системи "Система пошуку нерухомості" реалізовані в багатьох випадках. Реалізації розрізняються відповідно до прийнятих стилів роботи.

Зацікавлені сторони. "Система пошуку нерухомості" може представляти інтерес для:

– орендодавців;

– економістів;

– аналітиків;

– забудовників;

– скупників землі;

– агентств нерухомості;

– дослідних установ.

1 СИСТЕМА ПОШУКУ НЕРУХОМОСТІ НА ОСНОВІ ІНТЕЛЕКТУАЛЬНОГО АНАЛІЗУ ДАНИХ НА БАЗІ ТЕХНОЛОГІЙ PARSER

1.1 Методи добування інформації

Можна виділити наступні методи добування інформації з сайтів:

1. Ручний – в даному випадку роль парсеру виконує людина. Вона виробляє весь ланцюжок дій, необхідний для отримання необхідної інформації. Інформація збирається звичайним копіпастом.

2. Гібридний – користувач як і раніше виконує основні дії для отримання інформації, але може використовувати допоміжні програмні засоби для автоматизації збору, наприклад, браузерний плагін, який на основі конфігурації витягує і структурує інформацію з певних місць сторінки при активації.

3. Автоматичний – отримання і структурування інформації виконується автоматично.

Для того щоб один раз отримати значення декількох полів з 10-15 сторінок, писати окремий парсер може бути і недоцільно, проте в разі великого числа сторінок, полів або високої періодичності збору інформації, варто задуматися про автоматизацію цього процесу.

Процес вилучення інформації з окремої веб-сторінки можна розбити на наступні етапи:

- побудова запиту для отримання інформації;
- виконання запиту і отримання відповіді;
- обробка відповіді, вилучення та структурування необхідної інформації;
- передача отриманої інформації для подальшої обробки.

Процес вилучення інформації може бути простим: завантажити URL, зчитати інформацію і віддати одержувачу; а може бути і складним:

авторизуватися в системі, сконструювати запит за інформацією з заголовків і значень JavaScript-змінних на сторінці, ім'я яких може змінюватися з кожним запитом, а JS-код перебувати в мініфіцірованому або обфусцірованому вигляді.

Якщо в першому випадку все досить просто, то в другому, щоб за розумний час розробити парсер, варто вдатися до використання headless-браузерів (без графічного інтерфейсу) з підтримкою сценаріїв на кшталт PhantomJS для вилучення даних для оптимізації часу на вивчення того, як сайт взаємодіє з бекенд. Також для цих цілей можна вдатися до Selenium WebDriver з одним з реальних браузерів[10].

1.1.1 Інструменти

Для автоматизованого вилучення інформації з веб-сторінок існує 3 типи інструментів:

1. Бібліотеки.

Цей підхід вимагає розуміння процесу формування запитів і логіки роботи програми, що тягне за собою додаткові витрати на вивчення низькорівневої роботи сайту. Можливо, це доречно і виправдано для одиничного сайту, але в разі якщо потрібно написати кілька парсерів для декількох різнорідних сайтів за обмежений час – навряд чи.

До таких інструментів належать численні бібліотеки для різних мов програмування: jSoup для Java, SimpleHTMLDom для PHP, lxml.html для Python і інші.

2. Headless-браузери.

Даний підхід дозволяє обробляти сторінку в браузері з підтримкою JavaScript, що дозволяє писати свої сценарії для отримання необхідної інформації і навіть використовувати JavaScript бібліотеки на зразок jQuery для отримання інформації зі сторінки, що прискорює розробку парсерів. Відсутність графічного

інтерфейсу дозволяє запускати дані браузери навіть на серверах, що підтримують тільки консольний режим.

До таких інструментів можна віднести PhantomJS і SlimerJS.3. SaaS рішення.

Дані сервіси надають графічний інтерфейс, за допомогою якого можна вказати адресу сторінки, вказати блоки, з яких потрібно витягти інформацію, а також створити ряд правил по вилученню даних. Такі сервіси не мають тієї гнучкості, яку надають низькорівневі рішення. Деякі з них коштують досить дорого, зате ними просто користуватися[10].

1.2 Аналіз алгоритмів інтелектуального аналізу даних

В інтелектуальному аналізі даних (або машинному навчанні) алгоритм – це набір евристики і обчислень, який створює на основі даних модель. Щоб створити модель, алгоритм спочатку аналізує надані дані, здійснюючи пошук певних закономірностей і тенденцій. Алгоритм застосовує результати цього аналізу до множини ітерацій, щоб підібрати оптимальні параметри для створення моделі інтелектуального аналізу даних. Потім ці параметри застосовуються до всього набору даних, щоб виявити придатні до використання закономірності і отримати докладну статистику.

Модель інтелектуального аналізу даних, створювана алгоритмом з наданих даних, може мати різні форми, включаючи наступні:

– набір кластерів, що описують зв'язок варіантів в наборі даних;

– дерево рішень, яке передбачає результат і описує, який вплив на цей результат надають різні критерії;

– математичну модель, що прогнозує продажу;

– набір правил, що описують групування продуктів в транзакції, а також ймовірності одночасної покупки продуктів.

В інтелектуальному аналізі даних SQL Server використовуються найбільш

популярні і вивчені методи виявлення закономірностей в даних. Наприклад, алгоритм кластеризації методом К-середніх є одним з найстаріших алгоритмів кластеризації і широко застосовується в багатьох інструментах і з багатьма реалізаціями і параметрами. При цьому алгоритм кластеризації методом k-середніх, реалізований в інтелектуальному аналізі даних SQL Server, був розроблений групою Microsoft Research, а потім був оптимізований для роботи зі службами Служби Analysis Services. Всі алгоритми інтелектуального аналізу даних Майкрософт доступні для гнучкого налаштування і програмування з використанням наданих API. За допомогою компонентів інтелектуального аналізу даних в службах Служби Integration Services можливо також автоматизувати створення, навчання і перенавчання моделей.

Крім того, підтримується використання сторонніх алгоритмів, відповідних специфікації OLE DB для інтелектуального аналізу даних. Є також можливість розробляти власні алгоритми, які можна зареєструвати в якості служб, а потім використовувати в платформі інтелектуального аналізу даних SQL Server [1].

1.3 Огляд алгоритмів

1. Алгоритми ієрархічної кластеризації.

Серед алгоритмів ієрархічної кластеризації виділяються два основних типи: висхідні і низхідні алгоритми. Спадні алгоритми працюють за принципом "зверху-вниз": на початку всі об'єкти поміщаються в один кластер, який потім розбивається на все більш дрібні кластери. Більш поширені висхідні алгоритми, які на початку роботи поміщають кожен об'єкт в окремий кластер, а потім об'єднують кластери у все більші, поки всі об'єкти вибірки не будуть міститися в одному кластері. Таким чином будується система вкладених розбиттів. Результати таких алгоритмів зазвичай представляють у вигляді дерева – дендрограми. Класичний приклад такого дерева – класифікація тварин і рослин.

Для обчислення відстаней між кластерами частіше за все користуються двома відстанями: з одиночним зв'язком або з повним зв'язком.

До недоліку ієрархічних алгоритмів можна віднести систему повного розбиття, яка може бути зайвою в контексті розв'язуваної задачі.

2. Алгоритми квадратичної помилки.

Завдання кластеризації можна розглядати як побудову оптимального розбиття об'єктів на групи. При цьому оптимальність може бути визначена як вимога мінімізації середньоквадратичної помилки розбиття:

$$e^2(X,L)=\sum_{j=1}^{K}\sum_{i=1}^{n_j}\left\|x_i^{(j)}-c_j\right\|^2, \qquad (1.1)$$

де c_j – "центр мас" кластера j (точка з середніми значеннями характеристик для даного кластера).

Алгоритми квадратичної помилки відносяться до типу плоских алгоритмів. Найпоширенішим алгоритмом цієї категорії є метод k-середніх. Цей алгоритм будує задане число кластерів, розташованих якнайдалі один від одного. Робота алгоритму ділиться на кілька етапів:

- випадково вибрати k точок, які є початковими "центрами мас" кластерів;
- віднести кожен об'єкт до кластеру з найближчим "центром мас";
- перерахувати "центри мас" кластерів відповідно до їх поточного складу;
- якщо критерій зупинки алгоритму не задоволений, повернутися до п. 2.

Як критерій зупинки роботи алгоритму зазвичай вибирають мінімальну зміну середньоквадратичної помилки. Так само можливо зупиняти роботу алгоритму, якщо на кроці 2 не було об'єктів, що перемістилися з кластера в кластер.

До недоліків даного алгоритму можна віднести необхідність задавати кількість кластерів для розбиття.

3. Нечіткі алгоритми.

Найбільш популярним алгоритмом нечіткої кластеризації є алгоритм c-середніх (c-means). Він являє собою модифікацію методу k-середніх. Кроки роботи алгоритму наступні:

– вибрати початкове нечітке розбиття n об'єктів на k кластерів шляхом вибору матриці приналежності u розміру n x k;

– використовуючи матрицю u, знайти значення критерію нечіткої помилки:

$$E^2(X,U)=\sum_{i=1}^{N}\sum_{k=1}^{K}U_{ik}\left\|x_i^{(k)}-c_k\right\|^2, \tag{1.2}$$

де c_k – "центр мас" нечіткого кластера k:

$$c_k=\sum_{i=1}^{N}U_k x_i\ ; \tag{1.3}$$

– перегрупувати об'єкти з метою зменшення цього значення критерію нечіткої помилки;

– повертатися в п. 2 до тих пір, поки зміни матриці u не стануть незначними.

Цей алгоритм може не підійти, якщо заздалегідь невідомо число кластерів, або необхідно однозначно віднести кожен об'єкт до одного кластеру.

4. Алгоритми, засновані на теорії графів.

Суть таких алгоритмів полягає в тому, що вибірка об'єктів представляється у вигляді графа *G = (V, E)*, вершинам якого відповідають об'єкти, а ребра мають вагу, що дорівнює "відстані" між об'єктами. Перевагою графових алгоритмів кластеризації є наочність, відносна простота реалізації і можливість внесення різних удосконалень, заснованих на геометричних міркуваннях. Основними алгоритмами є алгоритм виділення зв'язкових компонент, алгоритм побудови мінімального кістякового (остовного) дерева і алгоритм пошарової кластеризації.

5. Алгоритм виділення зв'язкових компонент.

В алгоритмі виділення зв'язкових компонент задається вхідний параметр R і в графі видаляються всі ребра, для яких "відстані" більше R. Сполученими залишаються тільки найбільш близькі пари об'єктів. Сенс алгоритму полягає в тому, щоб підібрати таке значення R, що лежить в діапазоні всіх "відстаней", при якому граф "розвалиться" на кілька зв'язкових компонент. Отримані компоненти і є кластери.

Для підбору параметра R зазвичай будується гістограма розподілів попарних відстаней. У завданнях з добре вираженою кластерною структурою даних на гістограмі буде два піки – один відповідає внутрікластерним відстаням, другий – міжкластерним відстаням. Параметр R підбирається із зони мінімуму між цими піками. При цьому управляти кількістю кластерів за допомогою порогу відстані досить важко.

6. Алгоритм мінімального кістякового дерева.

Алгоритм мінімального кістякового дерева спочатку будує на графі мінімального покриття дерева, а потім послідовно видаляє ребра з найбільшою вагою.

7. Пошарова кластеризація.

Алгоритм пошарової кластеризації заснований на виділенні зв'язкових компонент графа на деякому рівні відстаней між об'єктами (вершинами). Рівень відстані задається порогом відстані c. Наприклад, якщо відстань між об'єктами

$$0 \le \rho(x, x') \le 1, \text{ тоді } 0 \le c \le 1. \tag{1.4}$$

Алгоритм пошарової кластеризації формує послідовність підграфів графа G, які відображають ієрархічні зв'язки між кластерами:

$$G^0 \subseteq G^1 \subseteq \ldots \subseteq G^m, \tag{1.5}$$

де $G^t = (V, E^t)$ – граф на рівні c^t;

$$E^t = \{e_{ij} \in E : \rho_{ij} \le c_t\}; \quad (1.6)$$

c^t – t-ий поріг відстані;

m – кількість рівнів ієрархії;

$G^0 = (V, o)$, де o – порожня множина ребер графа, що отримується при $t^0 = 1$;

$G^m = G$, тобто граф об'єктів без обмежень на відстань (довжину ребер графа), оскільки $t^m = 1$.

За допомогою зміни порогів відстані $\{c^0, ..., c^m\}$, де $0 = c^0 < c^1 < ... < c^m = 1$, можливо контролювати глибину ієрархії одержуваних кластерів. Таким чином, алгоритм пошарової кластеризації здатний створювати як плоске розбиття даних, так і ієрархічне [8].

1.4 Порівняння алгоритмів

Порівняння алгоритмів кластеризації наведено в табл. 1.1, 1.2.

Таблиця 1.1 – Обчислювальна складність алгоритмів

Алгоритм кластеризації	**Обчислювальна складність**
Ієрархічний	$O(n^2)$
k-середніх	O(nkl), де k – число кластерів, l – число ітерацій
c-середніх	
Виділення зв'язкових компонент	Залежить від алгоритму
Мінімальне кістякове дерево	$O(n^2 \log n)$
Пошарова кластеризація	O(max(n, m)), де m < n(n-1)/2

Таблиця 1.2 – Порівняльна таблиця алгоритмів

Алгоритм кластеризації	**Форма кластерів**	**Вхідні дані**	**Результати**
Ієрархічний	Довільна	Число кластерів або поріг відстані для усічення ієрархії	Бінарне дерево кластерів
k-середніх	Гіперсфера	Число кластерів	Центри кластерів
c-середніх	Гіперсфера	Число кластерів, ступінь нечіткості	Центри кластерів, матриця приналежності

Продовження табл. 1.2

Виділення зв'язкових компонент	Довільна	Поріг відстані R	Деревовидна структура кластерів
Мінімальне кістякове дерево	Довільна	Число кластерів або поріг відстані для видалення ребер	Деревовидна структура кластерів
Пошарова кластеризація	Довільна	Послідовність порогів відстані	Деревовидна структура кластерів з різними рівнями ієрархії

1.4.1 Вибір правильного алгоритму

Вибір правильного алгоритму для використання в конкретній аналітичній задачі може бути досить складним. У той час як можна використовувати різні алгоритми для виконання однієї і тієї ж задачі, кожен алгоритм видає різний результат, а деякі алгоритми можуть видавати більше одного типу результатів. Наприклад, можна використовувати алгоритм дерева прийняття рішень Microsoft не тільки для прогнозування, але також в якості способу зменшення кількості стовпців в наборі даних, оскільки дерево прийняття рішень може ідентифікувати стовпці, які не впливають на кінцеву модель інтелектуального аналізу даних.

1.4.2 Вибір алгоритму по типу

SQL Server

Інтелектуальний аналіз даних включає такі типи алгоритмів:

– алгоритми класифікації здійснюють прогнозування однієї або декількох дискретних змінних на основі інших атрибутів в наборі даних;

– регресивні алгоритми здійснюють прогнозування однієї або декількох безперервних числових змінних, наприклад прибутку або збитків, на основі інших атрибутів в наборі даних;

– алгоритми сегментації ділять дані на групи або кластери елементів, що мають схожі властивості;

– алгоритми взаємозв'язків здійснюють пошук кореляції між різними атрибутами в наборі даних. Найбільш частим застосуванням цього типу алгоритму є створення правил взаємозв'язку, які можуть використовуватися для аналізу споживчого кошика;

– алгоритми аналізу послідовностей узагальнюють послідовності, які часто зустрічаються в даних, такі як серія переходів по веб-сайту або подій, зареєстрованих в журналі перед ремонтом обладнання.

Однак ніщо не змушує користувача обмежуватися одним алгоритмом в своїх рішеннях. Досвідчені аналітики часто використовують один алгоритм для виявлення найбільш ефективних вхідних даних (тобто змінних), після чого застосовують інший алгоритм для прогнозування певного результату на основі цих даних. SQL Server Інтелектуальний аналіз даних дозволяє на базі однієї структури інтелектуального аналізу побудувати багато моделей таким чином, що в рамках одного рішення для інтелектуального аналізу даних можна було використовувати алгоритм кластеризації, модель дерева рішень, а також модель спрощеного алгоритму Байеса для отримання різних представлень даних. В одному рішенні також можна використовувати кілька алгоритмів для виконання окремих завдань. Наприклад, за допомогою регресії можна отримувати фінансові прогнози, а за допомогою алгоритму нейронної мережі виконувати аналіз чинників, що впливають на прогнози [9].

1.4.3 Вибір алгоритму по завданню

Щоб полегшити вибір алгоритмів для вирішення певної задачі, в таблиці 1.3 наведено типи завдань, для вирішення яких зазвичай використовується кожен алгоритм.

Таблиця 1.3 – Вибір алгоритму по завданню

Приклади завдань	Відповідні алгоритми Майкрософт
Прогнозування дискретного атрибуту: Позначка клієнтів зі списку потенційних покупців як хороших і поганих кандидатів. Обчислення ймовірності відмови сервера протягом наступних шести місяців. Класифікація варіантів розвитку хвороб пацієнтів і дослідження пов'язаних факторів**.**	Алгоритм дерева прийняття рішень (Майкрософт) Алгоритм Байеса (Майкрософт) Алгоритм кластеризації (Майкрософт) Алгоритм нейронної мережі (Майкрософт)
Прогнозування безперервного атрибуту: Прогноз продажів на наступний рік. Прогноз кількості відвідувачів сайту з урахуванням минулих років і сезонних тенденцій. Формування оцінки ризику з урахуванням демографії.	Алгоритм дерева прийняття рішень (Майкрософт) Алгоритм часових рядів (Майкрософт) Алгоритм лінійної регресії (Майкрософт)
Прогнозування послідовності: Аналіз маршруту переміщення по веб-сайту компанії. Аналіз факторів, що ведуть до відмови сервера. Відстеження і аналіз послідовностей дій під час відвідування поліклініки з метою формулювання рекомендацій по спільних діях**.**	Алгоритм кластеризації послідовностей (Майкрософт)
Знаходження груп загальних елементів в транзакціях: Використання аналізу споживчого кошика для визначення місць розміщення продуктів. Виявлення додаткових продуктів, які можна запропонувати купити клієнту. Аналіз даних опитування, проведеного серед відвідувачів події, з метою виявлення того, які дії і стенди були пов'язані, щоб планувати майбутні дії.	Алгоритм взаємозв'язків (Майкрософт) Алгоритм дерева прийняття рішень (Майкрософт)
Знаходження груп схожих елементів: Створення профілів ризиків для пацієнтів на основі таких атрибутів, як демографія і поведінка. Аналіз користувачів по шаблонах перегляду і покупки. Визначення серверів, які мають аналогічні характеристики використання**.**	Алгоритм кластеризації (Майкрософт) Алгоритм кластеризації послідовностей (Майкрософт)

Виходячи з представлених в таблиці 1.3 алгоритмів і їх можливостей було вибрано алгоритм кластеризації як більш вподобаний з усіх в представленій дипломній роботі. Кластеризація послідовностей Microsoft – це унікальний алгоритм, який поєднує в собі аналіз послідовностей і кластеризації. Даний алгоритм можна використовувати для перегляду даних, що містять події, які можуть бути пов'язані в послідовність. Алгоритм знаходить найпоширеніші послідовності і виконує кластеризацію для пошуку ідентичних послідовностей.

1.5 Технічне завдання на розробку додатка

ТЕХНІЧНЕ ЗАВДАННЯ

При розробці додатка Система пошуку нерухомості на основі інтелектуального аналізу даних з використанням мов C#, та технології Parser потрібно вирішити наступні завдання:

– Проаналізувати предметну область та існуючі програмні засоби, які її описують;

– Спроектування системи пошуку нерухомості на основі інтелектуального аналізу даних з використанням мов C#, та технології Parser;

– Розробки системи пошуку нерухомості на основі інтелектуального аналізу даних з використанням мов C#, та технології Parser.

Метою дипломної роботи є створення додатку "Система пошуку нерухомості на основі інтелектуального аналізу даних". Досягнення поставленої мети дозволить прискорити пошук потрібної інформації.

Досягнення мети дипломної роботи було зведено до вирішення наступних завдань.

Створення додатку в якому містять:

– район (центр, околиця);

– загальна площа і корисна площа в $м^2$;

– призначення (магазин, офіс, склад, різне);

– наявність додаткових послуг – водопостачання, каналізації, додаткових телефонних ліній та ін. (є немає);

– купівля нерухомості:

а) кількість поверхів у будинку;

б) поверх, на якому знаходиться об'єкт (підвал, перший поверх, другий поверх і вище).

У зв'язку з кризою, яка останнім часом охопила всі сфери життя, для

покупця нерухомості дуже актуально зберегти баланс: ціна-якість. Тому на сьогоднішній день на ринку нерухомість стала ще більш актуальною темою.

Ступінь новизни. Інформаційні системи "Система пошуку нерухомості" реалізовані в багатьох випадках. Реалізації розрізняються відповідно до прийнятих стилів роботи.

Зацікавлені сторони. "Система пошуку нерухомості" може представляти інтерес для:

- орендодавців;
- економістів;
- аналітиків;
- забудовників;
- скупників землі;
- агентств нерухомості;
- дослідних установ.

Висновки до першого розділу

Після проведення огляду предметної області було виконано порівняльний аналіз аналогічних алгоритмів. Огляд алгоритмів по типу, та вибір алгоритму по завданню. Методи добування інформації та інструменти.

Для вирішення задачі потрібно проаналізувати предметну область та існуючі програмні засоби, які її описують, спроектувати та розробити додаток з використанням мови C#.

Для вирішення задачі потрібно проаналізувати предметну область та існуючі програмні засоби, які її описують, спроектувати та розробити систему пошуку нерухомості на основі інтелектуального аналізу даних з використанням мов C#, Parser.

Були сформульовані основні задачі, описані методи та засоби, які були обрані для реалізації додатку. Було складено технічне завдання на розробку, з урахуванням недоліків розглянутих аналогічних алгоритмів.

2 ОБҐРУНТУВАННЯ ІНСТРУМЕНТІВ ДЛЯ РОЗРОБКИ СИСТЕМИ ПОШУКУ НЕРУХОМОСТІ НА ОСНОВІ ІНТЕЛЕКТУАЛЬНОГО АНАЛІЗУ ДАНИХ

2.1 Середовище розробки

C# є основною мовою розробки програм на платформі .NET корпорації Microsoft. У ній вдало поєднуються випробувані засоби програмування з самими останніми нововведеннями і надається можливість для ефективного і дуже практичного написання програм, призначених для обчислювального середовища сучасних підприємств. Це одна з найважливіших мов програмування XXI століття. C# і .NET Framework спільно утворюють вельми витончене середовище програмування.

На сьогоднішній момент мова програмування C# одна з найпотужніших, що швидко розвивається і затребувана в IT-галузі. На ній пишуться найрізноманітніші програми: від невеликих десктопних програмок до великих веб-порталів і веб-сервісів, які обслуговують щодня мільйони користувачів.

У порівнянні з іншими мовами C# досить молода, але в той же час вона вже пройшла великий шлях. Перша версія мови вийшла разом з релізом Microsoft Visual Studio .NET в лютому 2002 року. Поточною версією мови є версія C# 7.0, яка вийшла 7 березня 2017 року разом з Visual Studio 2017.

C# є мовою з Сі-подібним синтаксисом і близька в цьому відношенні до C++ і Java.

C# є об'єктно-орієнтованою і в цьому плані багато перейняла у Java і C++. Наприклад, C# підтримує поліморфізм, успадкування, перевантаження операторів, статичну типізацію. Об'єктно-орієнтований підхід дозволяє вирішити завдання з побудови великих, але в той же час гнучких, масштабованих і розширюваних додатків. C # продовжує активно розвиватися, і з кожною новою

версією з'являється все більше цікавих функціональностей, як, наприклад, лямбда, динамічне зв'язування, асинхронні методи і т.ін.

2.2 Зв'язок C# з середовищем .NET Framework

Незважаючи на те що C# є самодостатньою мовою програмування, у неї є особливий взаємозв'язок із середовищем виконання .NET Framework. Наявність такого взаємозв'язку пояснюється двома причинами. По-перше, C# спочатку призначалася для створення коду, який повинен виконуватися в середовищі .NET Framework. По-друге, використовувані в C# бібліотеки визначені в середовищі .NET Framework. На практиці це означає, що C# і .NET Framework тісно пов'язані один з одним, хоча теоретично C# можна відокремити від середовища .NET Framework.

Призначення .NET Framework – служити середовищем для підтримки розробки та виконання сильно розподілених компонентних додатків. Воно забезпечує спільне використання різних мов програмування, а також безпеку, перенесення програм і загальну модель програмування для платформи Windows. Що ж стосується взаємозв'язку з C#, то середовище .NET Framework визначає два дуже важливі елементи. Першим з них є загальномовне середовище виконання (Common Language Runtime – CLR). Це система, що управляє виконанням програм. Серед інших переваг – CLR як складова частина середовища .NET Framework підтримує багатомовне програмування, а також забезпечує перенесення та безпечне виконання програм.

Другим елементом середовища .NET Framework є бібліотека класів. Ця бібліотека надає програмі доступ до середовища виконання. Так, якщо потрібно виконати операцію вводу-виводу, наприклад, вивести що-небудь на екран, то для цієї мети використовується бібліотека класів .NET. Клас – це об'єктно орієнтована конструкція, яка допомагає організувати програми. Якщо програма обмежується

засобами, обумовленими в бібліотеці класів .NET, то така програма може виконуватися всюди, де підтримується середовище виконання .NET. А оскільки в C# бібліотека класів .NET використовується автоматично, то програми на C# свідомо виявляються такими, що переносяться в усі наявні середовища .NET Framework.

2.3 Роль платформи .NET

Коли говорять C#, нерідко мають на увазі технології платформи .NET (WPF, ASP.NET). І, навпаки, коли говорять .NET, нерідко мають на увазі C#. Однак, хоча ці поняття пов'язані, ототожнювати їх невірно. Мова C# була створена спеціально для роботи з фреймворком .NET, проте саме поняття .NET дещо ширше.

Якось Білл Гейтс сказав, що платформа .NET – це найкраще, що створила компанія Microsoft. Можливо, він мав рацію. Фреймворк .NET представляє потужну платформу для створення додатків. Можна виділити наступні її основні риси:

1. Підтримка декількох мов. Основою платформи є загальномовне середовище виконання Common Language Runtime (CLR), завдяки чому .NET підтримує кілька мов: поряд з C# це також VB.NET, C++, F#, а також різні діалекти інших мов, прив'язані до .NET, наприклад, Delphi.NET. При компіляції код на будь-якій з цих мов компілюється в збірку спільною мовою CIL (Common Intermediate Language) – свого роду асемблер платформи .NET. Тому ми можемо зробити окремі модулі однієї програми на окремих мовах.

2. Кросплатформеність. .NET є платформою, яку переносять (з деякими обмеженнями). Наприклад, остання версія платформи на даний момент .NET Framework підтримується на більшості сучасних ОС Windows (Windows 10/8.1/8/7/Vista). А завдяки проекту Mono можна створювати додатки, які будуть працювати і на інших ОС сімейства Linux, в тому числі на мобільних платформах Android і iOS.

3. Потужна бібліотека класів .NET представляє єдину для всіх підтримуваних мов бібліотеку класів. І який би додаток ми не збиралися писати на C# – текстовий редактор, чат або складний веб-сайт – так чи інакше ми задіємо бібліотеку класів .NET.

4. Різноманітність технологій. Загальномовне середовище виконання CLR і базова бібліотека класів є основою для цілого стека технологій, які розробники можуть задіяти при побудові тих чи інших додатків. Наприклад, для роботи з базами даних в цьому стеку технологій призначена технологія ADO.NET. Для побудови графічних додатків з багатим насиченим інтерфейсом – технологія WPF. Для створення веб-сайтів – ASP.NET і т.ін.

Також ще слід відзначити таку особливість мови C# і фреймворка .NET, як автоматичне прибирання сміття. А це означає, що нам в більшості випадків не доведеться, на відміну від C++, піклуватися про звільнення пам'яті. Вищезазначене загальномовне середовище CLR саме викличе збирач сміття і очистить пам'ять.

2.4 Принцип дії CLR

Середовище CLR управляє виконанням коду .NET. Діє воно за наступним принципом. Результатом компіляції програми на C# є не виконуваний код, а файл, що містить особливого роду псевдокод, званий Microsoft Intermediate Language, MSIL (проміжна мова Microsoft). Псевдокод MSIL визначає набір інструкцій, що переносяться і не залежать від конкретного процесора. По суті, MSIL визначає переносну мову асемблера. Слід, однак, мати на увазі, що, незважаючи на удавану подібність псевдокоду MSIL з байт-кодом Java, це все ж таки різні поняття.

Призначення CLR – перетворити проміжний код у виконуваний код по ходу виконання програми. Отже, будь-яка програма, скомпільована в псевдокод MSIL, може бути виконана в будь-якому середовищі, де є реалізація CLR. Саме таким чином частково досягається переносимість в середовищі .NET Framework.

Псевдокод MSIL перетвориться в виконуваний код за допомогою JIT-компілятора. Скорочення JIT означає "точно в термін" і відображає оперативний характер даного компілятора. Процес перетворення коду відбувається наступним чином. При виконанні програми середовище CLR активізує JIT-компілятор, який перетворює псевдокод MSIL у власний код системи на вимогу для кожної частини програми. Таким чином, програма на C# фактично виконується як власний код, незважаючи на те, що спочатку вона скомпільована у псевдокод MSIL. Це означає, що така програма виконується так само швидко, як і в тому випадку, коли вона початково скомпільована у власний код, але в той же час вона набуває все переваги переносимості псевдокоду MSIL.

Крім псевдокоду MSIL, при компіляції програми на C# виходять також метадані, які служать для опису даних, які використовуються в програмі, а також забезпечують просту взаємодію одного коду з іншим. Метадані містяться в тому ж файлі, що і псевдокод MSIL.

2.5 Короткий огляд елементів C#

В об'єкті код, дані або ж і те й інше можуть бути закритими або ж відкритими. Закриті дані або код відомі і доступні тільки іншій частині об'єкту. Це означає, що закриті дані або код недоступні частині програми, що знаходиться за межами об'єкта. Якщо ж дані або код виявляються відкритими, то вони доступні іншим частинам програми, хоча і визначені всередині об'єкта. Як правило, відкриті частини об'єкта служать для організації керованого інтерфейсу з закритими частинами.

Основною одиницею інкапсуляції в C# є клас, який визначає форму об'єкта. Він описує дані, а також код, який буде ними оперувати. У C# опис класу служить для побудови об'єктів, які є екземплярами класу. Отже, клас, по суті, являє собою ряд схематичних описів способу побудови об'єкта.

Код і дані, що становлять разом клас, називають членами. Дані, які визначаються класом, називають полями, або змінними примірника. А код, який оперує даними, міститься у функціях-членах, найтиповішим представником яких є метод. У C# метод служить в якості аналога підпрограми. (До числа інших функцій-членів відносяться властивості, події і конструктори.) Таким чином, методи класу містять код, що впливає на поля, які визначаються цим класом.

Поліморфізм, що по-грецькі означає "безліч форм", – це властивість, що дозволяє одному інтерфейсу отримувати доступ до загального класу дій. Простим прикладом поліморфізму може служити кермо автомашини, яке виконує одні й ті ж функції своєрідного інтерфейсу незалежно від виду застосовуваного механізму управління автомашиною. Це означає, що кермо діє однаково не залежно від виду рульового управління: прямої дії, з посиленням або рейковою передачею. Отже, при обертанні керма вліво автомашина завжди повертає вліво, який би вид управління в ній не застосовувався. Головна перевага однакового інтерфейсу полягає в тому, що, знаючи, як поводитися з кермом, ви зумієте водити автомашину будь-якого типу.

Той же самий принцип може бути застосований і в програмуванні. Розглянемо для прикладу стек, тобто область пам'яті, що функціонує за принципом "останнім прийшов – першим обслужений". Припустимо, що в програмі потрібні три різні типи стеків: один – для цілих значень, інший – для значень з плаваючою точкою, третій – для символьних значень. В даному прикладі алгоритм, який реалізує всі ці стеки, залишається незмінним, незважаючи на те, що в них зберігаються різнотипні дані. У мові, що не є об'єктно-орієнтованою, для цієї мети довелося б створити три різних набори стекових підпрограм з різними іменами. Але завдяки поліморфізму для реалізації всіх трьох типів стеків в C# досить створити лише один загальний набір підпрограм. Знаючи, як користуватися одним стеком, зумієш скористатися і іншими.

У більш загальному сенсі поняття поліморфізму нерідко виражається в такий спосіб: "один інтерфейс – безліч методів". Це означає, що для групи взаємопов'язаних дій можна розробити загальний інтерфейс. Поліморфізм

допомагає спростити програму, дозволяючи використовувати один і той же інтерфейс для опису загального класу дій. Вибрати конкретну дію (тобто метод) в кожному окремому випадку – це завдання компілятора. Програмісту не потрібно робити це самому. Йому досить запам'ятати і правильно використовувати загальний інтерфейс[14].

2.6 Об'єктно-орієнтоване програмування

Основним поняттям C# є об'єктно-орієнтоване програмування (ООП). Методика ООП невіддільна від C#, і тому всі програми на C# є об'єктно-орієнтованими хоча б в найменшій мірі. У зв'язку з цим дуже важливо і корисно засвоїти основні принципи ООП, перш ніж приступати до написання найпростішої програми на C#.

ООП є ефективним підходом до програмування. Методики програмування зазнали істотних змін з моменту винаходу комп'ютера, поступово пристосовуючись, головним чином, до підвищення складності програм. Коли, наприклад, з'явилися перші ЕОМ, програмування полягало в ручному перемиканні на різні двійкові машинні команди з переднього пульта управління ЕОМ. Такий підхід був цілком виправданим, оскільки програми складалися всього з декількох сотень команд. Подальше ускладнення програм призвело до розробки мови асемблера, який давав програмістам можливість працювати з більш складними програмами, використовуючи символічне уявлення окремих машинних команд. Постійне ускладнення програм викликало потребу в розробці і впровадженні в практику програмування таких мов високого рівня, як, наприклад, FORTRAN і COBOL, які надавали програмістам більше засобів для того, щоб якось справиться з постійно зростаючою складністю програм. Але як тільки можливості цих перших мов програмування були повністю вичерпані, з'явилися розробки мов структурного програмування, в тому числі і C.

На кожному етапі розвитку програмування з'являлися методи та інструментальні засоби для "приборкання" зростаючої складності програм. І на кожному такому етапі новий підхід вбирав в себе все найкраще з попередніх, знаменуючи собою прогрес в програмуванні. Це ж можна сказати і про ООП. До ООП багато проектів досягали (а іноді і перевищували) межу, за якою структурний підхід до програмування опинявся вже непрацездатним. Тому для подолання труднощів, пов'язаних з ускладненням програм, і виникла потреба в ООП.

ООП увібрало в себе всі найкращі ідеї структурного програмування, об'єднавши їх з рядом нових понять. В результаті з'явився новий і кращий спосіб організації програм. У найзагальнішому вигляді програма може бути організована одним з двох способів: навколо коду (тобто того, що фактично відбувається) або ж навколо даних (тобто того, що піддається впливу). Програми, створені тільки методами структурного програмування, як правило, організовані навколо коду. Такий підхід можна розглядати "як код, що впливає на дані".

Зовсім інакше працюють об'єктно-орієнтовані програми. Вони організовані навколо даних, виходячи з головного принципу: "дані керують доступом до коду". В об'єктно-орієнтованій мови програмування визначаються дані і код, якому дозволяється впливати на ці дані. Отже, тип даних точно визначає операції, які можуть бути виконані над даними.

Для підтримки принципів ООП все об'єктно-орієнтовані мови програмування, в тому числі і C#, повинні володіти трьома загальними властивостями: інкапсуляцією, поліморфізмом і спадкуванням.

2.7 Основи мови

Простір імен .NET Framework має у своєму розпорядженні великий набір корисних функцій. Кожна з них є членом будь-якого класу. Класи групуються по просторах імен, які мають (як правило) вкладену структуру. Засобом навігації по

множинам класів в просторі імен є оператор usung. У додатку оголошується власний простір імен і використовуються раніше оголошені простори. В процесі побудови збірки VisualStudio.NET повинен знати розташування збірок із заявленими для використання просторами імен. Розташування частини зборок системі відомо спочатку.

Розташування інших необхідних додатком збірок вказується явно (вікно Solution Explorer проекту, пункт References, Add Reference ...). Там вказується відповідний .dll або .exe файл. Зокрема, збірка, що містить класи, згруповані в просторі імен System, розташовується в файлі mscorlib.dll. Найбільш часто використовуваний простір імен – System. Розташування відповідної збірки відомо. Якщо не використовувати оператора usung System; коректне звернення до функції WriteLine (...) члену класу Console виглядало б так: System.Console.Writeline ("Ha-Ha-Ha!");//Повний кваліфіковане ім'я функції-члена класу Console, що відповідає за виведення рядка у вікно програми. При компіляції модуля, транслятор по повному імені функції (якщо використовується оператор using – то по відновленому) знаходить її код, який і використовується при виконанні збірки.

Система типів .NET підтримує дві категорії типів, кожна з яких розділена на підкатегорії: типи значень (типи-значення) і посилальні типи (типи-посилання). Схема типів представлена на рис. 2.1. Всі типи, за винятком простих типів, можуть визначатися програмістом. Всі інші типи (похідні) вимагають попереднього оголошення.

Прості (елементарні) типи – це типи, ім'я та основні властивості яких відомі компілятору. Щодо вбудованих типів компілятору не потрібно ніякої додаткової інформації. Він підтримує ці типи самостійно. Серед простих типів розрізняються:

- цілочисельні;
- з плаваючою точкою;
- decimal;
- булевський.

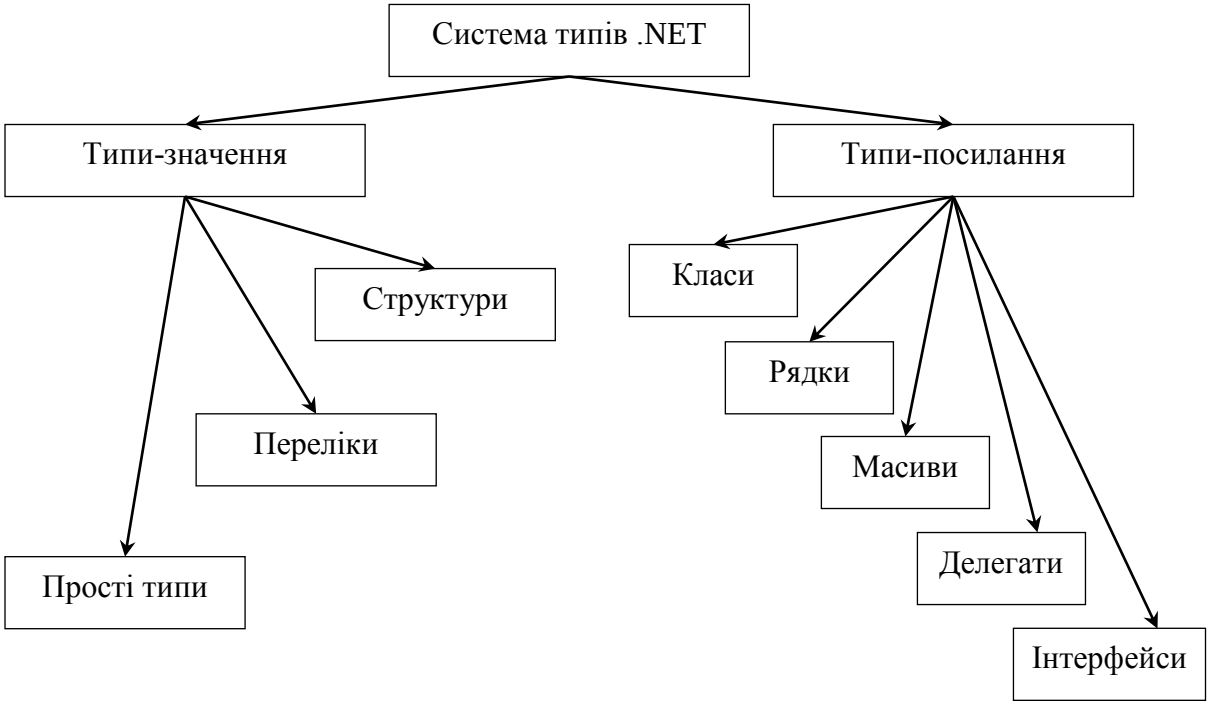

Рисунок 2.1 – Система типів .NET

Інкапсуляція – це механізм програмування, який об'єднує разом код і дані, якими він маніпулює, виключаючи як втручання ззовні, так і не правильне використання даних. В об'єктно-орієнтованій мові дані і код можуть бути об'єднані в абсолютно автономний чорний ящик. Усередині такого ящика знаходяться всі необхідні дані і код. Коли код і дані зв'язуються разом подібним чином, створюється об'єкт. Іншими словами, об'єкт – це елемент, що підтримує інкапсуляцію.

Оголошені в класі дані-члени зазвичай використовуються як змінні. Статичні члени зберігають значення, актуальні для всіх об'єктів – представників класу. Нестатичні дані-члени зберігають в змінних об'єкта інформацію, актуальну для даного об'єкта. Звернення до цих змінних проводиться з використанням точкової нотації, з явним зазначенням даного-члена, призначеного для збереження (або отримання) значення. На відміну від змінних, властивості не вказують конкретні місця зберігання. Замість цього у властивостях використовуються блоки операторів, що забезпечують доступ до даних-членів для читання і запису значень. Для завдання властивостей в мові C# використовується спеціальний синтаксис, що передбачає опис засобів отримання і установки

значення – вони називаються get accessor і set accessor. Наявність accessor'ів визначає доступність властивості для читання і запису. При зверненні до значення властивості викликається механізм читання (get accessor), при зміні значення викликається механізм запису (set accessor).

Значення властивості не повинно залежати від будь-яких обставин, зокрема, від кількості звернень до об'єкта. Якщо доступ до властивості породжує помітний побічний ефект, властивість, згідно з рекомендаціями щодо дотримання "гарного стилю", бажано реалізовувати як метод.

Літерали – подання значень у програмах на мовах високого рівня (C# в тому числі) – послідовності символів, що входять до алфавіту мови програмування і забезпечують явне уявлення значень, які використовуються для позначення початкових значень в оголошенні членів класів, змінних і констант в методах класу. Розрізняються літерали арифметичні (різних типів), логічні, символьні (включаючи Escape-послідовності), строкові.

Змінні примірника і методи є двома основними складовими класів. До сих пір клас Building містив тільки дані, але не методи. Хоча класи, що містять тільки дані, цілком припустимі, у більшості класів повинні бути також методи. Методи являють собою підпрограми, які маніпулюють даними, визначеними в класі, а в багатьох випадках вони надають доступ до цих даних. Як правило, інші частини програми взаємодіють з класом за допомогою його методів.

Метод складається з одного або декількох операторів. У грамотно написаному коді C# кожен метод виконує тільки одну функцію. У кожного методу є своє ім'я, за яким він викликається. Загалом, методу в якості імені можна привласнити будь-дійсний ідентифікатор. Слід, однак, мати на увазі, що ідентифікатор Main зарезервований для методу, з якого починається виконання програми. Крім того, в якості імен методів не можна використовувати ключові слова C#.

Масив являє собою сукупність змінних одного типу з загальним для звернення до них ім'ям. У C# масиви можуть бути як одновимірними, так і багатовимірними, хоча найчастіше застосовуються одномірні масиви. Масиви

служать самим різним цілям, оскільки вони надають зручні засоби для об'єднання пов'язаних один з одним змінних. Наприклад, в масиві можна зберігати максимальні добові температури, зареєстровані протягом місяця, перелік біржових курсів або ж назви книг з програмування з домашньої бібліотеки.

Головна перевага масиву – в організації даних таким чином, щоб ними було простіше маніпулювати. Так, якщо є масив, що містить дивіденди, що виплачуються по певній групі акцій, то, організувавши циклічне звернення до елементів цього масиву, можна без особливих зусиль розрахувати середній дохід від цих акцій. Крім того, масиви дозволяють організувати дані таким чином, щоб легко відсортувати їх.

Масивами в C# можна користуватися практично так само, як і в інших мовах програмування. Проте у них є одна особливість: вони реалізовані у вигляді об'єктів.

Нерідко додаток, створений на C#, називають керованим кодом (managed code). Це означає, що даний додаток створено на основі платформи .NET і тому керується загальномовним середовищем CLR, яке завантажує додаток і при необхідності очищає пам'ять. Але є також додатки, наприклад, створені на мові C++, які компілюються не в спільну мову CIL, як C# або VB.NET, а в звичайний машинний код. В цьому випадку .NET не керує додатком.

У той же час платформа .NET надає можливості для взаємодії з некерованим кодом. Ми поряд зі стандартними класами бібліотеки .NET можемо також використовувати збірки COM.

Код на C# компілюється в додатки або збірки з розширеннями exe або dll на мові CIL. Далі при запуску на виконання подібного додатку відбувається JIT-компіляція (Just-In-Time) в машинний код, який потім виконується. При цьому, оскільки наш додаток може бути великим і містити купу інструкцій, в поточний момент часу компілюватиметься лише та частина програми, до якої безпосередньо йде звернення. Якщо ми звернемося до іншої частини коду, то вона буде скомпільована з CIL в машинний код. При тому вже скомпільована частина програми зберігається до завершення роботи програми. У підсумку це підвищує продуктивність.

Серіалізація представляє процес перетворення будь-якого об'єкта в потік байтів. Після перетворення ми можемо цей потік байтів або записати на диск або зберегти його тимчасово в пам'яті. При необхідності можна виконати зворотний процес – десеріалізацію, тобто отримати з потоку байтів раніше збережений об'єкт.

Атрибут Serializable. При відсутності даного атрибута об'єкт Person не зможе бути серіалізований, і при спробі серіалізації буде викинуто виключення Serialization Exception.

Серіалізація застосовується до властивостей і полів класу. Якщо ми не хочемо, щоб якесь поле класу було серіалізоване, то ми його помічаємо атрибутом Non Serialized.

При спадкуванні подібного класу, слід враховувати, що атрибут Serializable автоматично не успадковується. І якщо ми хочемо, щоб похідний клас також міг би бути серіалізований, то знову ж таки застосовуємо до нього атрибут.

Хоча серіалізація є перетворенням об'єкта в певний набір байтів, але в дійсності тільки бінарним форматом вона не обмежується. Отже, в .NET можна використовувати такі формати:

– бінарний;
– SOAP;
– xml;
– JSON.

Для кожного формату передбачений свій клас: для сериализации в бінарний формат – клас BinaryFormatter, для формату SOAP – клас SoapFormatter, для xml – XmlSerializer, для json – DataContractJsonSerializer.

Хоча класи BinaryFormatter і SoapFormatter по-різному реалізують даний інтерфейс, але загальний функціонал буде той же: для серіалізації буде використовуватися метод Serialize, який в якості параметрів приймає потік, куди поміщає серіалізовані дані (наприклад, бінарний файл), і об'єкт, який треба серіалізувати . А для десеріалізації буде застосовуватися метод Deserialize, який в якості параметра приймає потік з серіалізовані даними.

Клас XmlSerializer не реалізує інтерфейс IFormatter і по функціональності в

цілому дещо відрізняється від BinaryFormatter і SoapFormatter, але і він також надає для серіалізації метод Serialize, а для десеріалізації Deserialize. І в цьому плані дуже легко при необхідності перейти від одного способу серіалізації до іншого [11].

Спадкування (inheritance) є одним з ключових моментів ООП. Завдяки спадкоємству один клас може успадкувати функціональність іншого класу.

Спадкування є одним з фундаментальних атрибутів об'єктно-орієнтованого програмування. Воно дозволяє визначити дочірній клас, який використовує (успадковує), розширює або змінює можливості батьківського класу. Клас, члени якого успадковуються, називається базовим класом. Клас, який успадковує члени базового класу, називається похідним класом.

C# і .NET підтримують тільки одиночне спадкоємство. Це означає, що кожен клас може успадковувати члени тільки одного класу. Но зате підтримується транзитивне спадкування, яке дозволяє визначити ієрархію спадкування для набору типів. Іншими словами, тип D може успадковувати можливості типу C, який в свою чергу успадковує від типу B, який успадковує від базового класу A. Завдяки транзитивності успадкування члени типу A будуть доступні для типу D.

Не всі члени базового класу успадковуються похідними класами. Наступні члени не успадковуються:

– статичні конструктори, які ініціалізують статичні дані класу;

– конструктори примірників, які викликаються для створення нового екземпляра класу. Кожен клас повинен визначати власні конструктори;

– методи завершення, які викликаються складальником сміття середовища виконання для знищення примірників класу.

Всі інші члени базового класу успадковуються похідними класами, але їх видимість не залежить від доступності. Доступність членів впливає на видимість для похідних класів наступним чином:

– закриті члени є видимими тільки в похідних класах, які вкладені в базовий клас. Для інших похідних класів вони невидимі;

– захищені члени є видимими тільки в похідних класах;

– внутрішні члени є видимими тільки в похідних класах, які знаходяться в тій же збірці, що і базовий клас. Вони не будуть видимими в похідних класах, розташованих в інших збірках;

– відкриті члени є видимими в похідних класах, а також входять в загальнодоступний інтерфейс похідних класів. Успадковані відкриті члени можна викликати так само, як якщо б вони були визначені в самому похідному класі.

Похідні класи можуть також перевизначати успадковані члени, тобто надавати альтернативну реалізацію. Перевизначити можна тільки ті члени, які в базовому класі відзначені ключовим словом virtual (віртуальний). За замовчуванням не можна перевизначати члени базового класу, не зазначені ключовим словом virtual. Спроба перевизначити член, який не є віртуальним, викликає помилку компілятора CS0506: ": неможливо перевизначити успадковані член, так як він не позначений як virtual, abstract або override".

У деяких випадках похідний клас зобов'язаний перевизначати реалізацію базового класу. Члени базового класу, відмічені ключовим словом abstract (абстрактний), обов'язково повинні перевизначатися в похідних класах. При спробі компіляції такого прикладу виникне помилка компілятора CS0534, "не реалізує успадкований абстрактний член".

Спадкування застосовується тільки для класів і інтерфейсів. Інші категорії типів (структури, делегати та перерахування) не підтримують успадкування. З цієї причини спроба компіляції коду з такого прикладу призводить до помилки компілятора CS0527: "Тип 'ValueType' в списку інтерфейсів не є інтерфейсом". Таке повідомлення про помилку означає, що спадкування не підтримується, незважаючи на можливість визначити інтерфейси, реалізовані в структурі.

Залежно від поставленого завдання і складності програми можна виділити різну кількість моделей. Моделі являють собою прості класи і розташовуються в проекті в каталозі Models. Моделі описують логіку даних.

Модель необов'язково складається тільки з властивостей, крім того, вона може мати конструктор, якісь допоміжні методи. Але головне не перевантажувати клас моделі і пам'ятати, що його призначення – описувати дані. Маніпуляції з

даними і бізнес-логіка – це більше сфера контролера.

Дані моделей зберігаються в базі даних. Щоб взаємодіяти з базою даних, дуже зручно користуватися фреймворком Entity Framework. Entity Framework підтримує підхід "Code first", який передбачає збереження або вилучення інформації з БД на SQL Server без створення схеми бази даних або використання дизайнера в Visual Studo. Навпаки, ми створюємо звичайні класи, а Entity Framework вже сам визначає, як і де зберігати об'єкти цих класів.

Випуск ASP.NET MVC 4 вже включає Entity Framework 5.0, проте в проектах по типу Empty вам доведеться підключати фреймворк через пакетний менеджер NuGet.

Щоб підключитися до бази даних через Entity Framework, нам потрібен контекст даних. Контекст даних являє собою клас, похідний від класу DbContext. Контекст даних містить одне або кілька властивостей типу DbSet<T>, де T представляє тип об'єкта, що зберігається в базі даних

Використовуючи механізм успадкування, ми можемо доповнювати і перевизначати загальний функціонал базових класах в класах-спадкоємців. Однак безпосередньо ми можемо наслідувати тільки від одного класу, на відміну, наприклад, від мови C++, де є множинне спадкування[12].

У мові C# подібну проблему дозволяють вирішити інтерфейси. Вони грають важливу роль в системі ООП. Інтерфейси дозволяють визначити деякий функціонал, який не має конкретної реалізації. Потім цей функціонал реалізують класи, які застосовують дані інтерфейси.

Для визначення інтерфейсу використовується ключове слово interface. Як правило, назви інтерфейсів в C# починаються з великої літери I, наприклад, IComparable, IEnumerable (так звана угорська нотація), однак це не обов'язкова вимога, а більше стиль програмування. Інтерфейси також, як і класи, можуть містити властивості, методи і події, тільки без конкретної реалізації.

Визначимо наступний інтерфейс IAccount, який буде містити методи і властивості для роботи з рахунком клієнта. Для додавання інтерфейсу в проект можна натиснути правою кнопкою миші на проект і в контекстному меню

вибрати Add->New Item і в діалоговому вікні додавання нового компонента вибрати Interface (рис. 2.2).

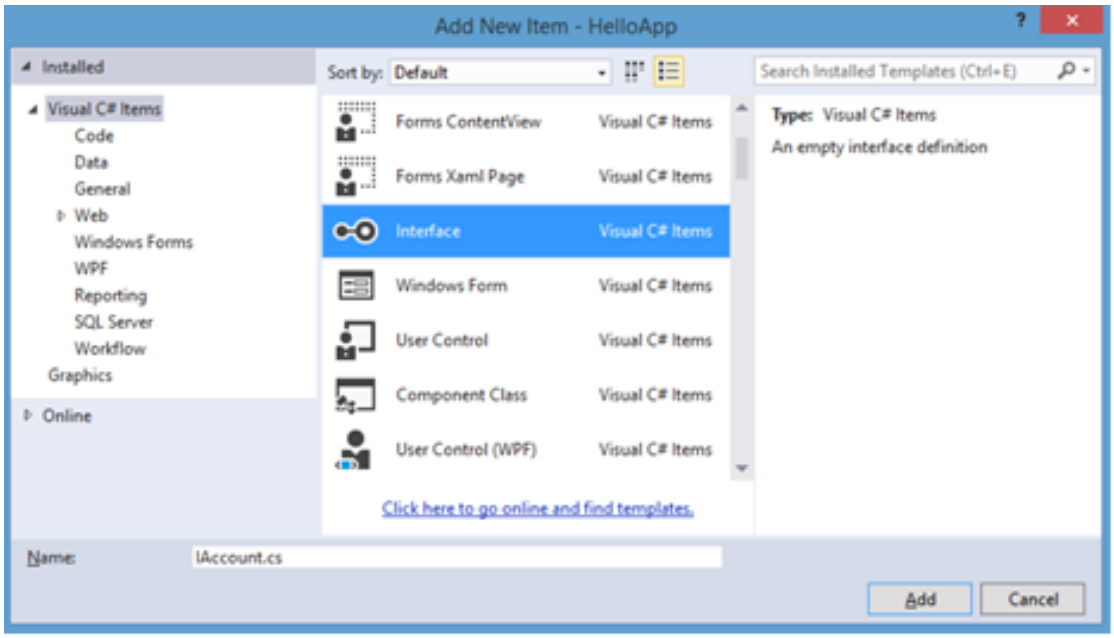

Рис. 2.2 – Діалогове вікно додавання нового компонента

В інтерфейсі методи і властивості не мають реалізації, в цьому вони зближуються з абстрактними методами абстрактних класів.

Ще один момент в оголошенні інтерфейсу: всі його члени – методи і властивості не мають модифікаторів доступу, але фактично за замовчуванням доступ public, так як мета інтерфейсу – визначення функціоналу для реалізації його класом. Тому весь функціонал повинен бути відкритий для реалізації.

Подібним чином додамо в проект ще один інтерфейс, який назвемо IClient.

Як і у випадку з абстрактними методами абстрактного класом клас Client реалізує всі методи і властивості інтерфейсу. При цьому оскільки всі методи і властивості інтерфейсу є публічними, при реалізації цих методів і властивостей в класі до них можна застосовувати тільки модифікатор public. Тому якщо клас повинен мати метод з якимось іншим модифікатором, наприклад, protected, то інтерфейс не підходить для визначення подібного методу

Якщо клас застосовує інтерфейс, то цей клас повинен реалізувати всі методи і властивості інтерфейсу. Однак також можна і не реалізувати методи,

зробивши їх абстрактними, переклавши право їх реалізації на похідні класи.

Все сказане щодо перетворення типів характерно і для інтерфейсів. Оскільки клас Client реалізує інтерфейс IAccount, то змінна типу IAccount може зберігати посилання на об'єкт типу Client:

Перетворення від класу до його інтерфейсу, як і перетворення від похідного типу до базового, виконується автоматично. Так як будь-який об'єкт Client реалізує інтерфейс IAccount.

Зворотне перетворення – від інтерфейсу до класу, що його реалізує буде аналогічно перетворенню від базового класу до похідного. Так як не кожен об'єкт IAccount є об'єктом Client (адже інтерфейс IAccount можуть реалізувати і інші класи), то для подібного перетворення необхідна операція приведення типів. І якщо ми хочемо звернутися до методів класу Client, які не визначені в інтерфейсі IAccount, але є частиною класу Client, то нам треба явним чином виконати перетворення типів: string clientName = ((Client) account).Name;

Microsoft Visual Studio – лінійка продуктів компанії Microsoft, що включають інтегроване середовище розробки програмного забезпечення і ряд інших інструментальних засобів. Дані продукти дозволяють розробляти як консольні додатки, так і додатки з графічним інтерфейсом, в тому числі з підтримкою технології Windows Forms, а також веб-сайти, веб-додатки, веб-служби як в рідному, так і в керованому кодах для всіх платформ, підтримуваних Windows, Windows Mobile, Windows CE, .NET Framework, Xbox, Windows Phone .NET Compact Framework і Silverlight.

Visual Studio включає в себе редактор вихідного коду з підтримкою технології IntelliSense і можливістю найпростішого рефакторінга коду. Вбудований відладчик може працювати як відладчик рівня вихідного коду, так і відладчик машинного рівня. Решта вбудовуваних інструментів включають в себе редактор форм для спрощення створення графічного інтерфейсу додатку, веб-редактор, дизайнер класів і дизайнер схеми бази даних. Visual Studio дозволяє створювати і підключати сторонні додатки (плагіни) для розширення функціональності практично на кожному рівні, включаючи додавання підтримки

систем контролю версій вихідного коду (як, наприклад, Subversion і Visual SourceSafe), додавання нових наборів інструментів (наприклад, для редагування і візуального проектування коду на предметно-орієнтованих мовах програмування) або інструментів для інших аспектів процесу розробки програмного забезпечення (наприклад, клієнт Team Explorer для роботи з Team Foundation Server) [15].

Visual Studio представляє собою інтегроване середовище розробки програм, створене корпорацією Microsoft. Таке середовище дає можливість правити, компілювати, виконувати і налагоджувати програми на C#, не залишаючи цю грамотно організовану середу. Visual Studio надає не тільки всі необхідні засоби для роботи з програмами, але і допомагає правильно організувати їх. Вона виявляється найбільш ефективною для роботи над великими проектами, хоча може бути з тим же успіхом використана і для розробки невеликих програм [16].

Висновки до другого розділу

Був проведений вибір та обґрунтування засобів реалізації. Також була розглянута мова C#, CRL, Visual Studio та зв'язок C# з середовищем .NET Framework.

Дизайн додатку повинен бути зручним та простим і в першу чергу орієнтуватися на користувача. Користувач повинен без особливих зусиль знайти і скористатися будь якою інформацією, що міститься у додатку.

3 ПРОЕКТУВАННЯ СИСТЕМИ ПОШУКУ НЕРУХОМОСТІ НА ОСНОВІ PARSER ТА КЛАСТЕРИЗАЦІЇ

3.1 Створення системи пошуку нерухомості Парсінг даних

Інформаційним середовищем для мого Парсера є сайт http://barcinohomes.com/poisk-obektov-po-karte. Саме з нього і починається моя робота. Насамперед, за допомогою F12 подивимося код сайту і виберемо потрібний нам сектор вилучення інформації.

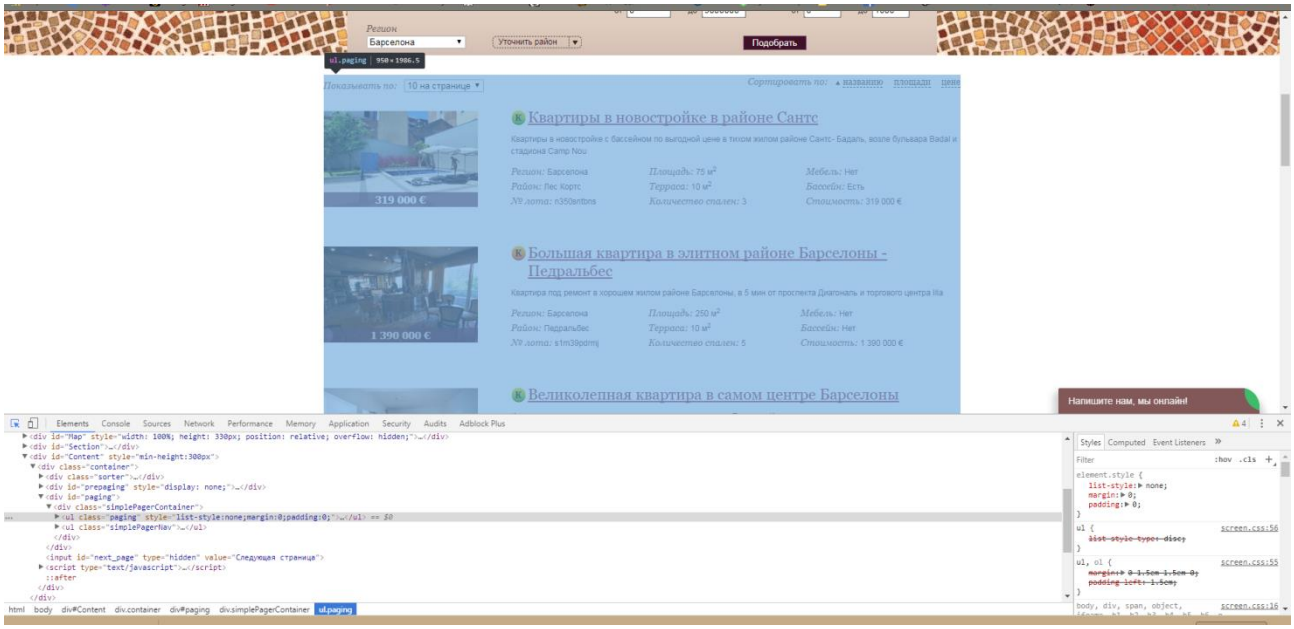

Рисунок 3.1 – Вибір класу об'єкта

Якщо на сайті немає захисту від ботів і весь необхідний контент віддається відразу, то можна обійтися більш легковажним рішенням – використовувати бібліотеку HtmlAgilityPack. Підключається вона через NuGet. Ця бібліотека будує DOM дерево по HTML.

Щоб завантажити HTML в об'єкт моделі документа (DOM) і розбити на вузли використовується HTML Agility Pack (HAP). Був успішно використаний

Html Agility Pack для клієнта, проаналізовані HTML документи, щоб отримати необхідну інформацію. Розширення CSSSelector буде додавати новий потужний рівень абстракції, щоб зібрати необхідні дані.

HtmlAgility має ряд класів, доступних для додавання і перерахування, які представляють різні частини DOM, ці класи включають HtmlAttribute, HtmlAttributeCollection, HtmlCommentNode і так далі.

Всього нам доступно двадцять основних класів (рис. 3.2).

Class	Description
Crc32	A utility class to compute CRC32.
HtmlAttribute	Represents an HTML attribute.
HtmlAttributeCollection	Represents a combined list and collection of HTML nodes.
HtmlCommentNode	Represents an HTML comment.
HtmlDocument	Represents a complete HTML document.
HtmlEntity	A utility class to replace special characters by entities and vice-versa. Follows HTML 4.0 specification found at http://www.w3.org/TR/html4/sgml/entities.html
HtmlNode	Represents an HTML node.
HtmlNodeCollection	Represents a combined list and collection of HTML nodes.
HtmlNodeNavigator	Represents an HTML navigator on an HTML document seen as a data store.
HtmlParseError	Represents a parsing error found during document parsing.
HtmlTextNode	Represents an HTML text node.
HtmlWeb	A utility class to get HTML document from HTTP.
HtmlWebException	Represents an exception thrown by the HtmlWeb utility class.
MixedCodeDocument	Represents a document with mixed code and text. ASP, ASPX, JSP, are good example of such documents.
MixedCodeDocumentCodeFragment	Represents a fragment of code in a mixed code document.
MixedCodeDocumentFragment	Represents a base class for fragments in a mixed code document.
MixedCodeDocumentFragmentList	Represents a list of mixed code fragments.
MixedCodeDocumentFragmentList.MixedCodeDocumentFragmentEnumerator	Represents a fragment enumerator.
MixedCodeDocumentTextFragment	Represents a fragment of text in a mixed code document.

Рисунок 3.2 – Основні класи для PARSER

Назви методів відповідають інтерфейсам DOM: GetElementbyId(), CreateAttribute(), CreateElement() і т.ін. HTML переганяється в Xml, а HtmlDocument і інші класи, з огляду на це доступні такі можливості як:

– Linq to Objects (via LINQ to Xml);

– XPATH;

– XSLT.

3.2 Створення проекту в Visual Studio

3.2.1 Створення додатку

При створенні програми в Visual Studio необхідно спочатку створити проект і рішення. Створимо консольний додаток Windows.

1. У рядку меню виберемо Файл, Створити, Проект.

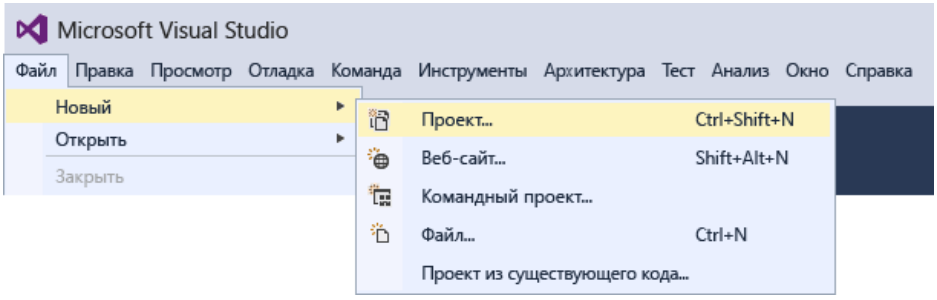

Рисунок 3.2 – Створення програми

2. В категорії Visual C# виберемо шаблон додаток WPF і назвемо проект.

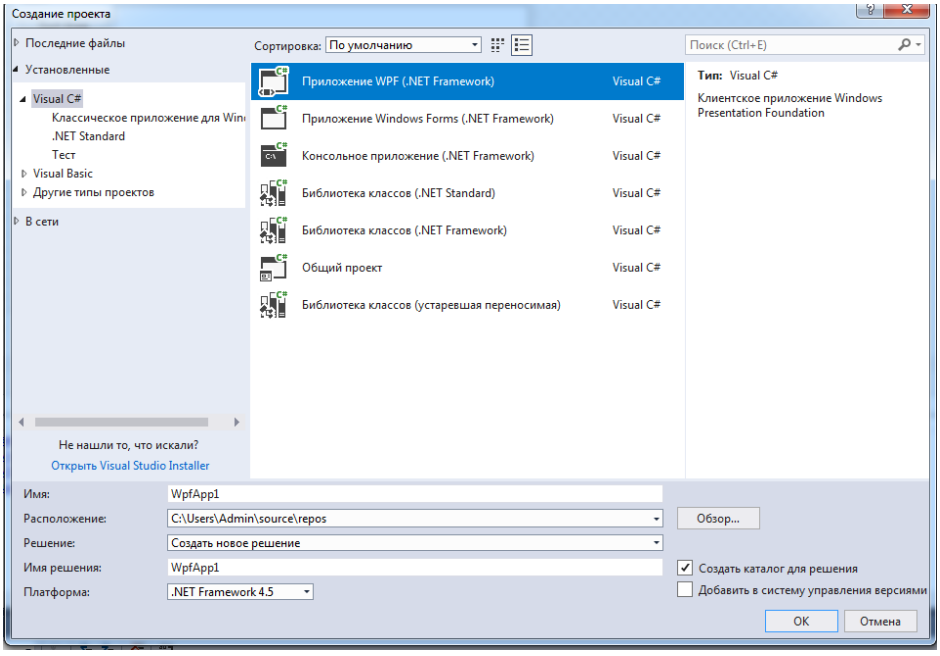

Рисунок 3.3 – Створення проекту

Проект GreetingsConsoleApp і рішення з основними файлами для додаток WPF створяться і автоматично завантажуються в Оглядач рішень. Файл GreettingsConsoleApp.cpp відкриється в редакторі коду. В оглядачі рішень відображаються наступні елементи.

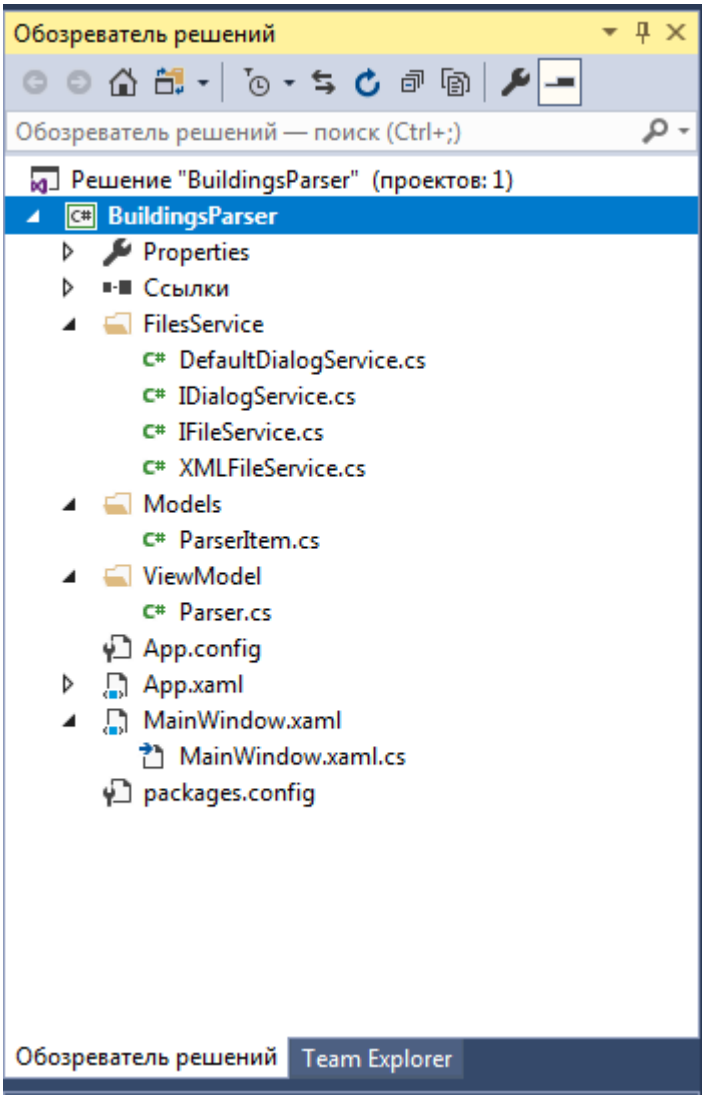

Рисунок 3.4 – Елементи проекту

3.3 Модель даних

ParserItem використовуєтьсяя в класі парсер що б легше було записувати інформацію і мати легший доступ до неї.

```
namespace BuildingsParser.Models
{
    public class ParserItem
    {
        public string Name { get; set; }
        public string Description { get; set; }


        public string Region { get; set; }
        public int Area { get; set; }
        public string Furniture { get; set; }
        public string District { get; set; }
        public string Terrace { get; set; }
        public string Pool { get; set; }
        public string LotNumber { get; set; }
        public int Bedrooms { get; set; }
        public int Price { get; set; }

    }
}
```

Парсінг сайтів здійснюється таким чином:

Усередині методу створюємо змінну Items і присвоюється новий лист.

```
List<ParserItem> Items = new List<ParserItem>();
```

Присвоювання змінній docHtml сторінку сайту якого ми збираємося парсити:

```
HtmlDocument docHtml = new HtmlWeb().Load("http://barcinohomes.com/poisk-obektov-po-karte");
```

Шукаємо всі теги ("`//ul`") – Маркований список і присвоюємо її змінній nodes з типом даних HtmlNodeCollection:

```
HtmlNodeCollection nodes = docHtml.DocumentNode.SelectNodes("//ul");
```

Шукаємо усі теги ("`// li`") – елементи маркованого списку в першому рядку колекції nodes і прирівнюємо її колекції lists:

```
HtmlNodeCollection lists = nodes[0].SelectNodes("//li");
```

Далі ми перебираємо всі рядки масиву lists:

```
foreach (HtmlNode list in lists)
 ищем в нем нужные нам елементы
HtmlNodeCollection divs2 = divs[3].ChildNodes;
HtmlNodeCollection content = divs2[3].ChildNodes;
HtmlNodeCollection table = content[5].ChildNodes;
HtmlNodeCollection table1 = table[1].ChildNodes;
HtmlNodeCollection table2 = table[3].ChildNodes;
HtmlNodeCollection table3 = table[5].ChildNodes;
```

Форматуємо їх і додаємо в масив Items:

```
Items.Add(new ParserItem()
{
Name = content[1].InnerText,
Description = content[3].InnerText,
Region = splited1,
Area = Int32.Parse(splited2[1]),
Furniture = splited3,
District = splited4,
Terrace = splited5[1],
Pool = splited6,
LotNumber = splited7,
Bedrooms = Int32.Parse(splited8),
Price = Int32.Parse(splited9)
}
```

Коли ми звертатимося до методу він буде повертати (return Items;).

3.4 Серіалізація і робота з даними

Серіалізація. Створюємо клас який приймає потрібний тип даних, всередині створюємо змінну і два методи.

Методи приймають тільки один параметр, шлях до файлу.

1 метод десеріалізує і прирівнює його до змінної створеної раніше:

```
public bool GetModel(string Path)
```

2 метод серіалізує все що є у змінній і створює файл:

```
public bool CreateModel(string Path)
```

Діалогові вікна. Ми створюємо інтерфейс з назвою IDialogService і створюємо в ньому методи і змінні

```
public interface IDialogService
```

Створюємо клас DefaultDialogService і успадковує інтерфейс IDialogService. Ми визначаємо змінну і три методи:

1. Метод відкриває з файлу;

2. Зберігає в файл;

3. Потрібен для ідентифікації помилок.

Головне вікно програми. Ініціалізуємо наш серіалізатор і діалогові вікна:

```
DefaultDialogService dialogService = new DefaultDialogService ();
XMLSerialization <Container> fileService = new XMLSerialization <Container>
();
```

У конструкторі створюємо екземпляр класу Parser, викликаємо метод з цього класу Pars () і прирівнюємо його до змінної Items. Головна сторінка програми на рис. 3.5.

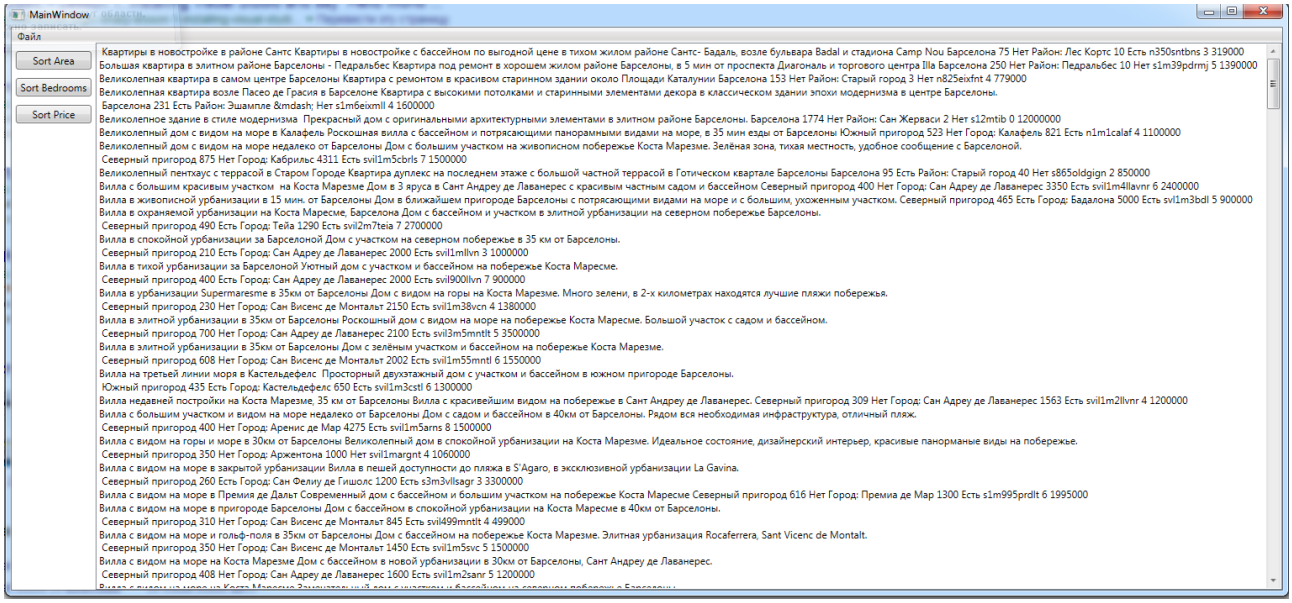

Рисунок 3.5 – Головна сторінка програми

Кнопки Меню. Оберемо один з наданих варіантів (рис. 3.6).

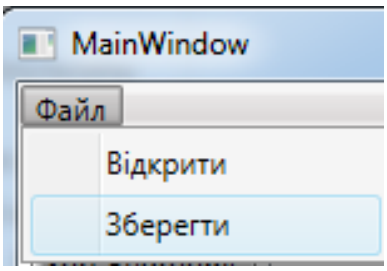

Рисунок 3.6 – Вибір функції

Виберемо один з наданих варіантів:

```
private void MenuItem1_OnClick(object sender, RoutedEventArgs e)
        {
            if (dialogService.OpenFileDialog())
            {
                listBox1.Items.Clear();
                fileService.GetModel(dialogService.FilePath);
                Items = fileService._Model;
                ParseItems();
            }
        }
```

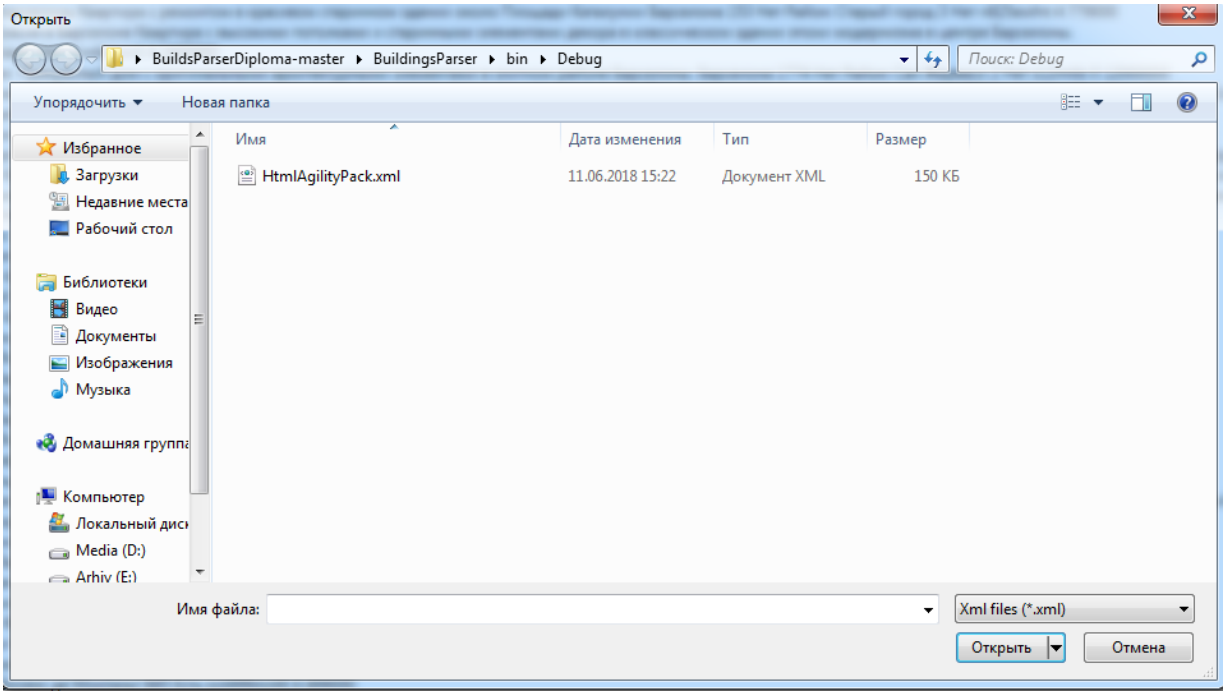

Рисунок 3.7 – Відкриття файлу

```
        private void MenuItem2_OnClick(object sender, RoutedEventArgs e)
        {
            if (dialogService.SaveFileDialog())
            {
                fileService._Model = Items;
                fileService.CreateModel(dialogService.FilePath);
            }
        }
```

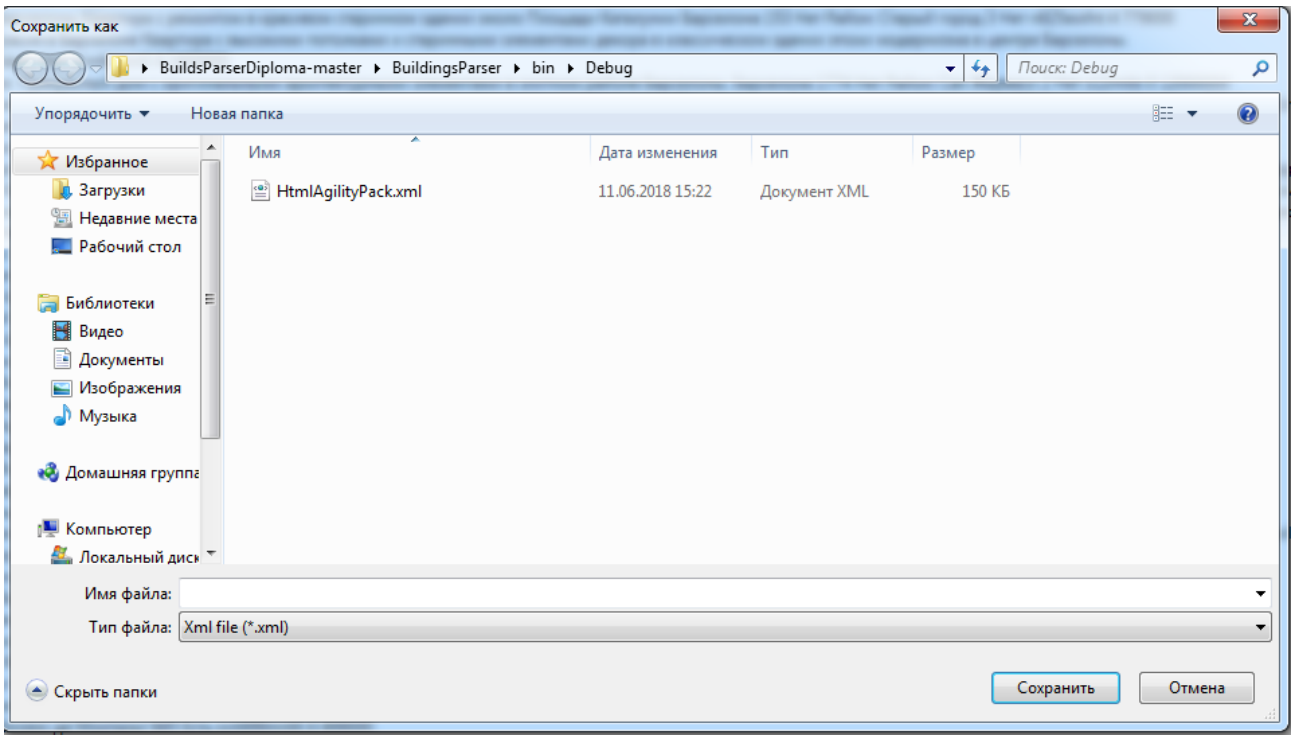

Рисунок 3.8 – Збереження даних

Алгоритм кластеризації спочатку визначає зв'язки в наборі даних і формує ряд кластерів на основі цих зв'язків.

Після першого визначення кластерів алгоритм обчислює, як кластери являють групування точок, а потім намагається повторно визначити групування, щоб створити кластери, які краще представляють дані. Алгоритм послідовно виконує цей процес.

Можна налаштовувати роботу даного алгоритму, вибираючи конкретний метод об'єднання в кластери, обмежуючи максимальну кількість кластерів або змінюючи розмір, необхідний для створення кластера. Цей алгоритм включає два популярних методи кластеризації.

Кластеризація методом К-середніх яка реализована в данному додатку та при натисканні однієї з наданних кнопок як показано на рис. 3.9. виробляє процес кластерізації по центру кластера кожного типу.

Рісунок 3.9 – Кнопка запуску сортування

Висновок до розділу 3

Була описана розробка коду розмітки, програмного коду, з поясненнями до них. Описано функціональність додатку та представлені скріншоти головних активностей додатка, з поясненнями до кожної.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

В даному розділі проведено аналіз потенційних небезпечних та шкідливих виробничих факторів, причин пожеж. Розглянуті заходи, які дозволяють забезпечити гігієну праці і виробничу санітарію. На підставі аналізу розроблені заходи з техніки безпеки та рекомендації з пожежної профілактики.

Завданням даної роботи бакалавра було створення системи пошуку нерухомості Парсінг даних. Так як в процесі проектування використовувався ПК, то аналіз потенційно небезпечних і шкідливих виробничих чинників виконується для персонального комп'ютера на якому буде розроблена система і додаток.

4.1 Загальні питання з охорони праці

В законі України «Про охорону праці» визначається, що охорона праці – це система правових, соціально-економічних, організаційно-технічних, санітарно-гігієнічних і лікувально- профілактичних заходів та засобів, спрямованих на збереження життя, здоров'я і працездатності людини у процесі трудової діяльності.

Пріоритет життя і здоров'я працівників щодо результатів виробничої діяльності підприємств – один з основних принципів державної політики в галузі охорони праці.

Одним з резервів підвищення ефективності виробництва є вдосконалення методів забезпечення безпеки праці, тому що травматизм визначає істотну частину непродуктивних втрат робочого часу, а боротьба з травматизмом, крім гуманістичного спрямування, має чітко виражений економічний аспект. Безпека праці виступає і як один з факторів, які забезпечують високу продуктивність праці.

Сутність охорони праці полягає у визначенні можливих небезпечних і шкідливих виробничих факторів, що можуть проявитися при проведенні

запланованих для виконання робіт; прогнозуванні моментів прояву зазначених факторів; проведенні необхідних профілактичних заходів.

При виконанні роботи безпосередньо на персональних комп’ютерах та з використанням друкувальної, копіювальної та іншої офісної техніки характерні такі небезпечні фактори, як:

– шум і вібрація, джерелом яких є комп’ютери, офісна техніка;

– забруднення атмосфери шкідливими речовинами (озон, оксиди азоту, пил);

– широке використання електричного струму: комп’ютерами, в штучному освітленні;

– електромагнітні поля, інфрачервоне та іонізуюче випромінювання;

– тепловиділення;

– статична електрика.

Крім того, програміст піддається значному розумовому і психоемоційному навантаженню, високій напрузі зорової діяльності.

4.1.1 Вимоги до приміщень

Геометричні розміри приміщення зазначені в табл. 4.1.

Таблиця 4.1 – Розміри приміщення

Найменування	**Значення**
Довжина, м	5
Ширина, м	5
Висота, м	3
Площа, $м^2$	25
Об’єм, $м^3$	75

Згідно з ДСН 3.3.6.042-99 [17] розмір площі для одного робочого місця оператора персонального комп’ютера має бути не менше 6 кв. м, а об’єм – не менше 20 куб. м. Отже, дане приміщення цілком відповідає зазначеним нормам.

4.1.2 Вимоги до організації місця праці

При порівнянні відповідності характеристик робочого місця нормативним основні вимоги до організації робочого місця за ДСанПіН 3.3.2.007-98 [20] (табл. 4.2) і відповідними фактичними значеннями для робочого місця, констатуємо повну відповідність.

Таблиця 4.2 – Характеристики робочого місця

Найменування параметра	**Фактичне значення**	**Нормативне значення**
Висота робочої поверхні, мм	750	680 ÷ 800
Висота простору для ніг, мм	730	не менше 600
Ширина простору для ніг, мм	660	не менше 500
Глибина простору для ніг, мм	700	не менше 650
Висота поверхні сидіння, мм	470	400 ÷ 500
Ширина сидіння, мм	400	не менше 400
Глибина сидіння, мм	400	не менше 400
Висота поверхні спинки, мм	600	не менше 300
Ширина опорної поверхні спинки, мм	500	не менше 380
Радіус кривини спинки в горизонтальній площині, мм	400	400
Відстань від очей до екрану дисплея, мм	800	700 ÷ 800

4.2 Виробнича санітарія

На підставі аналізу небезпечних та шкідливих факторів при виробництві (експлуатації), пожежної безпеки можуть бути надалі вирішені питання необхідності забезпечення працюючих достатньою кількістю освітлення, вентиляції повітря, організації заземлення, тощо.

4.2.1 Аналіз небезпечних та шкідливих факторів при виробництві (експлуатації) виробу

Аналіз небезпечних та шкідливих виробничих факторів виконується у табличній формі (табл. 4.3). Роботу, пов'язану з ЕОП з ВДТ, у тому числі на тих, які мають робочі місця, обладнані ЕОМ з ВДТ і ПП, виконують із забезпеченням виконання НПАОП 0.00-1.28-10 [21], яке встановлює вимоги безпеки до обладнання робочих місць, до роботи із застосуванням ЕОМ з ВДТ і ПП. Основними робочими характеристиками персонального комп'ютера є:

– робоча напруга *U=+220В +-5%;*
– робочий струм *I=2А*;
– споживана потужність *P=350 Вт*.

Таблиця 4.3 – Аналіз небезпечних і шкідливих виробничих факторів

Небезпечні і шкідливі виробничі фактори	**Джерела факторів (види робіт)**	**Кількісна оцінка**	**Нормативні документи**
Фізичні:			
підвищена або знижена вологість повітря	-//-	2	[17]
підвищений рівень напруги електричної мережі, замикання якої може відбутися через тіло людини	-//-	4	[18] [19]
Психофізіологічні:			
нервово-психічна перевантаження (розумове, перенапруження аналізаторів – зорових)	- пошук інформації для постановки теми; - пошук та аналіз аналогів і літератури; - пошук наявних технологій, моделювання та аналіз алгоритмів; - виконання роботи за темою диплома, тестування; - оформлення роботи	4	[20] [21]
фізичні (статичне – сидіння)	порушення умов праці (організації місця праці: сидіння користувача) та організації робочого часу (безперервна робота)	2	[20] [21]

Робочі місця мають відповідати вимогам державних санітарних правил і норм роботи з візуальними дисплейними терміналами електронно-обчислювальних машин, затверджених постановою Головного державного санітарного лікаря України від 10.12.98 N 7 [20].

4.2.2 Пожежна безпека

Висока щільність елементів в електронних схемах призводить до значного підвищення температури окремих вузлів (80…100℃). При проходженні електричного струму по провідниках і деталей виділяється тепло, що в умовах їх високої щільності може привести до перегріву, і може служити причиною запалювання ізоляційних матеріалів. Слабкий опір ізоляційних матеріалів дії температури може викликати порушення ізоляції і привести до короткого замикання між струмоведучими частинами обладнання (шини, електроди).

Для гасіння пожеж в офісному приміщенні пропонується використовувати порошкові або вуглекислотні вогнегасники, так як вони є універсальними.

Заземлені конструкції, що знаходяться в приміщеннях, де розміщені робочі місця (батареї опалення, водопровідні труби, кабелі із заземленим відкритим екраном), надійно захищені діелектричними щитками та/або сітками з метою недопущення потрапляння працівника під напругу.

В приміщенні наявна затверджена "План-схема евакуації з кабінету (приміщення)".

Горючими матеріалами в приміщенні, де розташовані ЕОМ, є:

– поліамід – матеріал корпусу мікросхем, горюча речовина, температура самозаймання 420℃;

– полівінілхлорид – ізоляційний матеріал, горюча речовина, температура запалювання 335℃, температура самозаймання 530℃;

– склотекстоліт ДЦ – матеріал друкарських плат, важкогорючий матеріал,

показник горючості 1.74, не схильний до температурного самозаймання;

– пластикат кабельний №489 – матеріал ізоляції кабелів, горючий матеріал, показник горючості більше 2.1;

– деревина – будівельний і обробний матеріал, з якого виготовлені меблі, горючий матеріал, показник горючості більше 2.1, температура запалювання 255℃, температура самозаймання 399℃.

Простори усередині приміщень в межах, яких можуть утворюватися або знаходиться пожежонебезпечні речовини і матеріали відповідно до НАПБ Б.03.002-2007 [22] відносяться до пожежонебезпечної зони класу П-IIа. Це обумовлено тим, що в приміщенні знаходяться тверді горючі та важкозаймисті речовини та матеріали. Приміщенню, у якому розташоване робоче місце, присвоюється II ступень вогнестійкості.

Причинами можливого загоряння і пожежі можуть бути:

– несправність електроустановки;

– конструктивні недоліки устаткування;

– коротке замикання в електричних мережах;

– запалювання горючих матеріалів, що знаходяться в безпосередній близькості від електроустановки.

Продуктами згорання, що виділяються на пожежі, є: окис вуглецю; сірчистий газ; окис азоту; синильна кислота; акромін; фосген; хлор і ін. При горінні пластмас, окрім звичних продуктів згорання, виділяються різні продукти термічного розкладання: хлорангідридні кислоти, формальдегіди, хлористий водень, фосген, синильна кислота, аміак, фенол, ацетон, стирол [23].

4.2.3 Електробезпека

Виконуються наступні вимоги електробезпеки: ПК, периферійні пристрої та устаткування для обслуговування, електропроводи і кабелі за виконанням та

ступенем захисту відповідають класу зони за ПУЕ (правила улаштування електроустановок), мають апаратуру захисту від струму короткого замикання та інших аварійних режимів. Лінія електромережі для живлення ПК, периферійних пристроїв і устаткування для обслуговування, виконана як окрема групова три провідна мережа, шляхом прокладання фазового, нульового робочого та нульового захисного провідників. Нульовий захисний провідник використовується для заземлення (занулення) електроприймачів. Штепсельні з'єднання та електророзетки крім контактів фазового та нульового робочого провідників мають спеціальні контакти для підключення нульового захисного провідника. Електромережа штепсельних розеток для живлення персональних ПК, укладено по підлозі поруч зі стінами відповідно до затвердженого плану розміщення обладнання та технічних характеристик обладнання. Металеві труби та гнучкі металеві рукави заземлені. Захисне заземлення включає в себе заземлюючих пристроїв і провідник, який з'єднує заземлюючий пристрій з обладнанням, яке заземлюється – заземлюючий провідник.

4.3 Гігієнічні вимоги до параметрів виробничого середовища

Для зменшення негативного впливу ЕОМ з ВДТ і ПП планують комплекс медико-гігієнічних, адміністративно-технічних і ергономічних заходів:

– контроль за конструкцією, добрим станом і функціями ВДТ;

– створення оптимальних умов для праці у виробничому приміщенні (мікроклімат, освітлення, захист від опромінювання комп’ютера, іонізації повітря тощо);

– відповідність місця праці рекомендаціям ергономіки та гігієни;

– раціональний режим праці;

– підвищення опору організму користувачів комп’ютера до дії несприятливих чинників (антистресова дія, аеробіка та спеціальні фізичні вправи, психологічні та соціальні заходи, профілактичне харчування);

– диспансерне медико-гігієнічне обслуговування з цілеспрямованим проведенням оздоровчих (наприклад, корекція зору) і профілактичних заходів;

– особиста участь працівника у догляді за своїм здоров’ям;

– вентиляція та кондиціонування приміщень установи

Для забезпечення оптимальних умов мікроклімату у приміщеннях з ВДТ використовують системи вентиляції та кондиціювання повітря, а також природне провітрювання.

4.3.1 Мікроклімат

Мікроклімат робочих приміщень – це клімат внутрішнього середовища цих приміщень, що визначається діючої на організм людини з'єднанням температури, вологості, швидкості переміщення повітря. В даному приміщенні проводяться роботи, що виконуються сидячи і не потребують динамічного фізичного напруження, то для нього відповідає категорія робіт Іа. Отже оптимальні значення для температури, відносної вологості й рухливості повітря для зазначеного робочого місця відповідають ДСН 3.3.6.042-99 [17] і наведені в табл. 4.4:

Таблиця 4.4 – Нормативні параметри мікроклімату для приміщень з ВДТ

Пора року	**Категорія робіт**	**Температура, C^0**	**Відносна вологість %**	**Швидкість руху повітря, м/с**
Холодна	Легка-Іа	22-24	40-60	0,1
Тепла	Легка-Іа	23-25	40-60	0,1

4.3.2 Освітлення

Для виробничих та адміністративних приміщень світловий коефіцієнт приймається не менше – 1/8, в побутових – 1/10:

$$S_b = \left(\frac{1}{5} / \frac{1}{10}\right) \cdot S_n, \qquad (4.1)$$

де S_b – площа віконних прорізів, м^2;

S_n – площа підлоги, м^2.

$$S_n = a \cdot b = 5 \cdot 5 = 25 \text{ м}^2;$$

$$S = \frac{1}{8} \cdot 25 = 3{,}125 \text{ м}^2.$$

Приймаємо 2 вікна площею S=1,6 м^2 кожне.

Світильники загального освітлення розташовуються над робочими поверхнями в рівномірно-прямокутному порядку. Для організації освітлення в темний час доби передбачається обладнати приміщення, довжина якого складає 5 м, ширина 5 м, світильниками ЛПО2П, оснащеними лампами типа ЛБ (дві по 80 Вт) з світловим потоком 5400 лм кожна. Розрахунок штучного освітлення виробляється по коефіцієнтах використання світлового потоку, яким визначається потік, необхідний для створення заданої освітленості при загальному рівномірному освітленні. Розрахунок кількості світильників n виробляється по формулі (4.2):

$$n = \frac{E \cdot S \cdot Z \cdot K}{F \cdot U \cdot M}, \qquad (4.2)$$

де Е – нормована освітленість робочої поверхні, визначається нормами – 300 лк;

S – освітлювана площа, м^2; S = 25 м^2;

Z – поправочний коефіцієнт світильника (Z = 1,15 для ламп розжарювання та ДРЛ; Z = 1,1 для люмінесцентних ламп) приймаємо рівним 1,1;

K – коефіцієнт запасу, що враховує зниження освітленості в процесі експлуатації – 1,5;

U – коефіцієнт використання, залежний від типу світильника, показника індексу приміщення і т.п. – 0,575;

M – число люмінесцентних ламп в світильнику – 2;

F – світловий потік лампи – 5400 лм (для ЛБ-80).

Підставивши числові значення у формулу (4.2), отримуємо:

$$n = \frac{300 \cdot 25 \cdot 1{,}1 \cdot 1{,}5}{5400 \cdot 0{,}575 \cdot 2} \approx 2{,}0.$$

Приймаємо освітлювальну установку, яка складається з 2-х світильників, які складаються з двох люмінесцентних ламп загальною потужністю 160 Вт, напругою – 220 В.

4.4 Вентилювання

У приміщенні, де знаходяться ЕОМ, повітрообмін реалізується за допомогою природної організованої вентиляції (вентиляційні шахти), тобто при $V_{приміщення} > 40$ м^3 на одного працюючого допускається природна вентиляція. Цей метод забезпечує приток потрібної кількості свіжого повітря, що визначається в СНіП. Також має здійснюватися провітрювання приміщення, в залежності від погодних умов, тривалість повинна бути не менше 10 хв. Найкращий обмін повітря здійснюється при наскрізному провітрюванні.

4.5 Заходи з організації виробничого середовища та попередження виникнення надзвичайних ситуацій

1. Заходи безпеки під час експлуатації персонального комп’ютера та периферійних пристроїв передбачають:

– правильне організування місця праці та дотримання оптимальних режимів праці та відпочинку під час роботи з ПК;

– експлуатацію сертифікованого обладнання;

– дотримання заходів електробезпеки;

– забезпечення оптимальних параметрів мікроклімату;

– забезпечення раціонального освітлення місця праці (освітленість робочого місця не перевищувала 2/3 нормальної освітленості приміщення);

– облаштовуючи приміщення для роботи з ПК, потрібно передбачити припливно-витяжну вентиляцію або кондиціювання повітря:

а) якщо об’єм приміщення 20 $м^3$, то потрібно подати не менш як 30 $м^3$/год повітря;

б) якщо об’єм приміщення у межах від 20 до 40 $м^3$, то потрібно подати не менш як 20 $м^3$/год повітря;

в) якщо об’єм приміщення становить понад 40 $м^3$, допускається природна вентиляція, у випадку, коли немає виділення шкідливих речовин.

2. Заходи безпеки під час експлуатації інших електричних приладів передбачають дотримання таких правил:

– постійно стежити за справним станом електромережі, розподільних щитків, вимикачів, штепсельних розеток, лампових патронів, а також мережевих кабелів живлення, за допомогою яких електроприлади під'єднують до електромережі;

– постійно стежити за справністю ізоляції електромережі та мережевих кабелів, не допускаючи їхньої експлуатації з пошкодженою ізоляцією;

– не тягнути за мережевий кабель, щоб витягти вилку з розетки;

– не закривати меблями, різноманітним інвентарем вимикачі, штепсельні розетки;

– не підключати одночасно декілька потужних електропристроїв до однієї розетки, що може викликати надмірне нагрівання провідників, руйнування їхньої ізоляції, розплавлення і загоряння полімерних матеріалів;

– не залишати включені електроприлади без нагляду;

4.5.1 Розрахунок захисного заземлення (забезпечення електробезпеки будівлі)

Згідно з класифікацією приміщень за ступенем небезпеки ураження електричним струмом [24], приміщення в якому проводяться всі роботи відноситься до першого класу (без підвищеної небезпеки). Під час роботи використовуються електроустановки з напругою живлення 36 В, 220 В, та 360 В. Опір контуру заземлення повинен мати не більше 4 Ом.

Послідовність розрахунку.

1. Визначається необхідний опір штучних заземлювачів $R_{шт.з}$:

$$R_{шт.з} = \frac{R_д \cdot R_{пр.з}}{R_{пр.з} - R_д}, \qquad (4.3)$$

де $R_{пр.з}$ – опір природних заземлювачів;

$R_д$ – допустимий опір заземлення.

Якщо природні заземлювачі відсутні, то $R_{шт.з} = R_д$.

Підставивши числові значення у формулу (4.3), отримуємо:

$$R_{шт.з} = \frac{4 \cdot 40}{40 - 4} \approx 4 Ом.$$

2. Опір заземлення в значній мірі залежить від питомого опору ґрунту ρ, Ом·м. Приблизне значення питомого опору глини приймаємо ρ=40 Ом·м (табличне значення).

3. Розрахунковий питомий опір ґрунту, $P_{розр}$, Ом·м, визначається відповідно для вертикальних заземлювачів $\rho_{розр.в}$, і горизонтальних $P_{розр.г}$, Ом·м за формулою:

$$P_{розр.} = \Psi * p \qquad (4.4)$$

де ψ – коефіцієнт сезонності для вертикальних заземлювачів I кліматичної зони з нормальною вологістю землі, приймається для вертикальних заземлювачів $P_{розр.в}$=1,7 і горизонтальних $P_{розр.г}$ =5,5 Ом·м.

$$Р_{озр.в} = 1,\ 7 * 40 = 68 \text{ Ом/м};$$
$$Р_{озр.г} = \cdot 5,5 * 40 = 220 \text{ Ом/м}.$$

4. Розраховується опір розтікання струму вертикального заземлювача Rв, Ом, за (4.5).

$$R_B = \frac{P_{розрв}}{2 * \pi * 1_B} * \left(\ln \frac{2 * 1_B}{d_{CT}} + \frac{1}{2} * \ln \frac{4 * t + 1_B}{4 * t - 1_B} \right) \qquad (4.5)$$

де $l_в$ – довжина вертикального заземлювача (для труб – 2-3 м; $l_в$=3 м);

$d_{ст}$ – діаметр стержня (для труб – 0,03-0,05 м; $d_{ст}$=0,05 м);

t – відстань від поверхні землі до середини заземлювача, яка визначається за формулою (4.6):

$$t = h_B + \frac{1_B}{2} \qquad (4.6)$$

де $h_в$ – глибина закладання вертикальних заземлювачів (0,8 м); тоді $t = 0.8 + \frac{3}{2} = 2{,}3м$.

$$R_B = \frac{68}{2*\pi*3}*\left(\ln\frac{2*3}{0{,}05} + \frac{1}{2}*\ln\frac{4*2{,}3+3}{4*2{,}3-3}\right) = 18{,}5Ом.$$

5. Визначається теоретична кількість вертикальних заземлювачів n штук, без урахування коефіцієнта використання ηв:

$$n = \frac{2*R_B}{R_д} = \frac{2*18{,}5}{4} = 9{,}25. \quad (4.7)$$

6. Визначається необхідна кількість вертикальних заземлювачів з урахуванням коефіцієнта використання $n_в$, шт:

$$n_B = \frac{2*R_B}{R_д*\eta_B} = \frac{2*18{,}5}{4*0{,}57} = 16{,}2 \approx 16. \quad (4.8)$$

7. Визначається довжина з'єднувальної стрічки горизонтального заземлювача lc, м:

$$l_c = 1{,}05*L_B*(n_B - 1). \quad (4.9)$$

де $l_в$ – відстань між вертикальними заземлювачами, (прийняти за $L_в$ = 3м);

$n_в$ – необхідна кількість вертикальних заземлювачів.

$$l_c = 1{,}05*3*(16-1) \approx 48м.$$

8. Визначається опір розтіканню струму горизонтального заземлювача

(з’єднувальної стрічки) $R_г$, Ом:

$$R_Г = \frac{P_{розрг}}{2*\pi*l_c} * \ln\frac{2*l_c^2}{d_{ст}*h_Г} \quad (4.10)$$

де $d_{см}$ – еквівалентний діаметр смуги шириною b, $d_{см}$ =0,95b, b = 0,15 м;

$h_г$ – глибина закладання горизонтальних заземлювачів (0,5 м);

l_c – довжина з’єднувальної стрічки горизонтального заземлювача $l_с$, м.

$$R_Г = \frac{220}{2*\pi*48} * \ln\frac{2*48^2}{0{,}95*0{,}15*0{,}5} = 8{,}1Ом.$$

9. Визначається коефіцієнт використання горизонтального заземлювача $\eta_с$. відповідно до необхідної кількості вертикальних заземлювачів $n_в$. Коефіцієнт використання з'єднувальної смуги $\eta_с$=0,3 (табличне значення).

10. Розраховується результуючий опір заземлювального електроду з урахуванням з’єднувальної смуги:

$$R_{заг} = \frac{R_В * R_Г}{R_В * \eta_c + R_Г * n_В * \eta_В} \quad (4.11)$$

Висновок: дане захисне заземлення буде забезпечувати електробезпеку будівлі, так як виконується умова: $R_{заг}$ Ом, а саме:

$$R_{заг} = \frac{18{,}5*8{,}1}{18{,}5*0{,}3+8{,}1*16*0{,}57} = 1{,}9 \le R_д.$$

ВИСНОВКИ

Після проведення аналізу предметної області було виконано порівняльний аналіз аналогічних алгоритмів. Розглянуто і проаналізовано існуючі алгоритми їх властивості та умови для їх використання. Описані бібліотеки, проаналізовані алгоритми за типом і вибором правильного для даного завдання.

Були сформульовані основні завдання, описані методи і способи їх вирішення, які були обрані для реалізації програми.

Складено технічне завдання на розробку з урахуванням недоліків аналогічних методів.

Для вирішення задачі проектування системи пошуку нерухомості на основі інтелектуального аналізу даних використовуються такі технології, як:

– C# (інтерфейс та програмування системи);

– Parser ();

– бібліотека HtmlAgilityPack (строит DOM дерево по HTML).

Взаємодія даних технологій допомагає створити онлайн-систему, за допомогою якої розраховуються кластери та обрати параметри для розрахунку за допомогою інтерфейсу, який робить систему більш корисною і практичною.

Були поставлені та вирішуються наступні завдання:

1) Вибір програмних засобів для розробки.
2) Огляд Методів добування інформації.
3) Огляд Алгоритмів.
4) Аналіз алгоритмів інтелектуального аналізу даних.
5) Порівняння алгоритмів.
6) Огляд .NET Framework.

В результаті проведеної роботи було зроблено аналіз умов праці, шкідливих та небезпечних чинників, з якими стикається робітник. Було визначено параметри і певні характеристики приміщення для роботи над запропонованим проектом, який описано в кваліфікаційній роботі, визначені заходи, які потрібно зробити для

того, щоб дане приміщення відповідало необхідним нормам і було комфортним і безпечним для робітника. Приведені рекомендації щодо організації робочого місця, а також важлива інформацію щодо пожежної та електробезпеки. Були наведені розміри приміщення, розраховано кількість ламп та захисне заземлення, а також – наведені інструкції з охорони праці, техніки безпеки при роботі на комп’ютері.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Воронцов К.В. Алгоритми кластеризації і багатовимірного шкалювання. Курс лекцій. МГУ, 2007.

2. Марченко А.Л. C#. Введение в программирование Учебное пособое Издательство Московского университета, 2005.

3. Інформаційно-аналітичний ресурс, присвячений машинному навчанню, розпізнаванню образів і інтелектуального аналізу даних – www.machinelearning.ru.

4. Котов А., Красильников Н. Кластеризація даних. 2006.

5. Мандель І. Д. Кластерний аналіз. – М .: Фінанси і Статистика, 1988.

6. Прикладна статистика: класифікація та зниження розмірності. / С.А. Айвазян, В.М. Бухштабер, І.С. Енюков, Л.Д. Мешалкин – М .: Фінанси і статистика, 1989.

7. Чубукова І.А. Курс лекцій "Data Mining", Інтернет-університет інформаційних технологій – www.intuit.ru/department/database/datamining.

8. Jain A., Murty M., Flynn P. Data Clustering: A Review. // ACM Computing Surveys. 1999. Vol. 31, no. 3.

9. Bourabai Research [Електронний ресурс] / Кластеризация – Режим доступу: http://bourabai.kz/tpoi/analysis6.htm

10. Bourabai Research [Електронний ресурс] / Parser – Режим доступу: http://bourabai.kz/alg/parser.htm

11. metanit [Електронний ресурс] / J Сериализация – Режим доступу: https://metanit.com/sharp/tutorial/6.1.php.

12. microsoft [Електронний ресурс] / Наследование – Режим доступу: https://docs.microsoft.com/ru-ru/dotnet/csharp/tutorials/inheritance.

13. metanit [Електронний ресурс] / Модели и БД – Режим доступу: https://metanit.com/sharp/tutorial/6.1.php.

14. metanit [Електронний ресурс] / Инкапсуляция – Режим доступу: https://metanit.com/sharp/tutorial/6.1.php.

15. metanit [Електронний ресурс] / Интерфейсы – Режим доступу: https://metanit.com/sharp/tutorial/6.1.php

16. wikipedia [Електронний ресурс] / Microsoft Visual Studio – Режим доступу: https://ru.wikipedia.org/wiki/Microsoft_Visual_Studio.

17. ДСН 3.3.6.042-99. Санітарні норми мікроклімату виробничих приміщень. Міністерство охорони здоров'я України (МОЗ). Постанова № 42 від 01.12.1999.

18. ГОСТ 12.1.030-81 ССБТ. Електробезпека. Захисне заземлення. Занулення.

19. ГОСТ 13109-97. Норми якості електричної енергії в системах електропостачання загального призначення.

20. ДСанПІН 3.3.2.007-98. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин. Міністерство охорони здоров'я України (МОЗ). Затверджено постановою № 7 головного державного санітарного лікаря України 10 грудня 1998 р.

21. НПАОП 0.00-1.28-10. Про погодження матеріалів правил охорони праці під час експлуатації електронно-обчислювальних машин. Державний комітет України з промислової безпеки, охорони праці та гірничого нагляду. Наказ №65 від 23.06.2010.

22. НАПБ Б.03.002-2007. Норми визначення категорій приміщень, будинків та зовнішніх установок за вибухопожежною та пожежною небезпекою. Наказ МНС № 833 від 03.12.2007 року.

23. ГОСТ 12.1.044-89. Система стандартів безпеки праці. Вогнестійкість. Номенклатура показників і методи їх визначення (ІСО 4589-84).

24. НПАОП 40.1-1.01-97. Правила безпечної експлуатації електроустановок. Наказ № 257 державного комітету України по нагляду за охороною праці від 6 жовтня 1997 р.

ДОДАТОК А

Лістинг коду DefaultDialogService.cs:

```
using System.Windows;
using Microsoft.Win32;

namespace BuildingsParser.FilesService
{
    public class DefaultDialogService : IDialogService
    {
        public string FilePath { get; set; }

        public bool OpenFileDialog()
        {
            OpenFileDialog openFileDialog = new OpenFileDialog();
            openFileDialog.Filter = "Xml files (*.xml)|*.xml|Text files
(*.txt)|*.txt|All files (*.*)|*.*";
            if (openFileDialog.ShowDialog() == true)
            {
                FilePath = openFileDialog.FileName;
                return true;
            }
            return false;
        }
        public bool SaveFileDialog()
        {
            SaveFileDialog saveFileDialog = new SaveFileDialog();
            saveFileDialog.Filter = "Xml file (*.xml)|*.xml";
            if (saveFileDialog.ShowDialog() == true)
            {
                FilePath = saveFileDialog.FileName;
                return true;
            }
            return false;
        }

        public void ShowMessage(string message)
        {
            MessageBox.Show(message);
        }
    }
}
```

Лістинг коду IDialogService.cs:

```
namespace BuildingsParser.FilesService
{
    public interface IDialogService
    {
        void ShowMessage(string message);
        string FilePath { get; set; }
        bool OpenFileDialog();
        bool SaveFileDialog();
    }
}
```

Лістинг коду IFileService.cs:

```
namespace BuildingsParser.FilesService
{
    public interface ISerialization<TModel> where TModel : class
    {
        //bool GetModel(string FilePath);
        //bool CreateModel(string FilePath);

        bool GetModel(string Path);
        bool CreateModel(string Path);
    }
}
```

Лістинг коду XMLFileService.cs:

```
using System.IO;
using System.Xml.Serialization;

namespace BuildingsParser.FilesService
{
    public class XMLSerialization<TModel> : ISerialization<TModel> where
TModel : class
    {
        public TModel _Model;
        public XMLSerialization() { }

        //public bool GetModel(string FilePath)
        //{
        //    return GetModel(FilePath, "");
        //}
        public bool GetModel(string Path)
        {
            using (FileStream fs = new FileStream(Path, FileMode.Open))
            {
                _Model                      =                      (TModel)new
XmlSerializer(typeof(TModel)).Deserialize(fs);
                return true;
            }
        }

        //public bool CreateModel(string FilePath)
        //{
        //    return CreateModel(FilePath, "");
        //}
        public bool CreateModel(string Path)
        {
            using (FileStream fs = new FileStream(Path, FileMode.Create))
            {
                new XmlSerializer(typeof(TModel)).Serialize(fs, _Model);
                return true;
            }
        }
    }
}
```

Лістинг коду ParserItem.cs:

```
namespace BuildingsParser.Models
{
    public class ParserItem
    {
        public string Name { get; set; }
        public string Description { get; set; }


        public string Region { get; set; }
        public int Area { get; set; }
        public string Furniture { get; set; }
        public string District { get; set; }
        public string Terrace { get; set; }
        public string Pool { get; set; }
        public string LotNumber { get; set; }
        public int Bedrooms { get; set; }
        public int Price { get; set; }

    }
}
```

Лістинг коду Parser.cs:

```
using System;
using System.Collections.Generic;
using BuildingsParser.Models;
using HtmlAgilityPack;

namespace BuildingsParser.ViewModel
{
    public class Parser
    {
        public List<ParserItem> Pars()
        {
            List<ParserItem> Items = new List<ParserItem>();
            HtmlDocument docHtml = new
HtmlWeb().Load("http://barcinohomes.com/poisk-obektov-po-karte");
            HtmlNodeCollection nodes =
docHtml.DocumentNode.SelectNodes("//ul");
            HtmlNodeCollection lists = nodes[0].SelectNodes("//li");

            foreach (HtmlNode list in lists)
            {
                if (list.ChildNodes.Count > 3)
                {
                    HtmlNodeCollection divs = list.ChildNodes;
                    if (divs[3].GetAttributeValue("class", null) == "item")
                    {
                        HtmlNodeCollection divs2 = divs[3].ChildNodes;
                        HtmlNodeCollection content = divs2[3].ChildNodes;
                        HtmlNodeCollection table = content[5].ChildNodes;
                        HtmlNodeCollection table1 = table[1].ChildNodes;
                        HtmlNodeCollection table2 = table[3].ChildNodes;
                        HtmlNodeCollection table3 = table[5].ChildNodes;

                        string splited1 =
```

```
table1[1].InnerText.Replace("Регион: ", "");

                        string[] splited2 = table1[3].InnerText.Split(' ');

                        string splited3 =
table1[5].InnerText.Replace("Мебель: ", "");

                        string splited4 = table2[1].InnerText;

                        string[] splited5 = table2[3].InnerText.Split(' ');

                        string splited6 =
table2[5].InnerText.Replace("Бассейн: ", "");

                        string splited7 = table3[1].InnerText.Replace("№
лота: ", "");

                        string splited8 =
table3[3].InnerText.Replace("Количество спален: ", "");

                        string splited9 =
table3[5].InnerText.Replace("Стоимость: ", "").Replace(" ", "").Replace("€",
"");

                        Items.Add(new ParserItem()
                        {
                            Name = content[1].InnerText,
                            Description = content[3].InnerText,
                            Region = splited1,
                            Area = Int32.Parse(splited2[1]),
                            Furniture = splited3,
                            District = splited4,
                            Terrace = splited5[1],
                            Pool = splited6,
                            LotNumber = splited7,
                            Bedrooms = Int32.Parse(splited8),
                            Price = Int32.Parse(splited9)
                        });
                    }
                }
            }
            return Items;
        }
    }
}
```

Лістинг коду MainWindow.xaml.cs:

```
using System.Collections.Generic;
using System.Linq;
using System.Windows;
using BuildingsParser.FilesService;
using BuildingsParser.Models;
using BuildingsParser.ViewModel;

namespace BuildingsParser
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
```

```
    public class Container
    {
        public List<ParserItem> Items = new List<ParserItem>();
    }
    public partial class MainWindow : Window
    {
        Container Items = new Container();

        DefaultDialogService dialogService = new DefaultDialogService();
        XMLSerialization<Container> fileService = new
XMLSerialization<Container>();


        private int Stan = 0;
        public MainWindow()
        {
            InitializeComponent();
            Parser Parser = new Parser();
            Items.Items = Parser.Pars();
            ParseItems();
        }


        public void ParseItems()
        {
            foreach (ParserItem item in Items.Items)
            {
                listBox1.Items.Add(item.Name + " " + item.Description + " "
+ item.Region + " " + item.Area + " " +
                                   item.Furniture + " " + item.District + "
" + item.Terrace + " " + item.Pool + " " +
                                   item.LotNumber + " " + item.Bedrooms + "
" + item.Price + " ");
            }
        }

        #region Buttons

        private void Button0_OnClick(object sender, RoutedEventArgs e)
        {
            listBox1.Items.Clear();
            if (Stan == 0)
            {
                SortArea();
            }
            else
            {
                SortAreaDesc();
            }
            ParseItems();
        }
        private void Button1_OnClick(object sender, RoutedEventArgs e)
        {

            listBox1.Items.Clear();
            if (Stan == 0)
            {
                SortBedrooms();
            }
            else
            {
                SortBedroomsDesc();
            }
            ParseItems();
```

```
        }
        private void Button2_OnClick(object sender, RoutedEventArgs e)
        {
            listBox1.Items.Clear();
            if (Stan == 0)
            {
                SortPrice();
            }
            else
            {
                SortPriceDesc();
            }
            ParseItems();
        }

        #endregion

        #region Sorts
        public void SortArea()
        {
            Stan = 1;
            Items.Items = Items.Items.OrderBy(i => i.Area).ToList();
        }
        public void SortAreaDesc()
        {
            Items.Items = Items.Items.OrderByDescending(i =>
i.Area).ToList();
            Stan = 0;
        }
        public void SortBedrooms()
        {
            Stan = 1;
            Items.Items = Items.Items.OrderBy(i => i.Bedrooms).ToList();
        }
        public void SortBedroomsDesc()
        {
            Items.Items = Items.Items.OrderByDescending(i =>
i.Bedrooms).ToList();
            Stan = 0;
        }
        public void SortPrice()
        {
            Stan = 1;
            Items.Items = Items.Items.OrderBy(i => i.Price).ToList();
        }
        public void SortPriceDesc()
        {
            Items.Items = Items.Items.OrderByDescending(i =>
i.Price).ToList();
            Stan = 0;

        }
        #endregion

        #region Menu Buttons

        private void MenuItem1_OnClick(object sender, RoutedEventArgs e)
        {
            if (dialogService.OpenFileDialog())
            {
                listBox1.Items.Clear();
                fileService.GetModel(dialogService.FilePath);
                Items = fileService._Model;
                ParseItems();
```

```
            }
        }

        private void MenuItem2_OnClick(object sender, RoutedEventArgs e)
        {
            if (dialogService.SaveFileDialog())
            {
                fileService._Model = Items;
                fileService.CreateModel(dialogService.FilePath);
            }
        }

        #endregion
    }
}
```

ДОДАТОК Б

Комп’ютерна презентація.

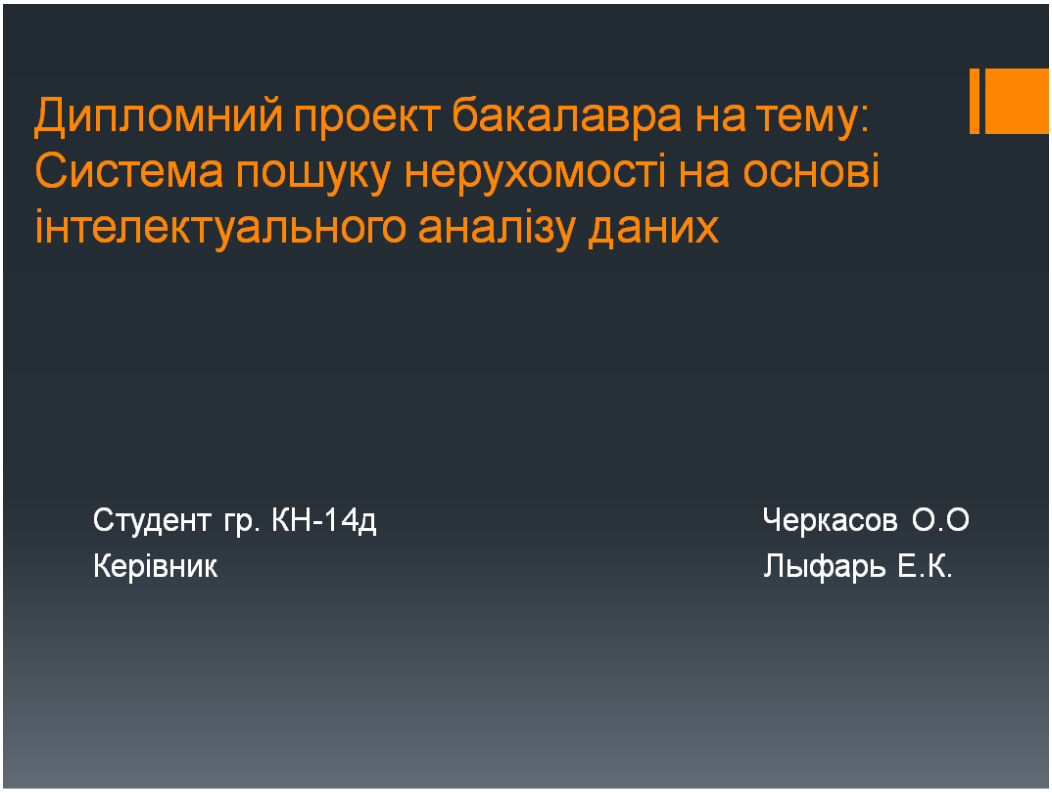

Рисунок Б.1 – Титульний лист

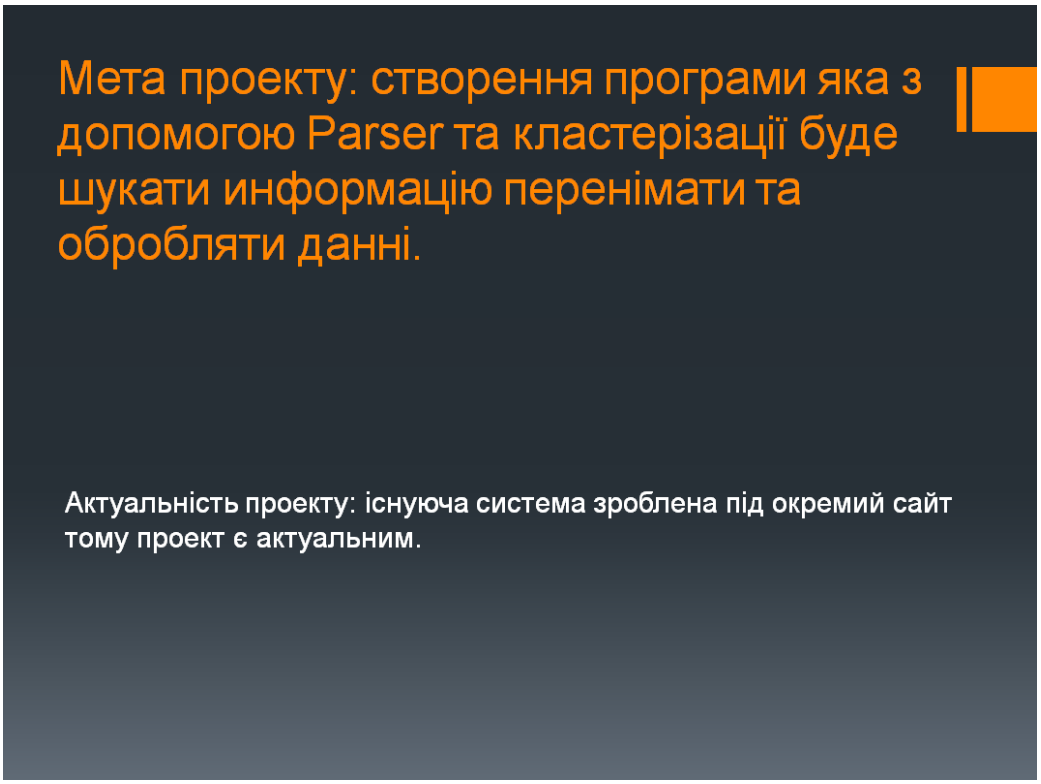

Рисунок Б.2 – Мета проекту

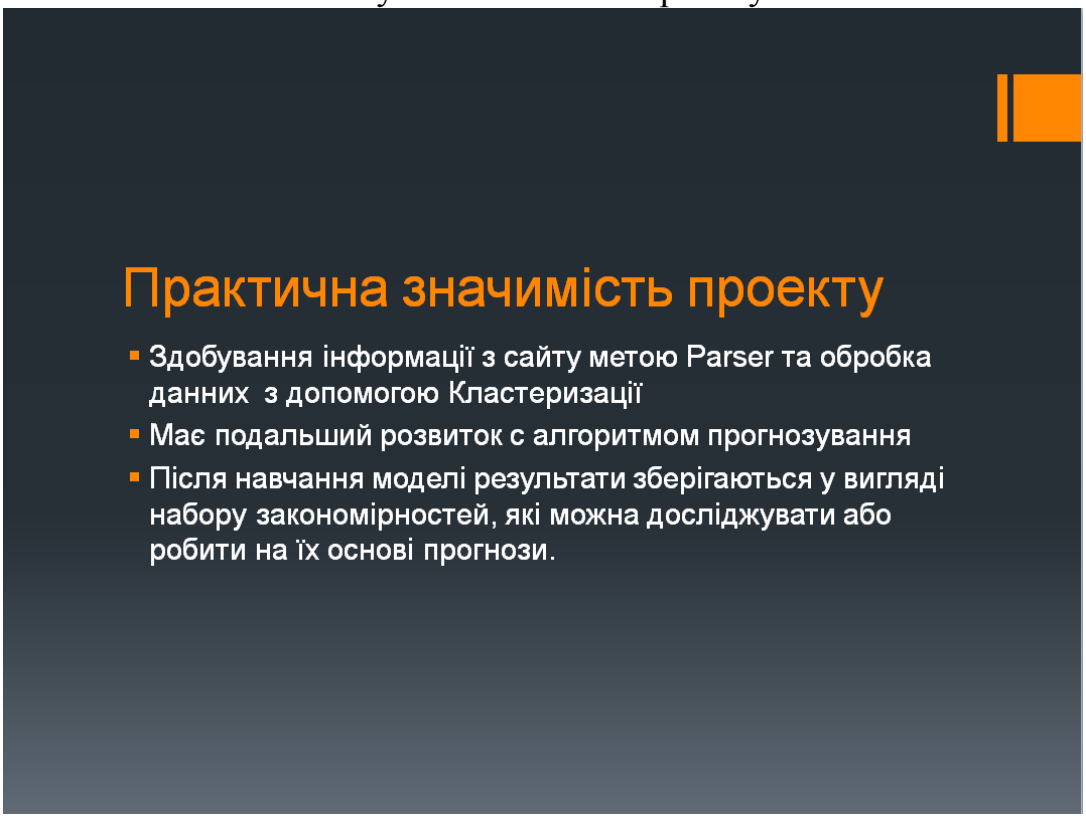

Рисунок Б.3 – Практична значимість проекту

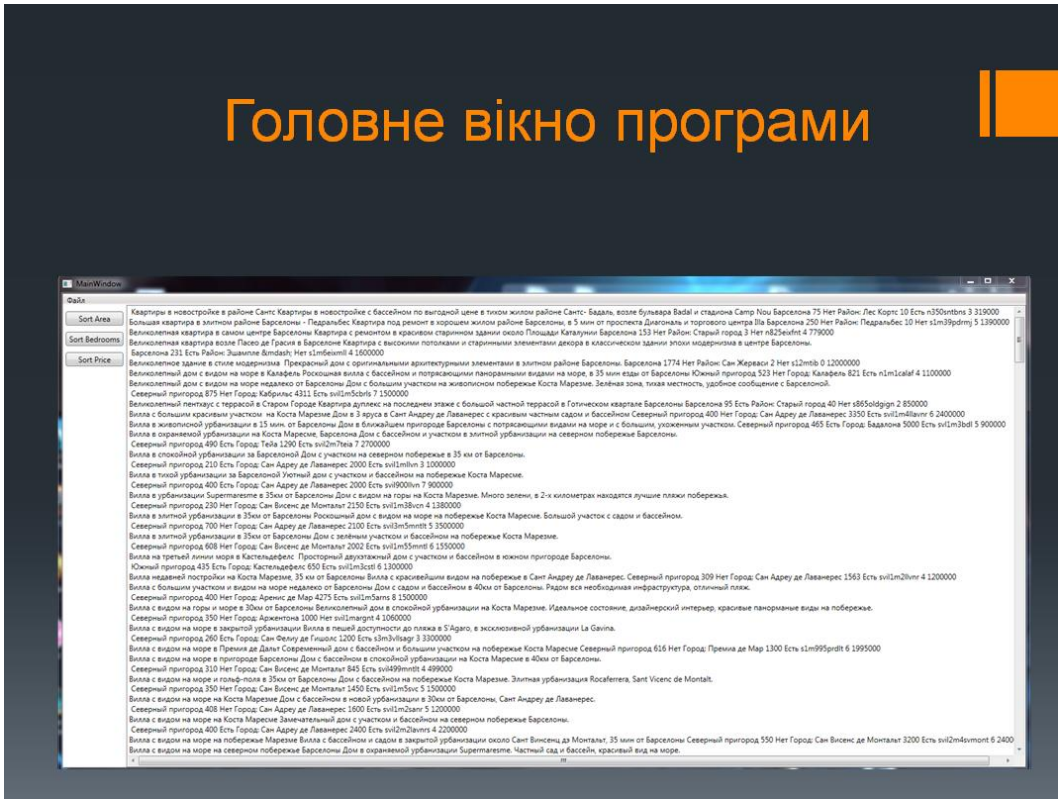

Рисунок Б.3 – Головне вікно програми

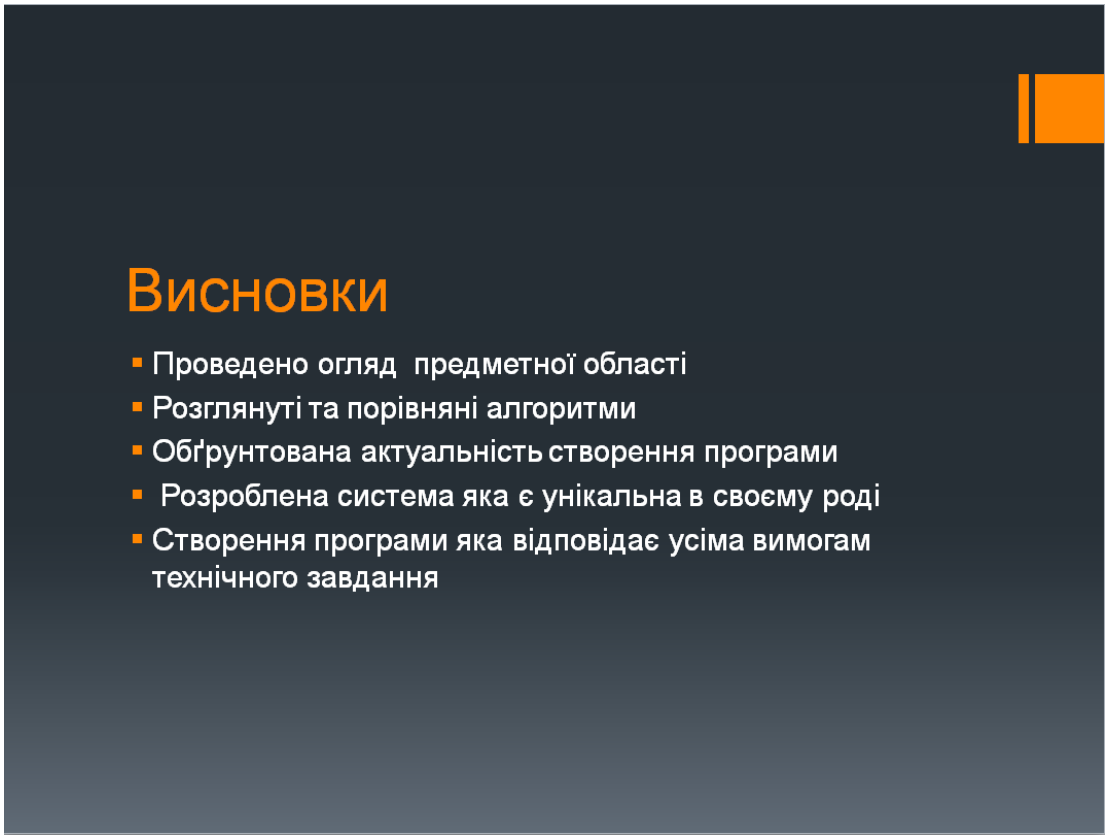

Рисунок Б.4 – Висновки

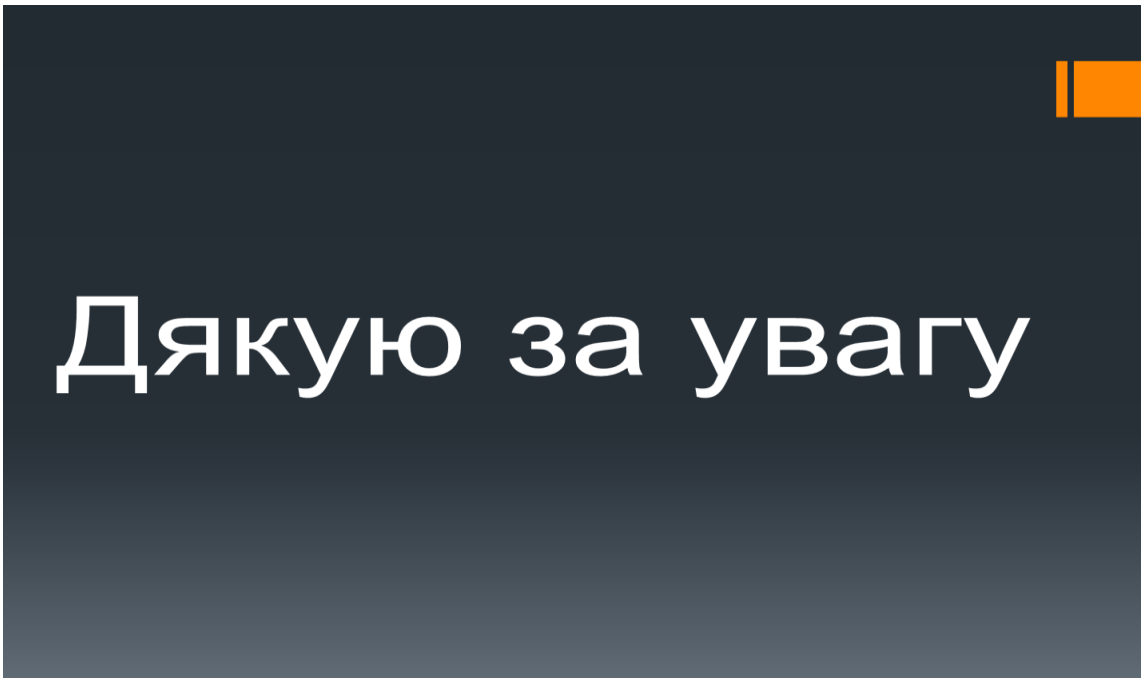

Рисунок Б.5 – Дякую за увагу