НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра автоматики та управління в технічних системах

«На правах рукопису»
УДК 681.03

«До захисту допущено»
В. о. завідувача кафедри
_____О.І. Ролік
«___»_____ 2018 р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 151 Автоматизація та комп'ютерно-інтегровані технології

на тему: «Засоби автоматизації самоналаштування паралельних програм для цільової платформи»

Виконав:
студент II курсу, групи ІА-61м
Старушик Артем Миколайович _____

Керівник:
Проф., д. ф.–м. н., проф.,
Дорошенко А. Ю. _____

Рецензент:
Доц., к. ф.–м. н., доц.,
Ігнатенко О. П. _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2018 року

РЕФЕРАТ

Магістерська дисертація на тему “Засоби автоматизації самоналаштування паралельних програм для цільової платформи” включає: __с., __ рис., __ табл., __ додатки, __ джерел.

Об'єкт дослідження – паралельні програми.

Мета роботи – автоматизація самоналаштування паралельних програм для цільової платформи.

Складні науково – технічні задачі потребують значних обчислювальних потужностей, тому оптимізація паралельних програм являється невід’ємною частиною їх процесу розробки.

Концепція автотюнінгу, останнім часом стала стандартом для вирішення задачі оптимізації паралельних програм. Її популярність зумовлена простотою застосування й незалежністю від якісних характеристик обчислювальної системи, оскільки зазвичай оптимізація виконується в цільовому обчислювальному середовищі, а отримана конфігурація паралельної програми залишається ефективною допоки не змінюються характеристики середовища виконання. Автоматизація даного процесу є наступним кроком в еволюції даного підходу.

Розроблений програмний комплекс для автоматизації самоналаштування паралельних програм з використанням однієї з найновітніших систем аналізу данних IBM Watson Analytics.

Прогнозні припущення про розвиток дослідження – покращення інтеграції, дослідження та використаннях нових можливостей системи аналізу данних.

АВТОТЮНІНГ, САМОНАЛАШТУВАННЯ, ПАРАЛЕЛЬНІ ПРОГРАМИ, IBM WATSON ANALYTICS, ОПТИМІЗАЦІЯ.

ABSTRACT

Master's thesis "Automation tools for self-adjustment of parallel programs for the target platform" consists of: __p., __ fig., __ tables, __ applications, __ sources.

Object of research - parallel programs.

The purpose of the work is to automate the self-configuration of parallel programs for the target platform.

Complex scientific and technical problems require significant computing power, therefore optimization of parallel programs is an integral part of their development process.

The autotuning concept has recently become the standard for solving the problem of optimizing parallel programs. Its popularity is due to the ease of use and independence from the qualitative characteristics of the computing system, since optimization is usually performed in the target computing environment, and the resulting configuration of the parallel program remains effective until the characteristics of the execution environment are changed. The automation of this process is the next step in the evolution of this approach.

A software package for automating self-adjusting parallel programs with one of the most advanced IBM Watson Analytics data analysis systems was developed.

Foreseeable assumptions about the development of the research are improving the integration, research and use of new features of the data analysis system.

AUTOTUNING, SELF-ADJUSTMENT, PARALLEL PROGRAMS, IBM WATSON ANALYTICS, OPTIMIZATION.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМОЛІВ, ОДИНИЦЬ, СКОЧЕНЬ І ТЕРМІНІВ

ORM – Object-Relational Mapping;

LINQ – Language-Integrated Query;

EDM – Entity Data Model;

IIS – Internet Information Server;

DNX – .NET Execution Environment;

SDK – Software Development Kit;

DTD – Document Type Definition;

IoC – Inversion of Control

ВСТУП

Паралельні обчислення — спосіб організації комп'ютерних обчислень, при якому програми розробляються, як набір взаємодіючих обчислювальних блоків, що працюють паралельно (одночасно) та являються декомпозицією певної задачі. Головною перевагою паралельних програм є можливість одночасного виконання певного набору операцій з метою досягнення високої продуктивності.

Незважаючи на загальновідомий закон Мура – емпіричне спостереження, зроблене Гордоном Муром, згідно якому потужність процесорів подвоюється кожні 18 - 24 місяці, реальна ситуація на поприщі паралельних обчислень є не настільки хорошою. Ефективність та продуктивність сучасних паралельних систем знаходиться нижче позначки в 10% від теоретично можливої пікової величини, відповідно до звіту Міжвідомчої комісії з розвитку надпотужних обчислень США.

Наявність потужної обчислювальної бази не гарантує факту ефективного її використання. Швидкодія паралельної системи значною мірою залежить від обраної архітектури, вдало розробленого алгоритму, правильної декомпозиції поставленої задачі та ряду інших факторів. Тому розробка продуктивного та ефективного паралельного рішення являється доволі нетривіальною задачею, тим паче в час масового використання багатоядерних мікропроцесорів.

Різноманіття та складність сучасних паралельних архітектур практично унеможливлює створення паралельних програм, ефективних на будь-якій із них. Кожен паралельний програмний комплекс потребує певної конфігурації для конкретного серидовища виконання для досягнення максимальної ефективності роботи.

Об’єктом досліджень є паралельні програми. Метою досліджень є автоматизація процесу самоналаштування паралельних програм. За результатами дослідження розроблений програмний комплекс, для автоматизації самоналаштування паралельних програм з використанням однієї з найновітніших систем аналізу данних IBM Watson Analytics.

Разультати досліджень детально висвітлені у вигляді публікацій у журналах «Проблеми прогамування» та «Winter Infocom 2017».

1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Паралельні обчислення

Ідея розпаралелювання заснована на тому, що більшість задач може бути розділено на деяку кількість інструкцій, які можуть виконуватись одночасно. Зазвичай паралельні обчислення вимагають чіткої координації дій та синхронізації інформації. Паралельні обчислення існують в декількох формах: паралелізм на рівні бітів, паралелізм на рівні інструкцій, паралелізм даних, паралелізм задач. Паралельні обчислення використовувалися багато років в основному в високопродуктивних обчисленнях, але останнім часом до них зріс інтерес внаслідок існування фізичних обмежень на зростання тактової частоти процесорів. Паралельні обчислення стали домінуючою парадигмою в архітектурі комп'ютерів та комп'ютерних систем та стали невід'ємною частиною усіх галузей програмування.

В свою чергу архітектура паралельних програм набагато складніша ніж аналогічних їм послідовних. Конкуренція за ресурси, доступ до даних, синхронізація результатів виконання породжує нові класи потенційних помилок в програмному забезпеченні, серед яких стан «гонки» є найпоширенішою. Взаємодія і синхронізація між процесами представляють великий бар'єр для отримання високої продуктивності паралельних систем.

Характер збільшення швидкості програми в результаті розпаралелювання пояснюється законом Амдала (рисунок 1.1): прискорення програми при розпаралелюванні залежить від того яку частину програми можна виконувати паралельно. Наприклад, якщо 90% програми можна виконувати паралельно, теоретичним максимумом прискорення при запуску її на паралельній машині буде десятикратне, незалежно від того, скільки процесорів використовується.

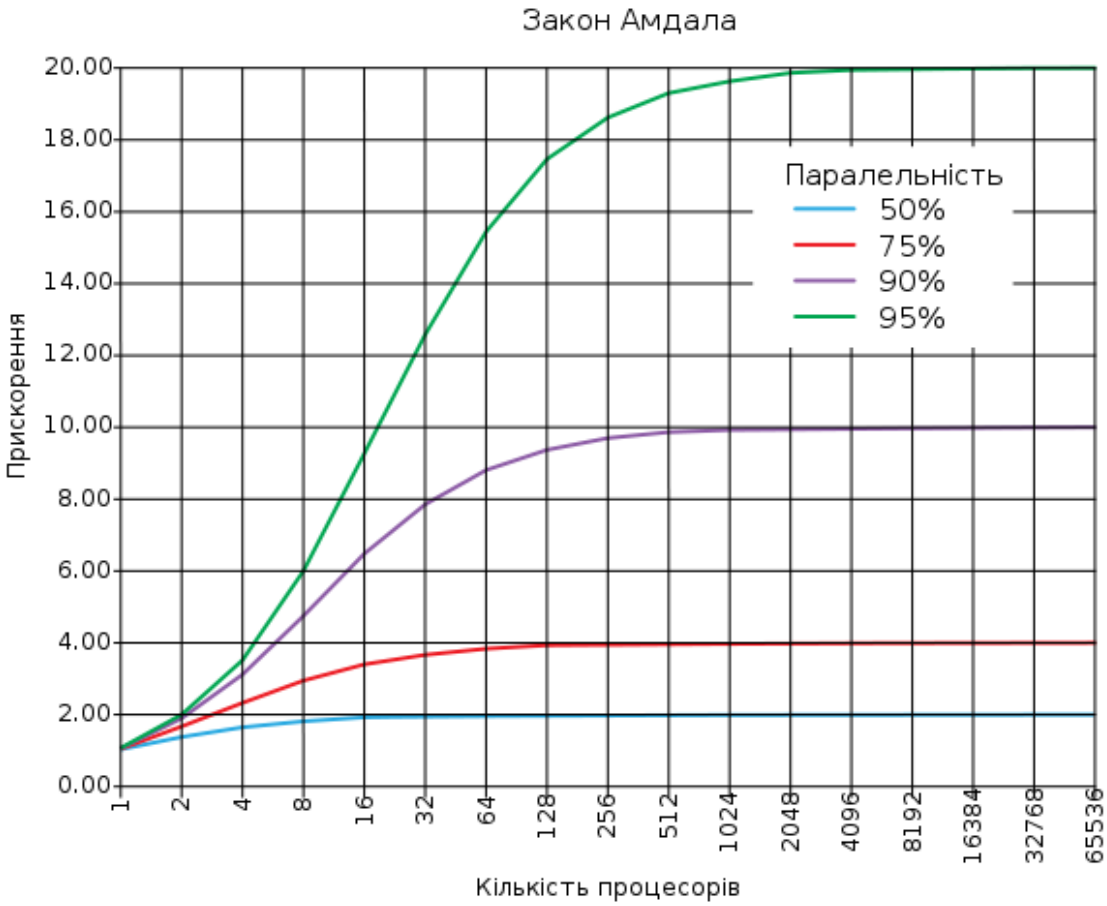

Рисунок 1.1 – Графічне зображення закону Амадала

Паралельні обчислення також можуть бути застосовані до проектування відмовостійких обчислювальних систем, зокрема, за допомогою Lockstep системи, що виконують ті ж операції паралельно. Це забезпечує надлишковість у разі падіння одного з компонентів і також дозволяє автоматично визначати та виправляти помилки якщо результати різняться. Ці методи можуть бути використані, щоб допомогти запобігти поодиноким подіям, які викликають тимчасові помилки. Хоча додаткові заходи можуть знадобитися у вбудованих або спеціалізованих системах, цей метод може забезпечити економічно ефективний підхід до досягнення n-модульної надмірності в комерційних системах.

1.2 Опитмізація паралельних програм

Складні науково – технічні задачі потребують значних обчислювальних потужностей, тому оптимізація паралельних програм являється невід'ємною частиною їх процесу розробки.

Царину методів оптимізації паралельних програм умовно можна поділити на дві групи: статичні та динамічні оптимізатори.

Статичні оптимізатори[1] – це додаткове програмне забезпечення та спеціалізовані компілятори, які мають певну експертну базу правил, на основі якої проводиться статичний аналіз вихідного коду. Перевагами такого підходу є відносна простота та швидкість, оскільки виконується процес послідовного аналізу вихідної програми та заміни певних структур, на аналогічні оптимізовані структури описані в експертній базі правил. Основним недоліком являється те, що такий підхід працює лише з вихідним кодом як таким, не враховуючи специфіку робочого алгоритму, середовища виконання, показники якості та точності результату. Складність та архітектурні особливості сучасних обчислювальних систем значно ускладнюють розвиток даного підходу.

Динамічна оптимізація – дозволяє емпіричним методом, в автоматичному режимі визначити найкращу конфігурацію для паралельної програми та впровадити цю конфігурацію на рівні вихідного коду. Даний підхід потребує додаткового аналізу та ресурсів для імплементації та подальшої роботи.

1.3 Концепція автотюнінгу

Концепція автотюнінгу[2], останнім часом стала стандартом для вирішення задачі оптимізації паралельних програм. Її популярність зумовлена простотою застосування й незалежністю від якісних характеристик обчислювальної

системи, оскільки зазвичай оптимізація виконується в цільовому обчислювальному середовищі, а отримана конфігурація паралельної програми залишається ефективною допоки не змінюються характеристики середовища виконання. Такий підхід дозволяє абстрагуватися від конкретоного обчислювального середовища на етапі створення паралельної системи та в подальшому оптимізувати її для цільової платформи.

Звичайно повністю автоматизувати процес автотюнінгу неможливо із-за декількох факторів: варіанти програми задаються розробником, оскільки він являється експертом в даній предметній області та знає які характеристики програми можуть значно вплинути на оптимізацію; тільки розробник може правильно оцінити роботу автотюнера, оскільки інколи важливимявляється не тільки час виконання, а й точність результату, використані ресурси, трафік тощо.

Автотюнінг не слід плутати з автоматичним розпаралелюванням. Автотюнінг не розпаралелює, а впливає на існуючу паралельну програму та досліджує різні компроміси серед паралелізму, синхронізації, балансування навантаження, локалізації та інших параметрів.

Перш ніж автоматична настройка стала популярною, багато методів оптимізації використовувались за фіксованою стратегією, яка брала за основу модель продуктивності для управління середовищем перетворення або бібліотекою виконання, щоб вибрати конкретні стратегії. Експерименти показали, що ці фіксовані стратегії часто можуть бути перевершені шляхом вивчення більшого простору пошуку перетворень та настроюваних параметрів. Початком ери автотюнінгу можна вважати кінець шістдесятих років, коли такі дослідники, як Девід Сайр (IBM) та Доменіко Феррарі (Університет Берклі), почали вивчати оптимізацію компіляторів для покращення різних аспектів виконання програм. З того часу астотюнінг став добре налагодженою технікою, яка постійно вдосконалювалася для автоматизації оптимізації програм.

Астотюнінг може відбуватися як фаза оффлайн налаштування, як фаза онлайн налаштування в рантаймі виконуваної прогарми, або ж в обох випадках.

Оффлайн налаштування означає, що процес астотюнінгу відбувається як незалежна частина роботи системи під час того періоду часу коли система не активна. Перевагою даного підходу є те, що автотюнінг не впливає на роботу системи.

Онлайн налаштування означає, що процес автотюнінгу відбувається прямо під час роботи досліджуваної системи. Перевагою даного підходу є те, що налаштування проходить відносно рельного продуктивного навантаження, що дозволяє отримати найбільш точні результати для конкретної системи.

Концепція автотюнінгу уже довела свою універсальність й ефективність, проте вона не позбавлена недоліків: необхідність використання автотюнера, значний час роботи автотюнера, інтеграція термів автотюнера у вихідний код програми, аналіз великої кількості статистичних даних, якими є результат роботи автотюнера.

Особливістю автотюнінгу є те, що найкраща конфігурація програми визначається емпіричним методом, на основі статистичних даних отриманих автотюнером. Конфігурація – набір параметрів у вихідному коді програми, які тією чи іншою мірою впливають на продуктивність паралельної програми (кількість потоків, ітерацій, величина розбиття тощо). Автотюнер формує проміжні конфігурації на основі заданих правил та оцінює їх ефективність в конкретному обчислювальному серидовищі. На основі аналізу отриманих статистичних даних формується найкраща конфігурація для паралельної програми, яка вводиться на рівні вихідного коду.

В цілому алгоритм дії автотюнера можна розбити на наступні кроки:

- виділення та аналіз метаданих з вихідного коду;

- формування, на основі виділених метаданих, множини конфігурацій для вихідної програми, що являє собою сукупність параметрів які тією чи іншою мірою

впливають на продуктивність паралельної програми. Кожна конфігурація являє собою унікальну трансформацію вихідного коду. В результаті якої генерується новий варіант програми;

- емпіричний аналіз ефективності отриманих конфігурацій;

- визначення найбільш ефективної конфігурації.

У загальному випадку кількість ітерацій роботи автотюнера дорівнює кількості виділених конфігурацій, оскільки перевіреною має бути кожна з них, для подальшого емпіричного аналізу. Тобто час роботи автотюнера прямомпропорційний кількості виділених конфігурацій, що не завжди зручно в умовах обмежених ресурсів. В такому випадку раціонально ввести певні часові обмеження, по закінченню яких ітерації автотюнера закінчуються та проводиться емпіричний аналіз отриманих результатів. Також, одним із підходів до оптимізації роботи автотюнера є використання спеціалізованих пошукових алгоритмів[3], що вибиратимуть наступну конфігурацію для перевірки. Інтелектуальне задання конфігурацій може значно зменшити множину, але потребує додаткового аналізу на етапі інтеграції автотюнера. Дані підходи значно зменшують час роботи роботи автотюнера, але в загальному випадку потребують додаткових ресурсів для імплементації.

Процес емпіричного аналізу ефективності виділених конфігурацій також являється нетривіальною задачею та може займати значний час. Велика кількість параметрів та, як результат, конфігурацій підвищують складність аналізу, виділення трендів та кореляцій з характеристиками програми. Тому для аналізу отриманих автотюнером результатів часто використовують окремі програмні комплекси, що дозволяють знаходити статистичні залежності між великою кількістю параметрів, так звана гібридна модель автотюнінгу[4].

2 ІСНУЮЧІ РІШЕННЯ ДЛЯ АНАЛІЗУ ДАННИХ

2.1 Студія машинного навчання Azure ML

Студія машинного навчання Microsoft Azure (Azure Machine Learning[5], Azure ML) - це інструмент для створення, тестування і розгортання рішень для прогнозного аналізу даних. Студія машинного навчання публікує моделі як веб-служби, які потім можна використовувати в призначених для користувача додатках і засобах бізнес-аналітики.

Для розробки моделі прогнозної аналітики зазвичай використовуються дані з одного або декількох джерел. Для отримання набору результатів ці дані перетворюються і аналізуються за допомогою різних операцій і статистичних функцій. Така розробка моделі - це ітеративний процес. Шляхом зміни різних функцій і їх параметрів виконується зведення результатів, поки не буде отримана підготовлена і ефективна модель.

Студія машинного навчання Azure надає інтерактивне візуальне робочий простір, що спрощує створення, тестування і виконання ітерацій моделі прогнозної аналітики. Взаємодія з Azure ML, зображена на рисунку 2.1 рисунок 2.1) відбувається наступним чином: користувач завантажує набори даних і модулі аналізу на інтерактивний полотно і пов'язуєте їх разом, щоб створити експеримент (рисунок 2.2), який потім виконується. Для виконання ітерацій в макеті моделі слід відредагувати експеримент, при необхідності зберегти копію і виконати його знову. Коли ви будете готові, навчальний експеримент можна перетворити в прогнозний, а потім опублікувати його як веб-службу, щоб модель стала доступна іншим користувачам.

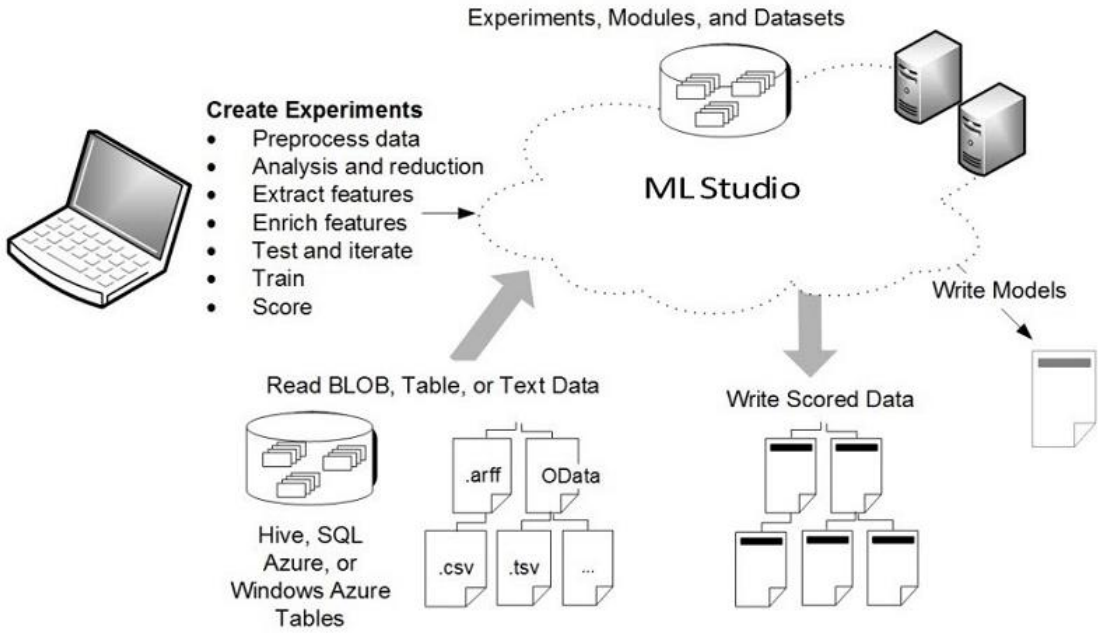

Рисунок 2.1 – Принцип взаємодії з Azure ML

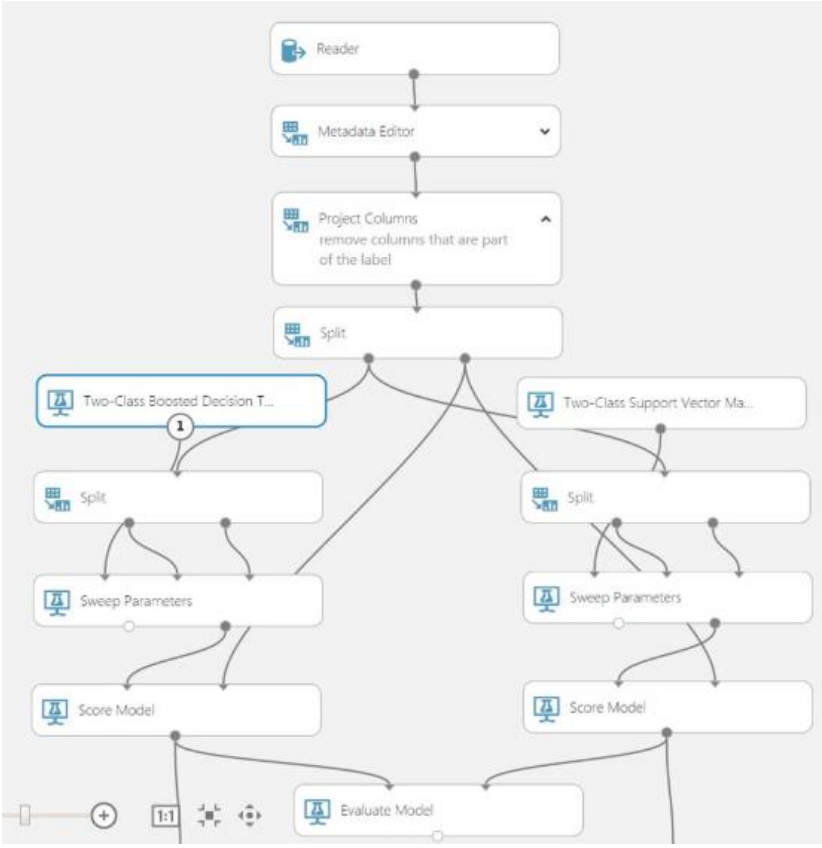

Рисунок 2.2 – Експеримент в Azure ML

Перевагами даної системи є: інтуїтивно зрозумілий інтерфейс, доступність, веб-розміщення та можливість доступу з будь якого комп'ютера підключеного до інтернету, швидкодія за рахунок спеціалізованих серверів Azure Analytics, доступ до можливостей платформи Azure. Серед основних недоліків можна виділити те, що для ефективної роботи з даною системою користувачу потрібно мати базові знання алгоритмів, структур даних та понять машинного навчання. Тому дана система має більш вузьке коло користувачів в порівнянні з іншими.

2.2 Google Data Studio

Google Data Studio - інструмент від компанії Google який дозволяє накопичувати, аналізувати та візуалізувати дані. Спеціалісти Google виділяють три основні кроки взаємодії з даним сервісом зображених на рисунку 2.3:

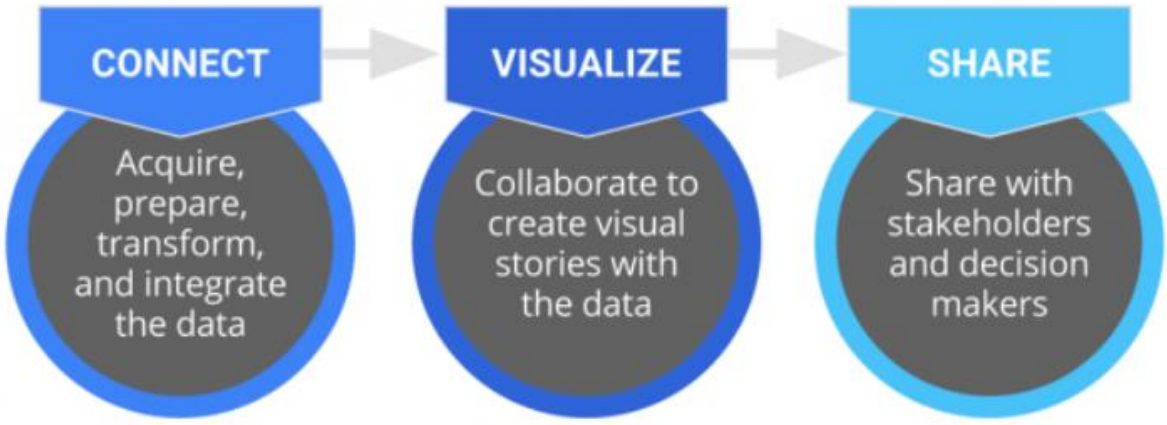

Рисунок 2.3 – Принцип взаємодії з Google Data Studio

- Connect – збір даних, який включє в себе як сторонні джерела даних, так і дані з внутрішніх сервісів Google (Google Analytics, Google Sheets, тощо) і при необхідності підготовка та форматування даних;

- Visualize – візуалізація та аналіз даних. З допомогою спеціалізованого програмного забезпечення відбувається аналіз отриманих даних на основі якого повертається відповідна інформація. Дані та результати аналізу можуть бути візуалізовані у вигляді таблиц, графікі, діаграм, тощо. Нововведенням є інтеграція з картами Google Maps, що дозволяє накладати дані на карту на основі потоку даних з інших джерел. Екосистема Google дозволяє використовуати Google Sheets, Google Presentation та інші сервіси для експорту ориманих даних;

- Share – поширення результатів у просторі екосистеми даних, для можливості подільшої обробки та передачі.

Google Data Studio має інтуїтивно зрозумілий інтерфейс зображений на рисунку 2.4 та являється лекгим в освоєнні інструменом для аналізу даних.

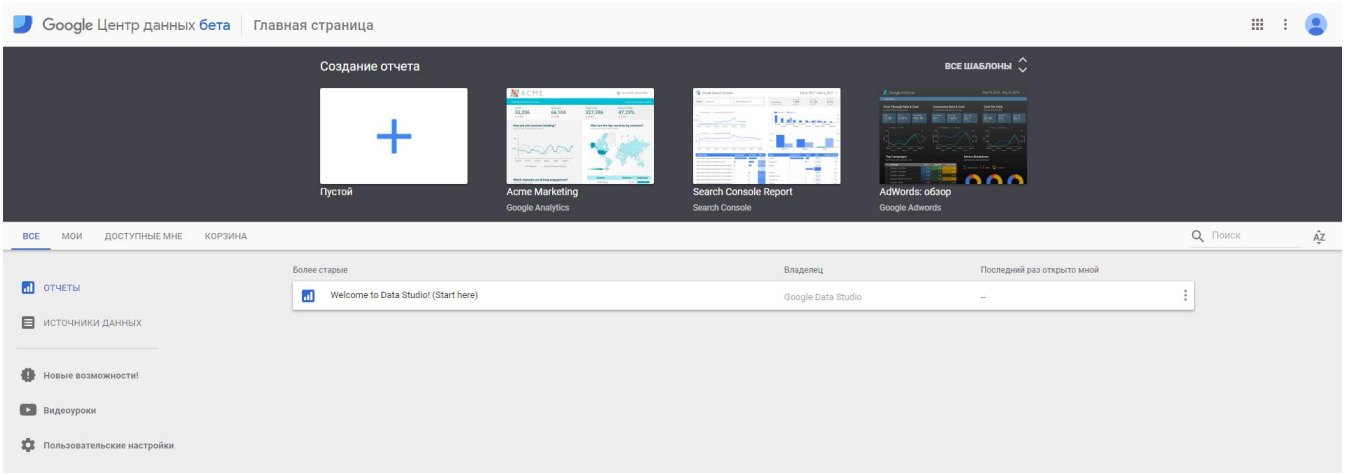

Рисунок 2.4 – Інтерфейс Google Data Studio

Перевагами даної системи є: інтуїтивно зрозумілий інтерфейс, доступність, веб-розміщення та можливість доступу з будь якого комп'ютера підключеного до інтернету, чудова екосистема Google, що забезпечить просту

інтеграцію, імпорт, експорт та передачу даних. Серед основних недоліків можна виділити: відносно малий функціонал в порівнянні з іншими подібними рішеннями, направленість в основному на бізнес сферу, відсутність постійної аудиторії, оскільки на даний момент система знаходиться в бета версії.

2.3 IBM Watson Analytics

Інструмент для аналізу даних від компанії IBM під назвою IBM Watson Analytics[6] часто називають когнітивною обчислювальною системою. Цей розумний data science-помічник виступає одночасно в якості інструменту і путівника для користувачів з різними рівнями навичок в data science. Працювати з ним в однаковій мірі можуть як і початківці, так і спеціалісти у даній сфері. Спеціаліст IBM виділяють три основні функції системи зображених на рисунку 2.5:

Рисунок 2.5 – Принцип взаємодії з Google Data Studio

- Explore – функціящо, дозволяє робити запити до даних. Для цього можна скористатися наявними шаблонами або ввести текст вручну. Запити можна формувати, як у вигляді формул так і у вигляді звичайного питяння у вигляді «Як змінна А залежить від змінної Б ?»

- Predict – функція, що дозволяє передбачати одну і більше змінні на основі інших змінних та їх залежностей. Для цього застосовується метод класифікації або регресії в залежності від того, змінна категоріальна або безперервна. 2.4 Порівняння та вибір системи аналізу даних

- Assemble – функція, що дозволяє створювати робочі журнали, внутрішні їєрархії даних, що містять презентаційні матеріали, візуалізації даних і звіти.

IBM Watson Analytics має інтуїтивно зрозумілий інтерфейс зоюражений на рисунку 2.6 та являється лекгим в освоєнні інструменом для аналізу даних.

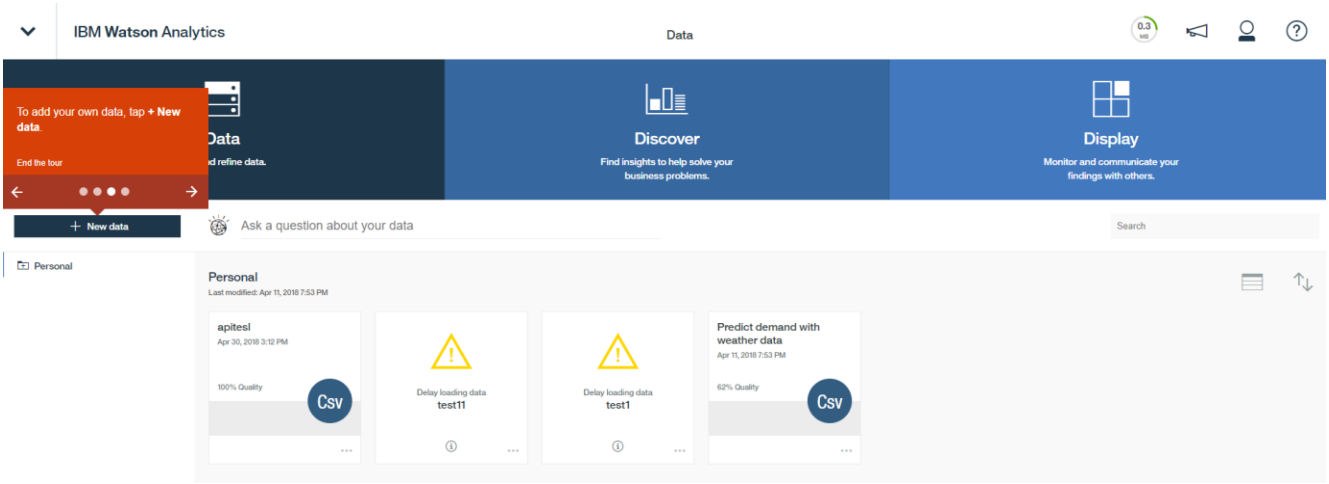

Рисунок 2.6 – Інтерфейс IBM Watson Analytics

Перевагами даної системи є: інтуїтивно зрозумілий інтерфейс, доступність, веб-розміщення та можливість доступу з будь якого комп'ютера підключеного до інтернету, ефективне використання користувачами з різними рівнями знань у сфері аналізу даних, когнітивні навички системи, що дозволяють формулювати питання у звичайному вигляді. Серед основних недоліків є

відсутність глобальної екосистеми, що робить дану систему самостійною та потребує створювати додаткові інструменти для завантаження, експорту та передачі даних.

2.4 Порівння на вибір системи аналізу даних

Хоча усі розглянуті системи є потужними інструментами для аналізу і обробки даних, все ж у них зовсім різні цільові аудиторії - принаймні, поки що. IBM Watson Analytics має більш широку аудиторію як спеціалістів, так і новачків у сфері аналізу даних.

Azure ML, в свою чергу, у більшому степені інтегрований з класичними техніками Data Science і передбачає знання статистики та базових алгоритмів машинного навчання та структур даних.

Також можна сказати, що інструменти Microsoft та Google набагато слабші в плані автоматичного аналізу та представлення результатів.

На основі вищеописаних характеристика та порівняння визначених систем, для подальшої роботи було обрано систему IBM Watson Analytics.

3 ОСОБЛИВОСТІ СИСТЕМИ IBM Watson Analytics

3.1 Суперкомп’ютер IBM Watson

IBM Watson - це суперкомпютер компанії IBM розроблений в рамках проекту IBM DeepQA дослідницькою групою на чолі з головним дослідником Девідом Ферруччі. Watson був названий на честь першого виконавчого директора IBM Томаса Дж. Уотсона.

IBM Watson був створений, як комп'ютерна система, що відповідає на запитання (question-answering system, QA), яку компанія IBM створила для застосування сучасних методів обробки природної мови, пошуку інформації, представлення знань, автоматичного обґрунтування та технологій машинного навчання до відповіді на питання.

Основна відмінність технології QA від пошуку документа полягає в тому, що при пошуці документа виконується запит на ключове слово та повертається список документів, класифікованих у певній послідовності відповідно до запиту (часто заснований на популярності та рейтингу сторінок), тоді як технологія QA приймає питання природною мовою, намагається зрозуміти його детальніше та повертає точну відповідь на це питання.

При створенні суперкомп'ютера компанія IBM заявила, що понад 100 різних методів використовуються для аналізу природної мови, виявлення джерел, пошуку та створення гіпотез, пошуку та оцінки доказів, а також об'єднання та ранжування гіпотез.

В останні роки можливості IBM Watson були розширені, і методи роботи з ним були змінені, задля того, щоб скористатися перевагами нових моделей розгортання (Watson на IBM Cloud), а також розвинути можливості машинного навчання та оптимізували апаратне забезпечення, доступне розробникам та дослідникам. Це вже не просто QA-система, здатна відповідати на поставлені

питання. Тепер Watson може бачити, чути, читати, говорити, розуміти, інтерпретувати, вчитися та рекомендувати.

IBM Watson використовує програмне забезпечення IBM DeepQA, високорівнева структура якого зображена на рисунку 3.1 та архітектуру Apache UIMA (Unstructured Information Management Architecture). Система була написана різними мовами, включаючи Java, C ++ і Prolog, і працює на операційній системі SUSE Linux Enterprise Server 11, використовуючи структуру Apache Hadoop для забезпечення розподілених обчислень.

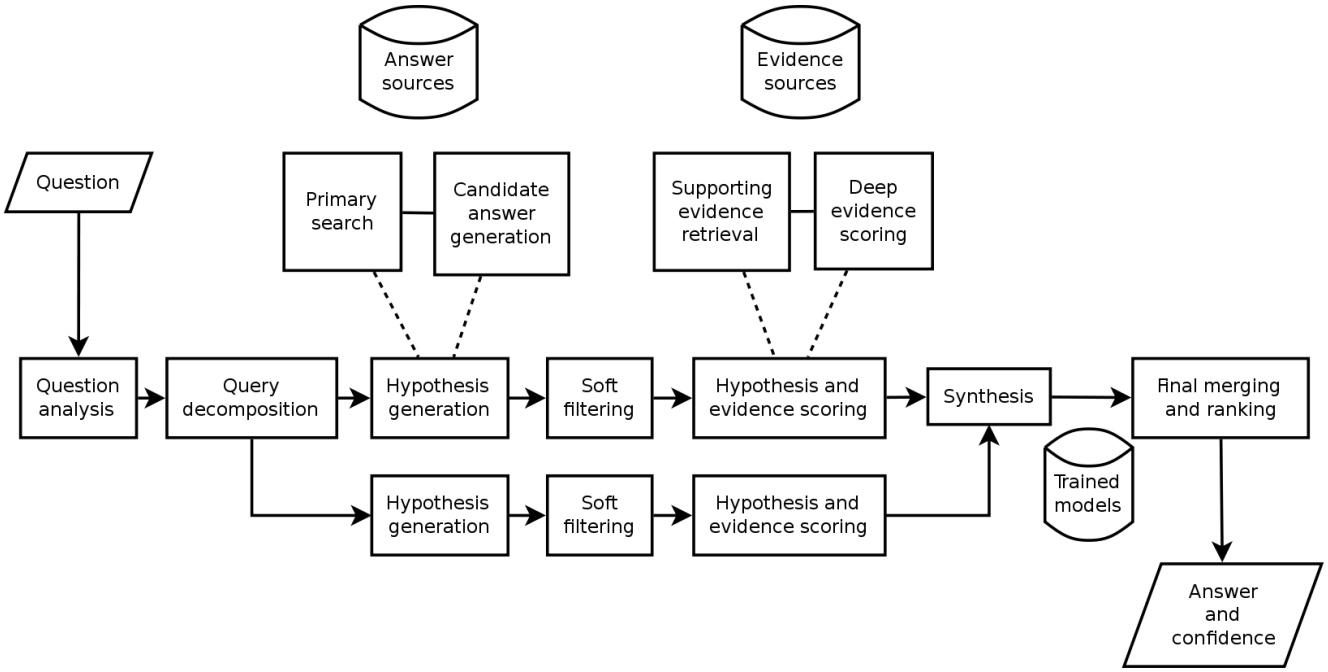

Рисунок 3.1 Високорівнева архітектура IBM DeepQA

Суперкомп'ютер Watson працює зі швидкістю 80 терафлопс (трильйонів операцій з плаваючою комою в секунду). Щоб відтворити (або перевищити) здатність людини працювати на відповідь на запитання, Watson має доступ до 90 серверів із об'єднаним сховищем даних понад 200 мільйонів сторінок інформації, які обробляються шістьма мільйонами логічних правил.

3.2 IBM Watson Analytics

Проблема пошуку прихованих закономірностей та взаємозв’язків - відома і за межами машинного навчання. За останні 20 років було створенно багато методологій інтелектуального аналізу данних і згідно з опитуванням[7], 43% спеціалістів цієї області використовують методологію CRISP-DM, в той час як частка всіх інших методологій складає 30% (27% припадає на власні методології). І хоча у відомих методологій існують відмінності, загалом процес пошуку данних можно представити у такому вигляді:

1) Постановка цілі
2) Початковий аналіз даних
3) Підготовка даних
4) Моделюваня
5) Оцінка
6) Використання результатів

IBM Watson Analytics – це інтелектуальна служба аналізу та візуалізації даних, яка дозволяє користувачам самостійно та швидко виявляти явні та приховані закономірності введених даних. Завдяки аналізу структури даних, когнітивним можливостям да відкритому API користувачі можуть вільно працювати з даними тра отримувати необхідні результати.

Система IBM Watson Analytics виникла як логічне продовження розвитку суперком’пютера IBM Watson, та надаоє доступ до його можливостей будь кому, і по своїй суті являється проміжною лакомю між суперкомп’ютером та користувачем.

IBM Watson Analyticsявляє собою аналітичний інструмент який створювався для того, щоб максимально автоматизути проце аналізу даних, в саме: початковий аналіз даних, підготовку даних та моделювання. Процес роботи с IBM Watson Analytics виглядає так:

1) Вибір джерела даних. В якості джерела можно використовувати як данні з електронної таблиці, так і автоматично імпортувати данні з сторонніх джерел (наприклад, з Twitter чи Salesforce).

2) Дослідження. За допомогою навідних запитань про завантажені данні, Ватсон домогоє візуалізувати данні для подальшого аналізу.

3) Прогнозування. IBM Watson Analytics у автоматичному режимі виділяє основні корреляції в данних, та дає можливість користувачу уточнювати їх за допомогою питань на природній мові.

4) Монтування. Система надає користувачам можливість сберегти обработані данні у вигляді наглядних інфографіків для подальшої презентації.

4 ТЕХНОЛОГІЧНА БАЗА РОЗРОБЛЕНОЇ СИСТЕМИ

Система розроблена з використанням мова програмування C# та найновішої версії технологічного стеку веб-розробки Microsoft : ASP.Net Core (MVC та WebAPI), Entity Framework 7. Розробка клієнтської частини включає використання мови програмування JavaScript, мови розмітки гіпертексту HTML, та каскадних таблиць стилей CSS.

4.1 Основні відомості про мову програмування C#

На сьогоднішній момент мова програмування C# одна із найпотужніших та найбільш затребуваних мов програмування у сфері інформаційних технологій. Дана мова дозволяє створювати найрізноманітніші проекти: від простих мобільних додатків, до високонавантажених веб-порталів.

C# відноситься до сімейства мов із C-подібним синтаксисом, тому синтаксично схожий на такі відомі мови, як Java та C++. Це об’єктно-орієнтовна мова програмування з відносно строгою типізацією, оскільки в останніх версіях присутні деякі можливості динамічної типізації. Підтримує перевантаження операторів, делегати, події, анонімні функції, замикання, атрибути, узагальнені типи та методи, ітератори.

Перейнявши багато у своїх попередників – мов C, C++, Smalltalk, Java, опираючись на різні методики та підходи до розробки, ця мова перейняла краще та виключила деякі моделі, які зарекомендували себе як проблематичні, наприклад, в C#, на відміну від C++, не підтримується багаторазове наслідування класів, проте присутнє багаторазове наслідування інтерфейсів.

C# продовжує активно розвиватись. З кожною новою версією з’являється все більше цікавих можливостей, як, наприклад, асинхронні методи, лямбда вирази, динамічне зв’язування тощо.

4.2 Особливості платформи .NET Core

.NET Core – це модульна версія .NET Framework з можливістю переносу на інші платформи. Вона є підмножиною повної версії .NET Framework, та надає ключові можливості для реалізації функцій додатків, необхідних для повторного використання цього коду незалежно від цільової платформи.

Тобто раніше структура середовища .Net мала вигляд, зображений на рисунку 4.1, де кожна цільова платформа мала свою, окрему вертикаль: свою модель додатку, функціональну базу та середовище виконання.

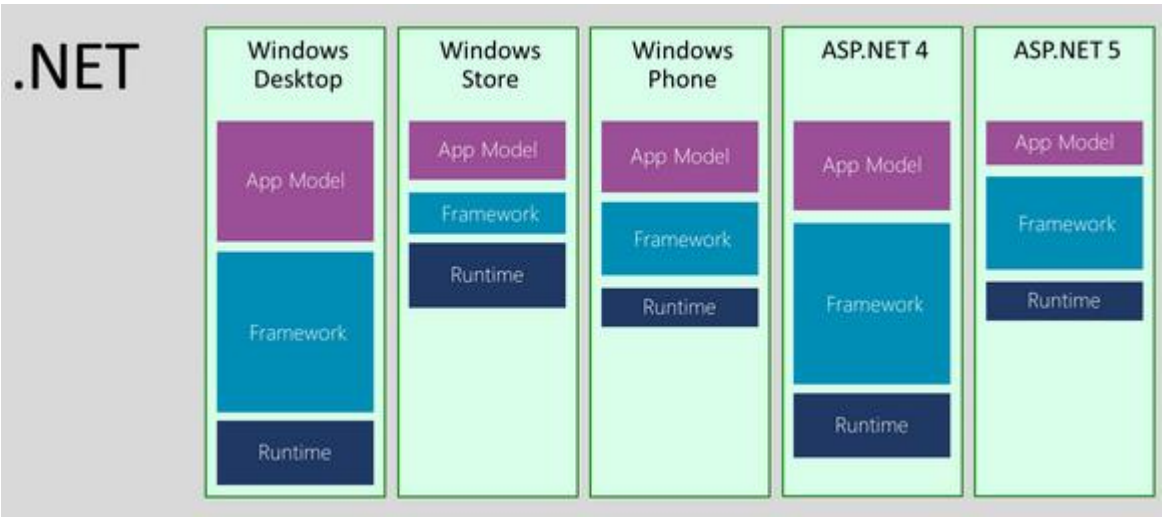

Рисунок 4.1 – Структура середовища .NET Framework

Якщо розробник націлений на одну вертикаль із даної структури то ніяких проблем не виникає. Кожна вертикаль надає набір API спеціально оптимізований для її функціональності. Проблема з’являється тоді, коли розробник хоче охопити

декілька вертикалей, оскільки з’являється питання написання коду, який би успішно виконувався на різних цільових вертикалях.

Це сприяло переосмисленню структури середовища .Net і, як наслідок, появі .Net Core. Нова структура зображена на рисунку 4.2 дозволяє обійти недоліки попередньої версії, оскільки .NET Core - це модульна реалізація, яка може використовуватися широким набором вертикалей, починаючи з дата-центрів і закінчуючи сенсорними пристроями, доступна з відкритим вихідним кодом, і підтримувана Microsoft на Windows , Linux і Mac OSX .

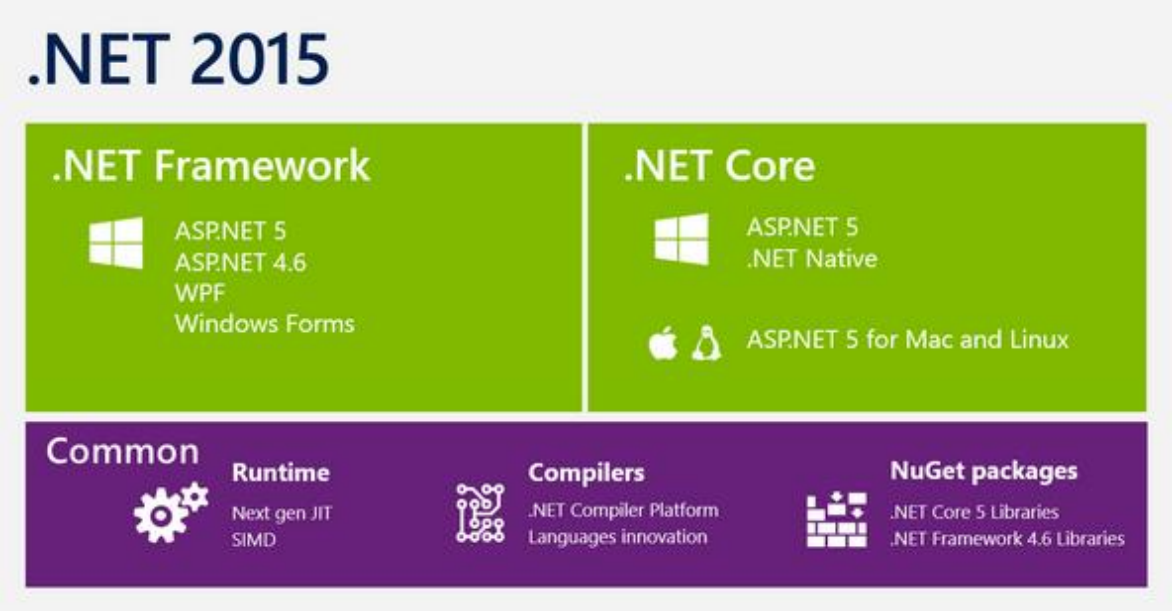

Рисунок 4.2 – Структура середовища .NET Core

Сам .NET Core компонується з бібліотек, що носять назву "CoreFX", і з власне невеликого середовища виконання "CoreCLR". Обидва цих компонента доступні через пакетний менеджер NuGet.

Основною перевагою .NET Core є можливість переносу, завдяки чому можна позбавити веб-додаток залежності від наявності на цільовій машині певної версії платформи .NET. Можна розгортати пакети CoreCLR разом з додатком

незалежно, що за версія .NET використовується на комп'ютері . При цьому можна розгорнути на одній машині відразу кілька додатків, які будуть використовувати різні версії CoreCLR .

CoreFX складається з набору бібліотек, що представляють інструменти з управління, доступом до консолі, колекціями, діагностикою, системою введення - виведення, LINQ, JSON, XML тощо. Варто зазначити, що кожна з бібліотек має мінімальне число залежностей від інших бібліотек, що покращує можливості переносу та розгортання пакетів CoreFX.

Для побудови додатків на базі .NET Core використовується кросплатформенне середовище виконання DNX. DNX – це середовище виконання і SDK, які необхідна для побудови та запуску додатків .NET на платформах Windows, Mac і Linux. При цьому DNX запускати не тільки веб-додатки, але і консольні, а також нативні мобільні додатки. Таким чином, можна розробляти проект на одні платформі, а запускати на іншій, з умовою що на ній розгорнута сумісна версія DNX.

4.3 Особливості платформи ASP.NET Core

Платформа ASP.NET 5 – технологія від компанії Microsoft, для створення веб-додатків різного типу. З однієї сторони, ASP.NET 5 являється продовженням розвитку платформи ASP.NET. Проте з іншої сторони, це фактично нова технологія та реорганізація платформи. Саме тому з недавнього часу платформу було названо ASP.NET Core. ASP.NET Core повністю кросплатформенна з відкритим вихідним кодом на GitHub.

Додаток ASP.NET Core може працювати з двома середовищами: .NET Core і з повною версією .NET Framework.

Завдяки модульності всі необхідні для веб-додатку компоненти можуть завантажуватися, як окремі модулі, через пакетний менеджер NuGet. Крім того, на

відміну від попередніх версій платформи немає необхідності використовувати глобальну бібліотеку System.Web.dll.

ASP.NET Core - більш оптимізована для роботи з хмарними технологіями.

При розгортанні для веб-додатку можна використовувати традиційний IIS. Але також можна запускати веб-додаток, використовуючи кросплатформенний веб-сервер Kestrel. Структура платформи ASP.NET Core зображена на рисунку 4.3.

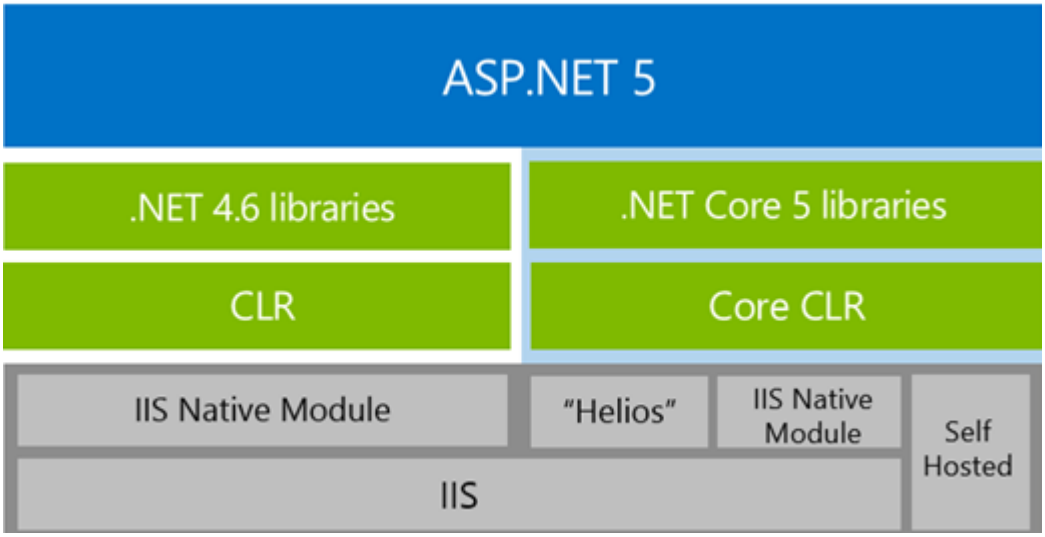

Рисунок 4.3 – Структура платформи ASP.NET

В загальному можна виділити наступні ключові особливості платформи, що в значній мірі відрізняють ііі від попредника:

– новий легкий конвеєр HTTP-запитів з модульною структурою;

– можливість розгортати додаток як на IIS, так і в рамках свого власного процесу;

– використання платформи .NET Core і її функціональності;

– поширення пакетів платформи через NuGet;

– конфігурація для спрощеного використання у хмарі;

– вбудована підтримка для інверсії управління;

– кросплатформенність: можливість розробки і розгортання додатків ASP.Net на Windows, Mac і Linux.

4.3.1 ASP.NET Core MVC

За замовчуванням у ASP.NET Core використовується шаблон проектування MVC (Model - View - Controller), від чого і походить концептуальна назва проекту.

Проте невірно ототожнювати ASP.NET Core цілком з фреймворком ASP.NET Core MVC. Фреймворк ASP.NET Core MVC працює поверх платформи ASP.NET Core, і призначений для того, щоб спростити створення програмних продуктів. Проте розробникможе і не використовувати MVC, а застосовувати чистий ASP.NET Core і на ньому цілком вибудовувати логіку програми.

Сам патерн MVC не є якоюсь новою ідеєю в архітектурі програмних продуктів, він з'явився ще в кінці 1970-х років в компанії Xerox як спосіб організації графічних компонентів у мові Smalltalk.

Концепція паттерна MVC передбачає поділ програми на три компоненти:

- Модель (model): описує використовувані в додатку дані, а також логіку, яка пов'язана безпосередньо з даними, наприклад, логіку валідації даних. Як правило, об'єкти моделей зберігаються в базі даних.

У MVC моделі представлені двома основними типами: моделі представлень, які використовуються представленнями для відображення і передачі даних, і моделі домену, які описують логіку управління даними.

Модель може містити дані, зберігати логіку управління цими даними. У той же час модель не повинна містити логіку взаємодії з користувачем і не має визначати механізм обробки запиту. Крім того, модель не повинна містити логіку відображення даних в представленні.

- Предсталення (view): відповідають за візуальну частину або інтерфейс користувача. Нерідко це html-сторінка, через яку користувач взаємодіє з додатком. Також представлення може містити логіку, пов'язану з відображенням даних. У той же час представлення не повинно містити логіку обробки запиту користувача або управління даними.

- Контролер (controller): представляє центральний компонент MVC, який забезпечує зв'язок між користувачем та програмою, представленням і сховищем даних. Він містить логіку обробки запиту користувача. Контролер отримує введені користувачем дані, обробляє їх та в залежності від результатів обробки відправляє користувачеві певний висновок, наприклад, у вигляді представлення, наповненого даними моделей.

Модель є незалежним компонентом - будь-які зміни контролера або представлення ніяк не впливають на модель. Контролер і представлення відносно незалежних компонентів. Так, з представлення можна звертатися до певного контролера, а з контролера створювати представлення, але при цьому нечасто їх можна змінювати незалежно від одного. Відносини між компонентами патерну можна описати схемою зображеною на рисунку 4.4.

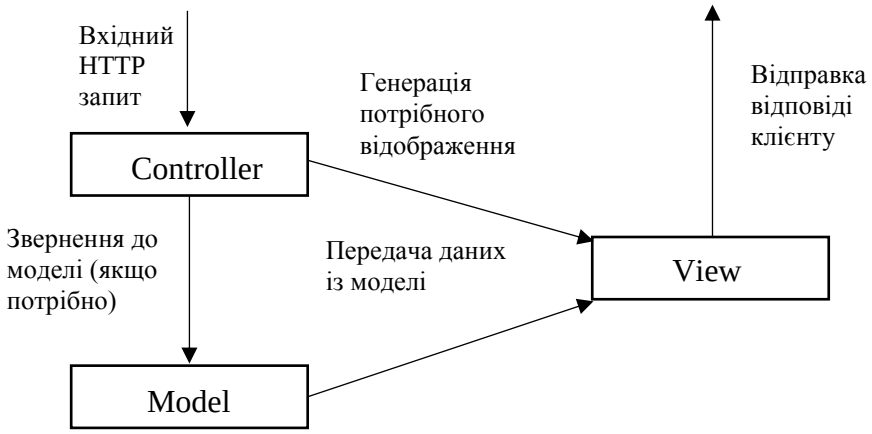

Рисунок 4.4 – Схема взаємодії компонентів MVC

Таке розмежування компонентів веб-додатку дозволяє реалізувати концепцію розділення відповідальності, при якій кожен компонент відповідає за свою строго відділену область функціональності системи. У зв'язку з чим легше побудувати роботу над окремими компонентами. І завдяки цьому додаток легше розробляти, підтримувати.

Створення проекту ASP.Net Core MVC в серидовищі Visual Studio включає декілька етапів:

1) Вибіри типу проекта (рисунок 4.5). В даному випадку це ASP.NET Core Web Application;

2) Вибір підтипу проекта (рисунок 4.6). В даному випадку це Web Application (Model – View – Controller);

3) Вибір типу аутентифікації (рисунок 4.7);

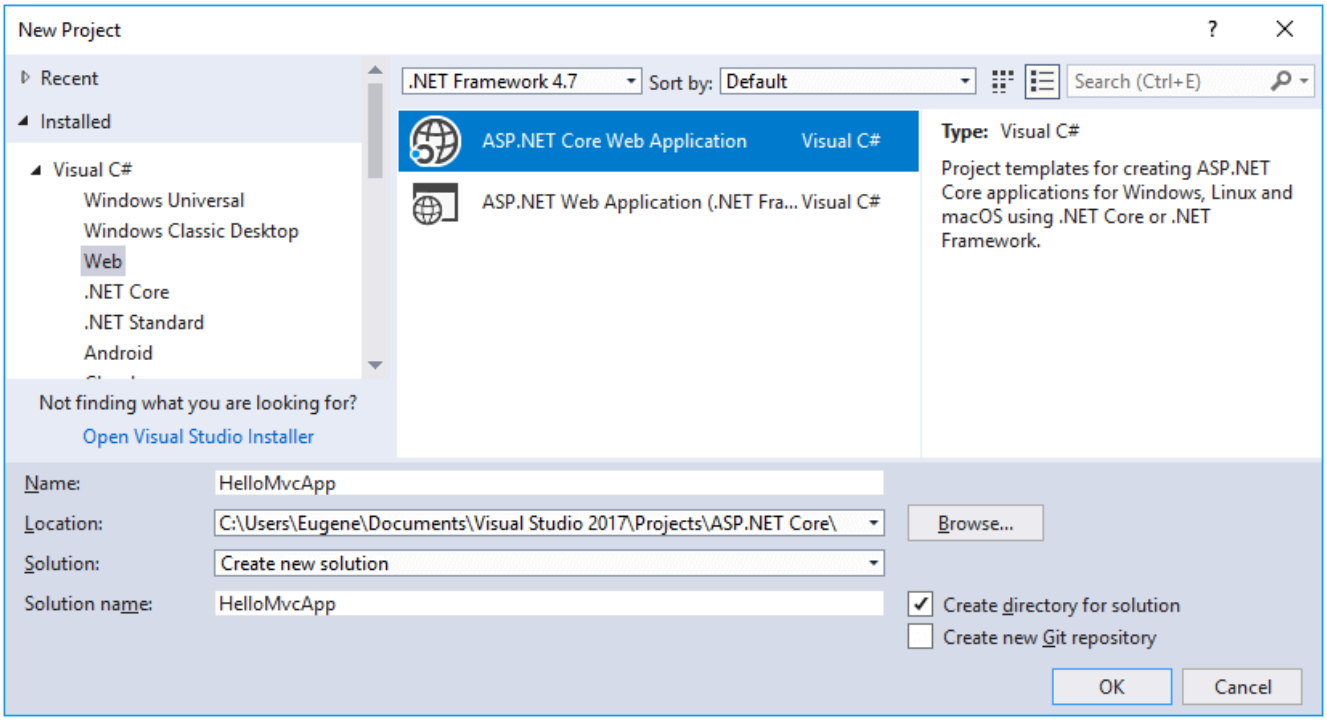

Рисунок 4.5 – Вікно вибору типу проекта

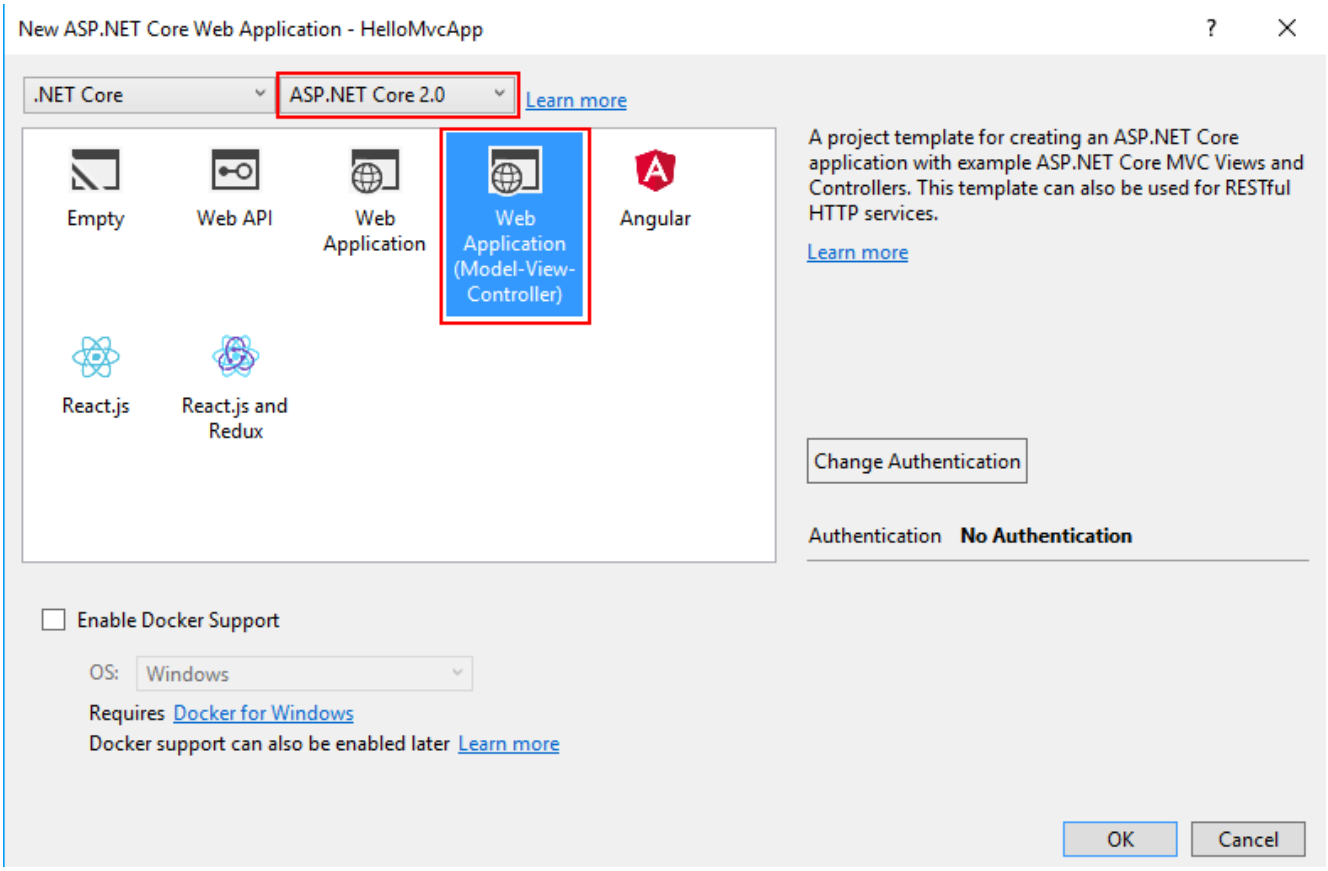

Рисунок 4.6 – Вікно вибору підтипу проекта

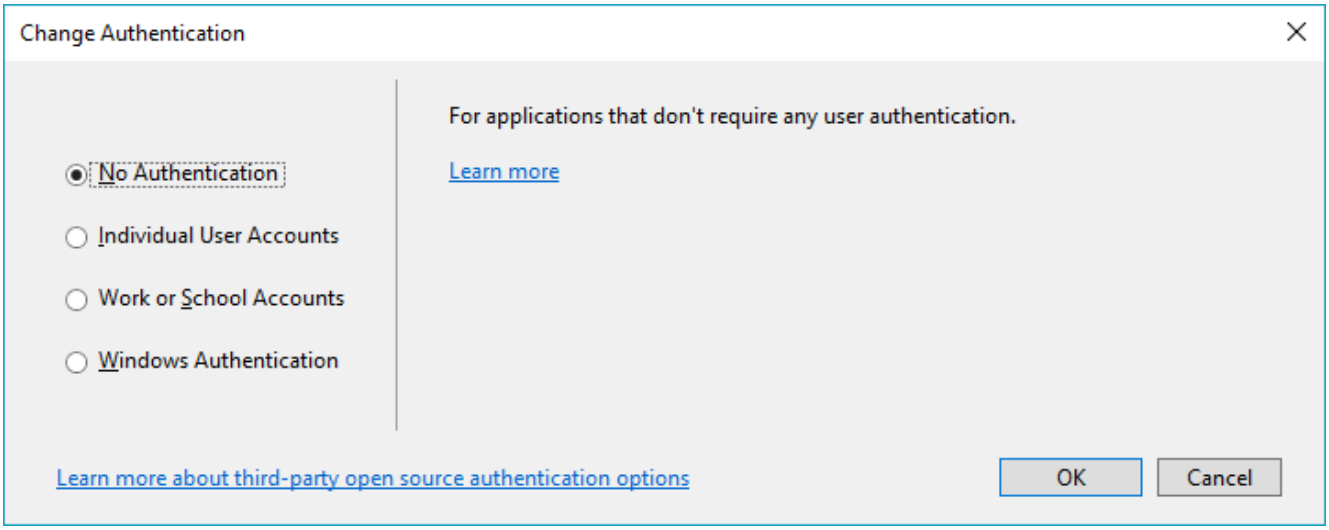

Рисунок 4.7 – Вікно вибору типу аутентифікації

Типова структура проекту має вигляд зображений на рисунку 4.8 та включає такі основні вузли:

- Dependencies: всі додані в проект пакети та бібліотеки;
- wwwroot: цей вузол (на жорсткому диску йому відповідає однойменна папка) призначений для зберігання статичних файлів - зображень, скриптів javascript, файлів css і т.д., які використовуються додатком. Мета додавання цієї папки в проект в порівнянні з іншими версіями ASP.NET, полягає в розмежуванні доступу до статичних файлів, до яких дозволений доступ з боку клієнта і до яких доступ заборонений;
- Controllers: папка для зберігання контролерів, які використовуються додатком;
- Models: папка для зберігання моделей;
- Views: каталог для зберігання уявлень;
- appsettings.json: конфігурація додатка у вигляді json файду;
- bower.json: файл, який керує клієнтськими залежностями (бібліотеки javascript і css), які підключаються через менеджер пакетів Bower;
- bundleconfig.json: файл, який містить завдання по мініфікації використовуваних скриптів і стилів, які виконуються при побудові проекту;
- Program.cs: файл, який визначає клас Program, який ініціалізує і запускає хост з додатком.

Startup.cs: файл, який визначає клас Startup, з якого починається робота програми. Тобто це вхідна точка в додаток.

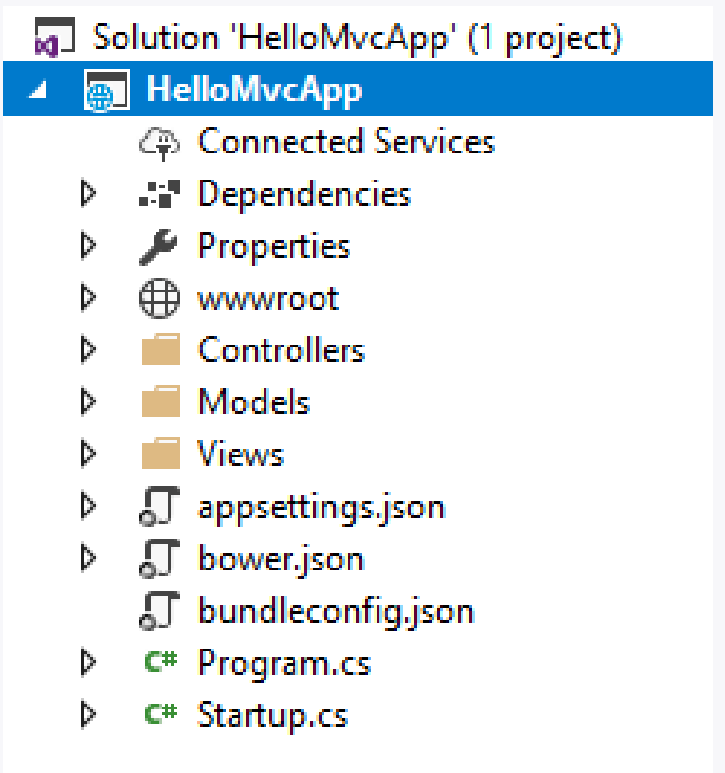

Рисунок 4.8 – Типова структура проекту Asp.NET MVC

4.3.2 ASP.NET Core WebAPI

Web API представляє спосіб побудови додатку ASP.NET, який спеціально розроблений для роботи в стилі REST (Representation State Transfer або передача стану представлення). REST-архітектура передбачає застосування наступних методів або типів запитів HTTP для взаємодії з сервером: GET, POST, PUT, DELETE.

Створення проекту ASP.Net Core WebAPI в серидовищі Visual Studio включаэ такы ж самы етапи, що і MVC, за вийнятком того, що обраний підтип програми буде обраний як Web API (рисунок 4.9).

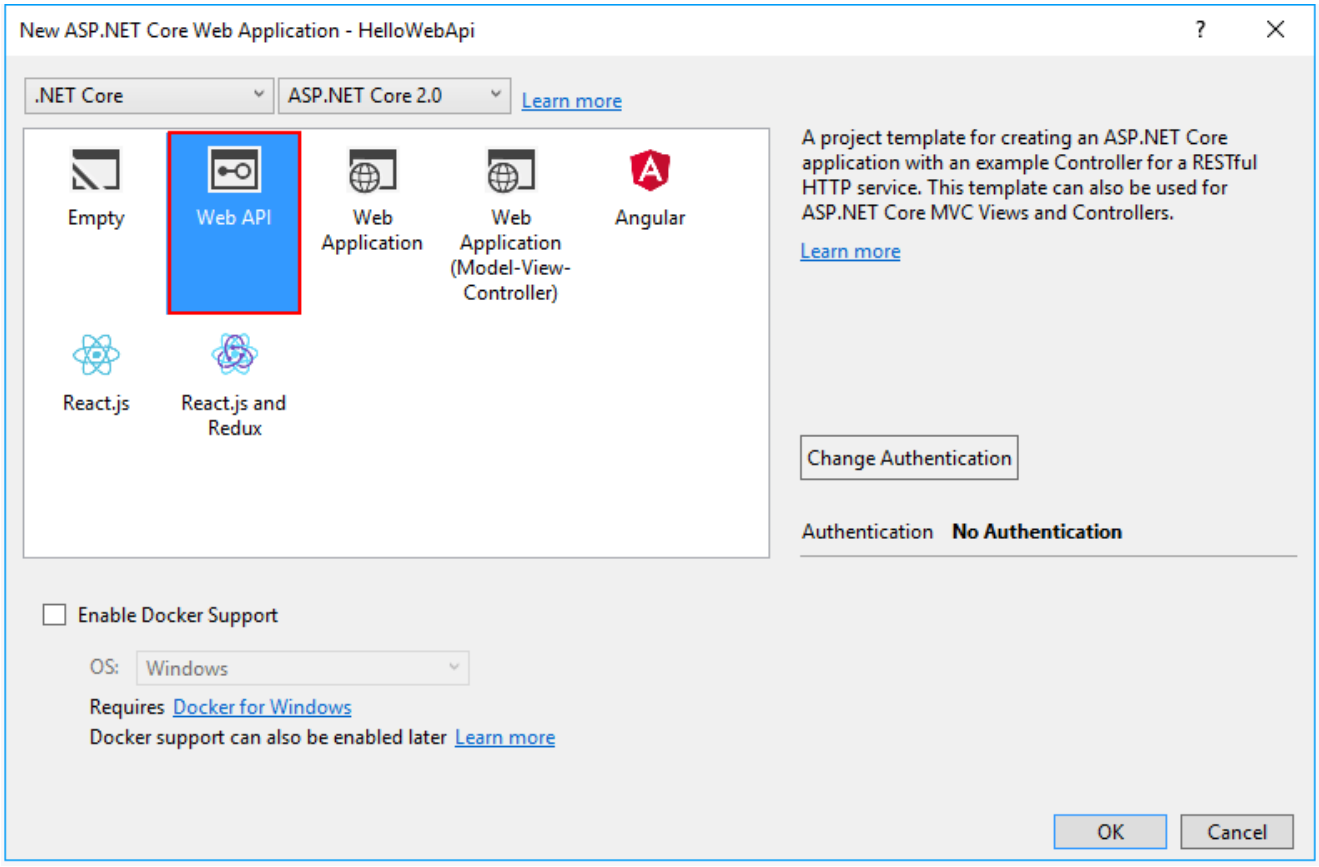

Рисунок 4.9 – Підтип проекту для Asp.Net WebAPI

Найчастіше REST-стиль особливо зручний при створенні будь-якого роду Single Page Application, які нерідко використовують спеціальні javascript-фреймворки типу Angular, React або Knockout. По суті Web API представляє собою веб-службу, до якої можуть звертатися інші додатки. Причому ці додатки можуть представляти будь-яку технологію і платформу - це можуть бути веб-додатки, мобільні або десктопні клієнти.

Структура проекту, зображена на рисунку 4.10, буде нагадувати структуру проекту MVC за виключенням того, що відстуні деякі вузли, що є ключовими для MVC.

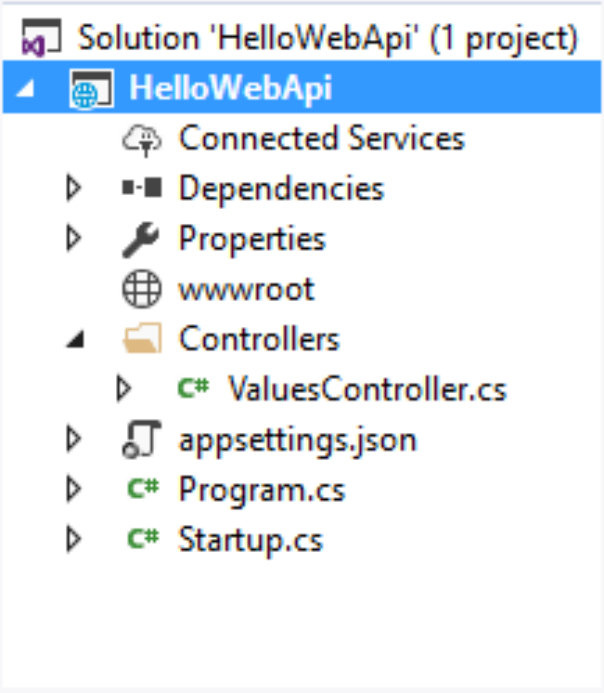

Рисунок 4.10 – Типова структура проекту Asp.NET MVC

4.3.2 Dependency injection

Dependency injection або впровадження залежностей – механізм, що дозволяє забезпечити слабку зв’язність компонентів системи. Такі компоненти зв’язані між собою через абстракції, наприклад через інтерфейси. При цьому відповідальність за створення відповідних залежностей покладається на зовнішній, спеціально призначений для цього загальний механізм.

Часто, в ролі таких механізмів виступають IoC-контейнери. Такі контейнери служать своєрідними фабриками, які встановлюють залежність між абстракціями та конкретними реалізаціями і, як правило, керують створенням цих об’єктів.

Такий дозволяє збільшити гнучкість системи, полегшити підтримку, розширення та заміну компонентів системи.

У попередніх версіях ASP.Net, для використання механізму впровадження залежностей потрібно було використовувати зовнішні IoC-контейнери, такі як Ninject, Unity, Autofac. Та починаючи з версії ASP.Net 5 у програмі присутній вбудований IoC-контейнер який представлений інтерфейсом IServiceProvider, керування яким проводиться у класі Startup.

4.4 Особливості Entity Framework 7

Entity Framework 7 – це спеціалізована об’єктно-орієнтована технологія на базі .NET Core для роботи з даними, ORM фреймворк для технології ADO.NET. На відміну від традиційних технологій роботи з базою даних, накштал ADO.NET, Entity Framework має більш високий рівень абстракції, що дозволяє працювати з даними незалежно від типу сховища, що значно підвищує можливості повторного використання коду. Якщо на фізичному рівні ми працюємо з таблицями, первинними та зовнішніми ключами, індексами, то на абстрактному рівні який нам надає дана технологія, ми вже працюємо з об’єктами, що відображають модель даних. Центральною місце в концепції Entity Framework займає поняття сутності. Сутність представляє собою набір даних, асоційованих з певним об’єктом. Тому дана дозволяє працювати не з таблицями та їх рядками, а з об’єктами. У кожної сутності є один або декілька параметрів, які відрізняють цю сутність від інших і будуть унікально її визначати. Подібні параметри називають первинними ключами. Також сутності можуть бути пов'язані асоціативними зв'язками, такими як: один до багатьох, один до одного і багато до багатьох. Імітуючи реальні бази даних та зв’язки через зовнішні ключі.

Однією із переваг Entity Framework є використання запитів LINQ для вибірки даних. За допомогою LINQ ми можемо не тільки отримувати об’єкти

сутностей із бази даних, а й отримувати об'єкти, пов'язані різними асоціативними зв'язками.

Іншим ключовим поняттям є блок EDM. Цей блок зіставляє класи сутностей з реальними таблицями в базі даних.

Entity Framework передбачає три можливі способи взаємодії з базою даних:

– Database first: Entity Framework самостійно генерує набір класів, які відображають модель даних уже існуючої бази даних;

– Model first: розробник створює модель даних, на основі якої Entity Framework надалі генерує реальну базу даних на сервері;

– Code first: розробник створює класи моделі даних, які будуть зберігатися в базі даних. Далі Entity Framework на основі створених класів моделі генерує базу даних з відповідними таблицями.

4.5 Особливості системи аутентифікації та авторизації ASP.Net Identity

ASP.Net Identity представляє вбудовану в ASP.Net систему автентифікації та авторизації. При створенні проекту в середовищі Visual Studio, розробнику дають можливість вибрати один із наступних типів автентифікації:

– No Authentication: ASP.Net Identity і вбудована система автентифікації відсутня;

– Individual User Accounts: проект за замовчуванням включає систему ASP.Net Identity, яка дозволяє авторизуватись користувачам як всередині програми, так і за допомогою зовнішніх сервісів;

– Organizational Accounts: підходить для сайтів та веб-додатків окремих компаній та організацій;

– Windows Authentication: система автентифікації за допомогою облікових записів Windows;

Ключовими об'єктами Asp.Net Identity є користувачі та ролі. Вся функціональність по створенню, видаленню користувачів, ролей, взаємодії зі сховищем користувачів знаходяться в бібліотеці Microsoft.AspNet.Identity.Core та являє собою ієрархію класів, зображену на рисунку 4.11.

Після вибору відповідного типу аутентифікації, автоматично у проект включаються всі необхідні залежності та створюється початковий шаблон. У вузлі References, зображеному на рисунку 4.12, відображаються усі пакети, необхідні для функціонування Identity.

Крім того, оскільки ASP.Net Identity для роботи з базами даних за замовчуванням використовує Entity Framework, в проект автоматично включаються необхідні пакети для роботи з базами даних та, за бажанням розробника, створюється власне сама база даних користувачів з початковою структурою таблиць.

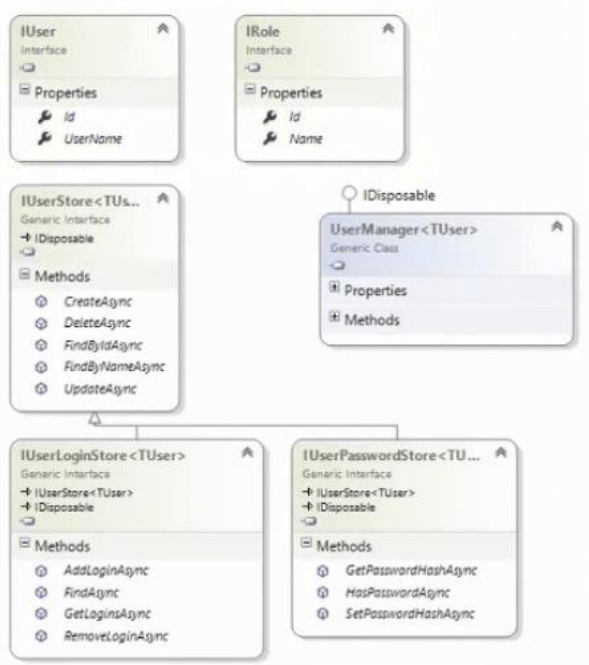

Рисунок 4.11 – Ієрархія класів Identity.Core

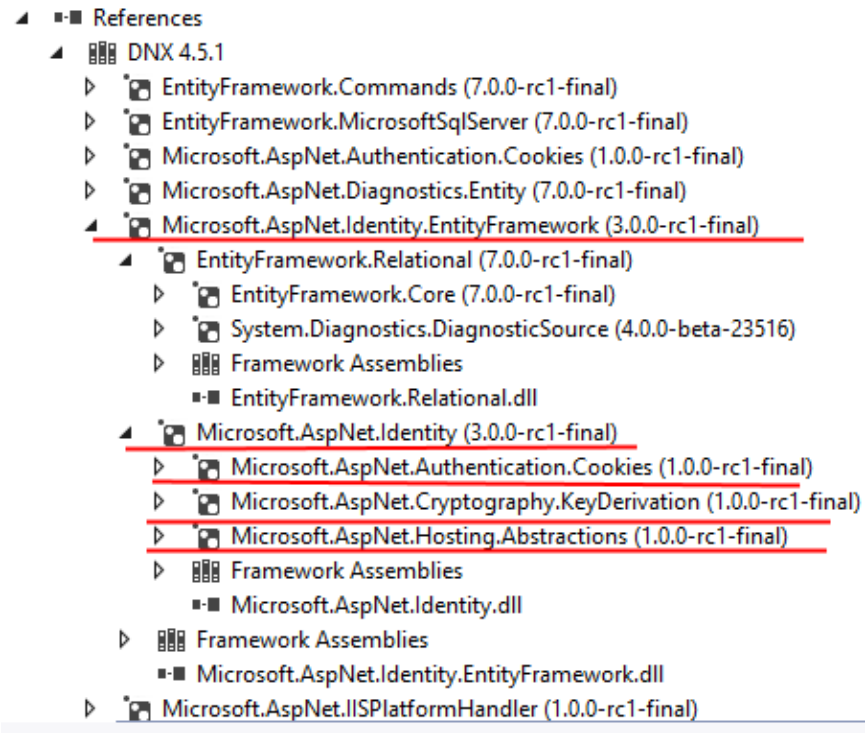

Рисунок 4.12 – Вузол References

Розглянемо детальніше вузол References:

– Microsoft.AspNet.Identity – основна бібліотека ASP.Net Identity, яка зберігає весь базовий функціонал;

– Microsoft.AspNet.Identity.EntityFramework – реалізація інтерфейсів ASP.Net Identity з використанням Entity Framework, що дозволяє взаємодіяти зі сховищем даних;

– Microsoft.AspNet.Authentication.Cookies – пакет, що дозволяє користувачам автентифікацію на основі cookies;

– Microsoft.AspNet.Cryptography.KeyDerivation – функціонал для роботи з ключами безпеки, шифруванням;

– Microsoft.AspNet.Hosting.Abstractions – абстракції для хостинга додатків ASP.Net;

Кожен користувач представляє об'єкт інтерфейсу IUser. А всі операції по управлінню користувачами виконуються через сховище, представлене інтерфейсом IUserStore.

Кожна роль представляє реалізацію інтерфейсу IRole, а управління ролями відбувається через сховище представлене інтерфейсом IRoleStore.

Безпосередню реалізацію інтерфейсів IUser, IRole, IUserStore і IRoleStore надає простір імен Microsoft.AspNet.Identity.EntityFramework, та являє собою ієрархію класів, зображену на рисунку 4.13.

Проте розробники не оперють безвопсередньо сутностями IdentityUser, IdentityDbContext, а мають справу з класами-наслідниками. Подібна архітектура дозволяє взяти вже готовий функціонал і при необхідності додати новий, наприклад, додати для користувача нову властивість або додати нову таблицю в базу даних.

Також в проект включається стартовий шаблон роботи з Identity, від чого структура проекту набуває вигляд, зображений на рисунку 3.6.

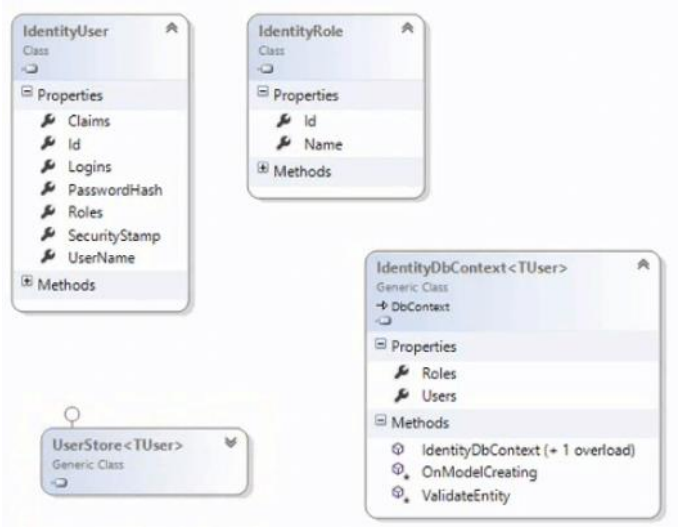

Рисунок 4.13 – Ієрархія класів Identity.EntityFramework

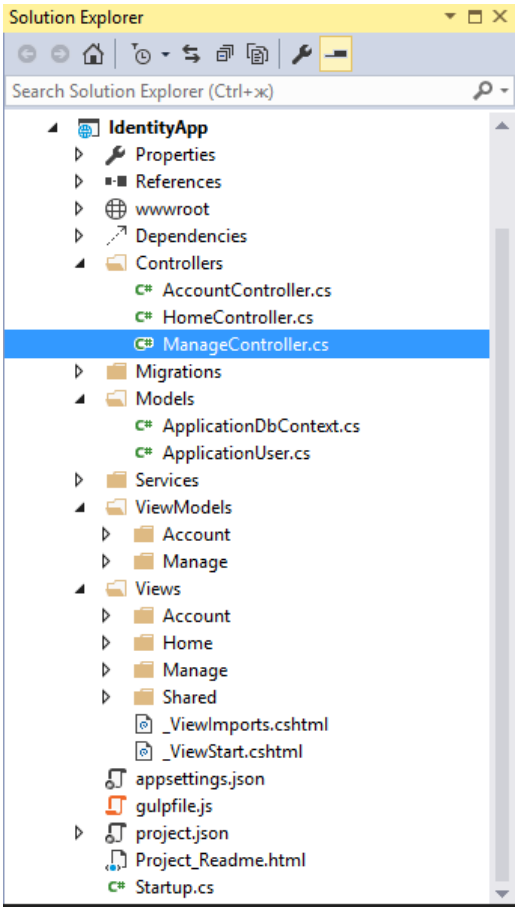

Рисунок 4.14 – Початкова структура проекту з Identity

4.6 Особливості мови JavaScript

JavaScript – прототипно-орієнтована мова програмування із С-подібним синтаксисом. Є реалізацією стандарту ECMAScript. Це проста в освоєнні, потужна мова сценаріїв, що широко використовується для управління поведінкою веб-сторінки.

Прототипне програмування – стиль об’єктно-орієнтованого програмування, де відсутнє поняття класу, а наслідування відбувається шляхом клонування існуючого примірника об’єкта – прототипу.

Основні архітектурні риси JavaScript: динамічна слабка типізація, автоматичне управління пам’яттю, прототипне програмування.

Найбільш широко використовується у веб-розробці, як мова сценаріїв, для надання інтерактивності веб-сторінкам. Працюючи на стороні клієнта, JavaScript може керувати розподілом даних, дизайном, реакцією на певні події тощо.

У всі основні браузери вбудований інтерпретатор JavaScript, саме тому вони можуть виконувати скрипти на сторінці. Але, зрозуміло, що дану мову можна використовувати не тільки в браузері. Це повноцінна мова програмування, програми якої можна запускати і на сервері, і будь-якому іншому середовищі, де встановлений відповідний інтерпретатор.

Можливості JavaScript залежать від середовища, у якому запущена програма. В браузері JavaScript вміє робити все, що стосується маніпуляції зі сторінкою, взаємодії з користувачем і, в деякій мірі, з сервером. Наприклад: створення нових елементів HTML, видалення існуючих, зміна стилів елементів, реакція на дії користувача, відправка запитів на сервер, отримувати та встановлювати cookie тощо.

JavaScript – швидка та потужна мова, але браузер накладає на неї значні обмеження. Це зроблено для безпеки користувачів, щоб шахраї не могли з її допомогою отримати особисті данні користувача чи нашкодити його комп’ютеру. До основних обмежень відносяться: JavaScript не може зчитувати чи записувати будь які файли на жорсткий диск, копіювати їх чи запускати програми, оскільки не має прямого доступу до операційної системи; JavaScript, що працює в одній вкладці, не може звертатися до інших вкладок чи вікон браузера.

Мова JavaScript активно розвивається у наш час, як і веб-технології в цілому. А нові стандарти ECMAScript сприяють лише покращенню та розширенню функціональних можливостей та синтаксису мови.

4.7 Мова розмітки гіпертексту HTML

Мова розмітки гіпертексту HTML – це стандартна мова розмітки, яка використовується для створення веб-сторінок. HTML було створено на основі мови SGML, що пояснює концепцію визначення типу документу та структурної розмітки тексту. Поряд з CSS та JavaScript, HTML є наріжним каменем технологій створення веб-сторінок та користувацьких інтерфейсів для мобільних та веб-додатків.

HTML елементи утворюють структурні блоки сторінки для зображень, тексту, відео чи інтерактивних форм з різноманітним вмістом.

Дана мова розмітки дозволяє створювати структуровані сторінки за допомогою семантичного розподілу тексту на заголовки, параграфи, посилання, переліки, цитати тощо. Структурні елементи формуються за допомогою використання тегів, структура яких зображена на рисунку 4.15.

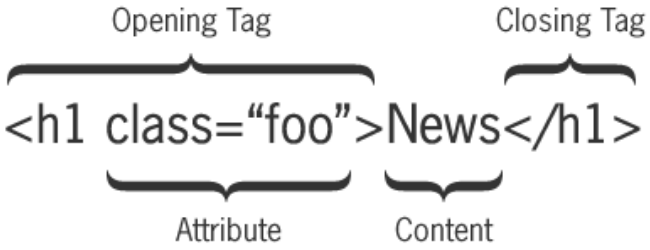

Рисунок 4.15 – Структура тегів HTML

Теги, такі як <img> та <input> визначають контент сторінки. Інші, такі як <p> та <h1> визначають семантичний розподіл тексту сторінки. Також одні теги можуть включати в себе інші, в якості допоміжних елементів. Браузери не відображають теги, але використовують їх для інтерпретації вмісту сторінки.

Типи даних HTML ґрунтуються на базових типах мови SGML, наприклад: color, text, datetime.

Як і природна мова, комп'ютерна мова має свої основні елементи: словник, синтаксис та граматика. HTML використовує машинно-зчитуючу граматику, яка називається DTD – механізм, успадкований від SGML. Процес перевірки документа на дотримання прави визначених мовою називається валідацією, а інструмент, що здійснює перевірку – валідатором. Документ, що не містить помилок, називають валідним. Згідно з цією концепцією, валідація HTML документу визначається, як процес його перевірки за правилами граматики визначеними в DTD, посилання на які визначені структурним елементом doctype.

Для перегляду HTML-розмітки можна використовувати будь-який текстовий редактор, а для перегляду, відтвореного за правилами розмітки, використовується браузер.

Зазвичай HTML-документи передаються двома шляхами:

– по мережі, за допомогою протоколів передачі даних. Найчастіше використовують протокол HTTP;

– по електронній пошті. Більшість поштових клієнтів мають функціонал для роботи з HTML, що дозволяє забезпечити візуальне форматування тексту. Проте з іншого боку, використання HTML в електронних повідомленнях дозволяє приховати фішинг – один із видів інтернет-шахрайства.

4.7 Каскадні таблиці стилей CSS

CSS – формальна мова, для опису зовнішнього вигляду документа, написаного за допомогою мови розмітки.

Зазвичай використовується для оформлення зовнішнього виду веб сторінок, написаних за допомогою HTML. Хоча може застосовуватися і разом з іншими мовами розмітки, такими як XML.

У сучасній веб-розробці каскадні таблиці стилей – це незамінний інструмент для візуального оформлення веб-сторінок та користувацьких інтерфейсів для мобільних та веб-додатків.

CSS був створений в першу чергу для того, щоб відокремити зміст документу, від його візуального представлення, що включає шрифти, кольори, відступи тощо. Такий поділ сприяє поліпшенню доступності контенту, забезпечує більшу гнучкість, контроль над відображенням в різних умовах та можливість повторного використання написаного коду.

Правило CSS, структура якого зображена на рисунку 4.16, складається з двох основних частин: селектор, який вибирає ті структурні частини документу, до яких правило застосовується і блок декларації, де оголошують значення властивостей.

Існує декілька видів селекторів, серед яких найбільш популярними є: універсальний селектор, селектор класів, селектор ідентифікаторів, селектор елементів, селектор нащадків та селектор дочірніх елементів.

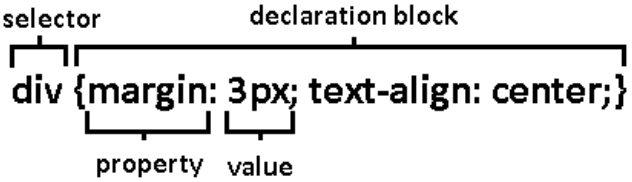

Рисунок 4.16 – Структура правила в CSS

Існує декілька видів селекторів, серед яких найбільш популярними є: універсальний селектор, селектор класів, селектор ідентифікаторів, селектор елементів, селектор нащадків та селектор дочірніх елементів.

Головна відмінність між класом та ідентифікатором в тому, що окремий клас може бути присвоєний декільком елементам в межах документу, а ідентифікатор – унікальне значення в межах документу, що присвоюється тільки

одному елементу. Також відмінність у тому , що можуть існувати множинні класи, коли клас окремого елемента складається з декількох слів, розділених пробілами. У такому випадку до елемента будуть застосовані правила для усіх класів у множині. Для ідентифікаторів таке неможливо .

Існує декілька способів підключення CSS до документу:

– таблиця стилей, описана в окремому файлі, може бути підключена до веб-документу за допомогою тегу <link>, розміщеного в блоці, що описується тегом <head>. Правила таблиці, підключеної таким способом, діють протягом всього документу;

– таблиця стилей, описана в окремому файлі, може бути підключена за допомогою директиви @import, розміщеної в блоці, що описується тегом <style>. Правила таблиці, підключеної таким способом, діють протягом всього документу;

– коли таблиця стилей описується в самому документі, її розміщують в блоці, що описується тегом <style>. Правила таблиці, підключеної таким способом, діють протягом всього документу;

– коли таблиця стилей описується в самому документі, її можна розміщувати в тілі окремого тегу, за допомогою атрибута style. Правила таблиці, підключеної таким способом, діють тільки на вміст тегу, в якому вони описані.

4.8 Серидовище розробки Visual Studio

В якості середовища розробки обрано Microsoft Visual Studio 2015, що надає весь необхідний функціонал для розробки веб-додатків на основі платформи .NET Core.

Microsoft Visual Studio – серія програмних продуктів компанії Microsoft для розробки програмного забезпечення. Включає в себе інтегроване середовище

розробки та перелік інших засобів для розробки та впровадження програмного забезпечення.

Visual Studio включає в себе редактор коду, що підтримує технологію IntelliSense – компонент завершення коду. Інші вбудовані інструменти включають в себе: конструктор форм для побудови інтерфейсів додатків, веб-дизайнер, функціональні блоки для роботи з базою даних тощо.

Підключення додаткової функціональності за допомогою плагінів дозволяє включити підтримку систем контролю версій, додаткових редакторів, наборів інструментів для інших аспектів життєвого циклу розробки програмного забезпечення.

Присутня вбудована підтримка різних мов програмування, таких як C, C++, Visual Basic, F#, C#. Підтримку інших мов, таких як Python, Ruby, M можна забезпечити за допомогою підключення відповідних мовних служб, встановлених окремо.

Visual Studio за замовчування включає наступні компоненти:

- Visual Basic .NET, а до його появи — Visual Basic
- Visual C++
- Visual C#
- Visual J#
- Visual F# (входить до складу Visual Studio 2010)
- Visual Studio Debugger.

5 ОСОБЛИВОСТІ ІНТЕГРАЦІЇ З СИСТЕМОЮ IBM WATSON ANALYTICS

API[8] – це прикладниий програамний інтерфеейс, набір визначень взаємодії різнотипного програмного забезпечення. Одним з найпоширеніших призначень API є надання набору широко використовуваних функцій однієї системи іншій.

В контексті веб-розробки API являє собою своєрідну веб-службу, до якої можна звернутися задля отримання певних даних або доступу до віддалених ресурсів.

IBM Watson Analytics має відкритий API системи, що дає можлвивість розробникам інтегрувати функціональність IBM з іншими програмними продуктами.

5.1 Доступ до API

Для отримання ноступу до відкритого API розробник повинен зареєструвати свій обліковий запис в серидовищі IBM, а саме в системі IBM Developers Center[9], інтерфейс якої зображений на рисунку 5.1.

IBM Developers Center – спеціалізований сервіс для розробників де можна отримати інформацію про потрібні технології, моделі доступу до них, навчальні матеріали, тощо.

Після отримання облікового запису в серидовищі IBM Developers Center, розробнико отримує доступ до панелі developerWorks де відбувається підключення та конфігурація потрібних сервісів. Інтерфейс панелі developerWorks з переліком підключених технологій зображений на рисунку 5.2.

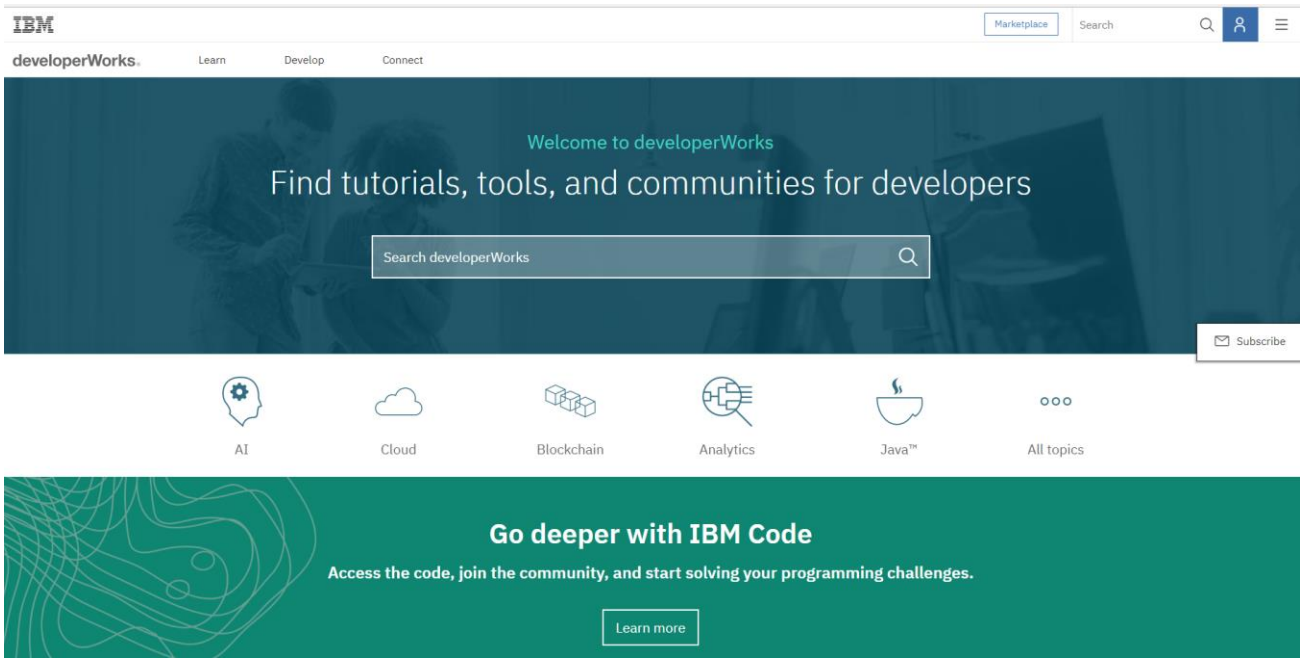

Рисунок 5.1 – Інтерфейс системи IBM Developers Center

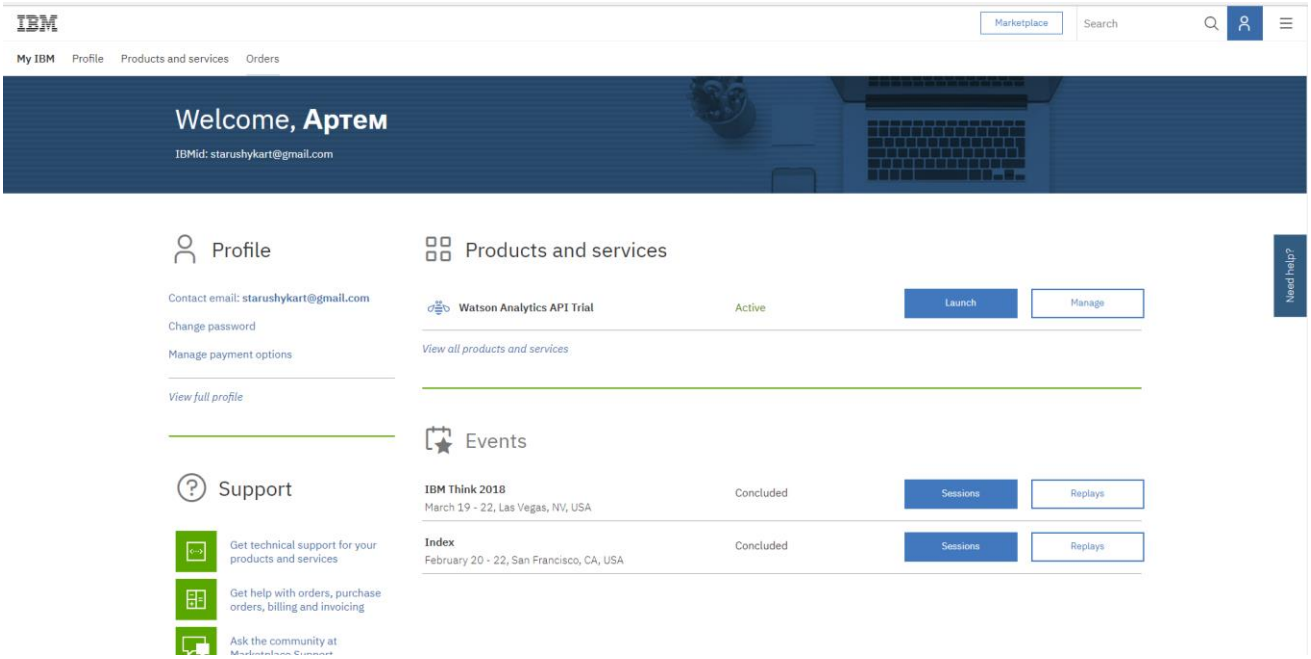

Рисунок 5.2 – Серидовище developerWorks з переліком підключених технологій

Для отримання індивідуальних ключів доступу до API потрібно перейти на вкладку api explorer. Дана вкладка призначення для перегляду, конфігурування та оновлення ключів доступу. Вкладка api explorer з переліком персональних ключів доступу зображена на рисунку 5.3.

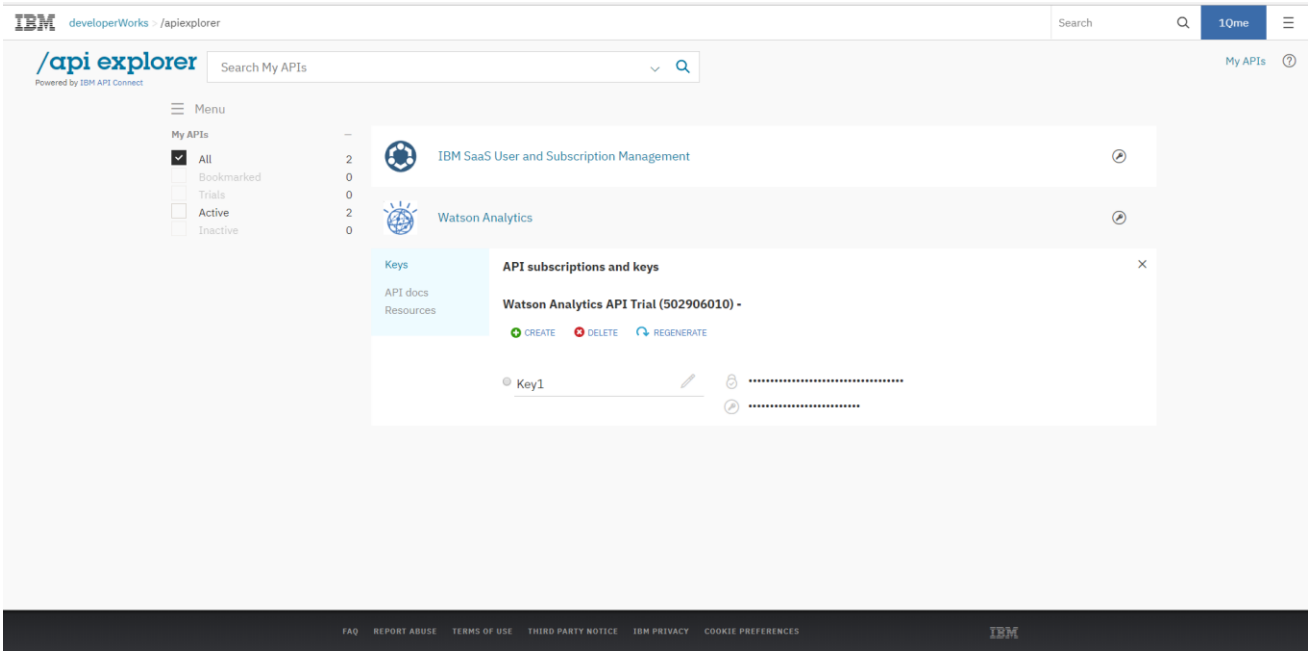

Рисунок 5.3 - Вкладка api explorer з переліком персональних ключів

Ключі доступу необхідні для ідентифікації розроблюваного додатку в серидовищі IBM. Кожен додаток має індивідуальний набір ключей, на основі яких реєструється клієнтська конфігурація та проводиться подільша робота з API.

5.2 IBM Watson Analytics API Explorer

Після отримання та конфігурації ключів доступу розробник може почати роботу з API. Для цього із вкладки api explorer потрібно перейти до конрктеної технології та її документації. Вікно IBM Watson Analytics API Explorer зображено на рисунку 5.4.

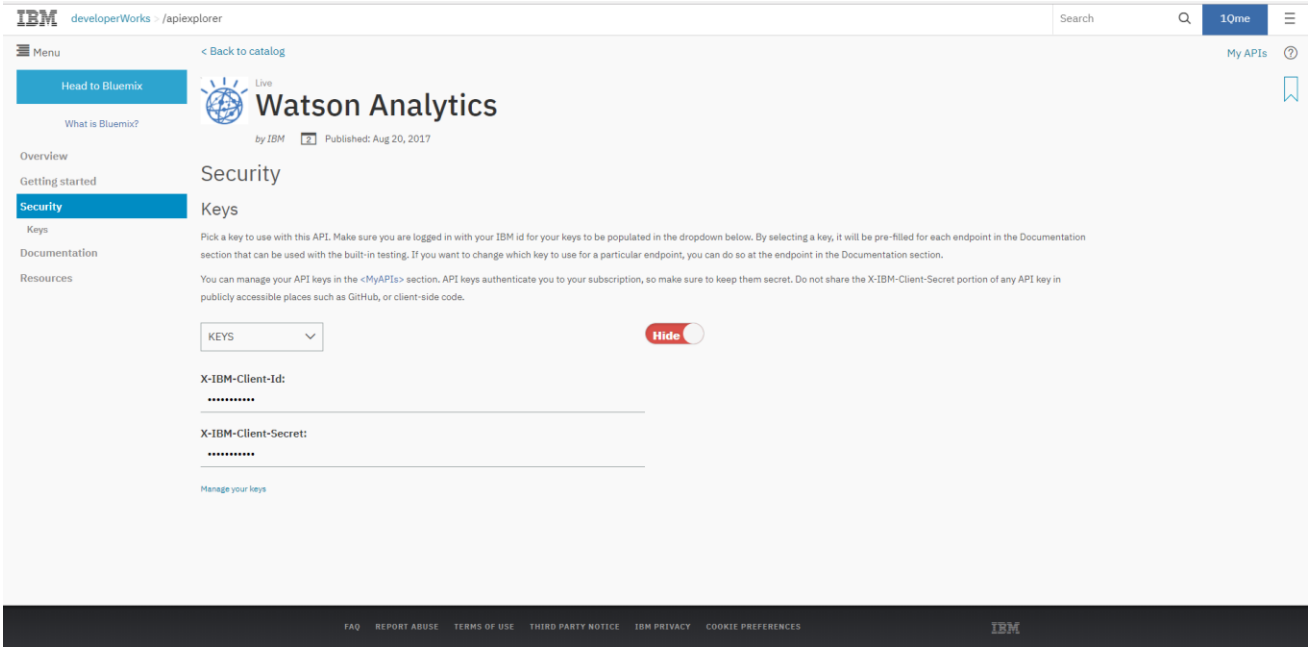

Рисунок 5.4 - IBM Watson Analytics API Explorer

Як бачимо, для зручності використання на даній вкладці також можна переглянути відповідні ключі доступу, що дозволяє зменшити кількість переходів на інші вклади під час роботи.

Зручна панель навігації дозволяє швидко та ефективно переміщатися по параграфах для ефективної роботи та включає в себе перелік таких секцій: загальні відосомості, інформації для початку роботи, панель доступу, документацію, та додаткові ресурсі.

IBM Watson Analytics API по своїй структурі розділений на декілька секцій зображених на рисунку 5.5.

Розглянемо детальніше вищезгадані секції:

- Registration API – категорія запитів до системи IBM Watson Analytics, призначених для реєстрації та отримання інформації про клієнтську конфігурацію. Клієнтська конфігурація реєструється на отснові попредньо отриманих ключів доступу.

- Authorization API – категорія запитів до системи IBM Watson Analytics, призначених для авторизації користувача в системі через розроблюваний додаток. Авторизація користувача проводиться на основі попередньо зареєстрованої клієнтської конфігурації.

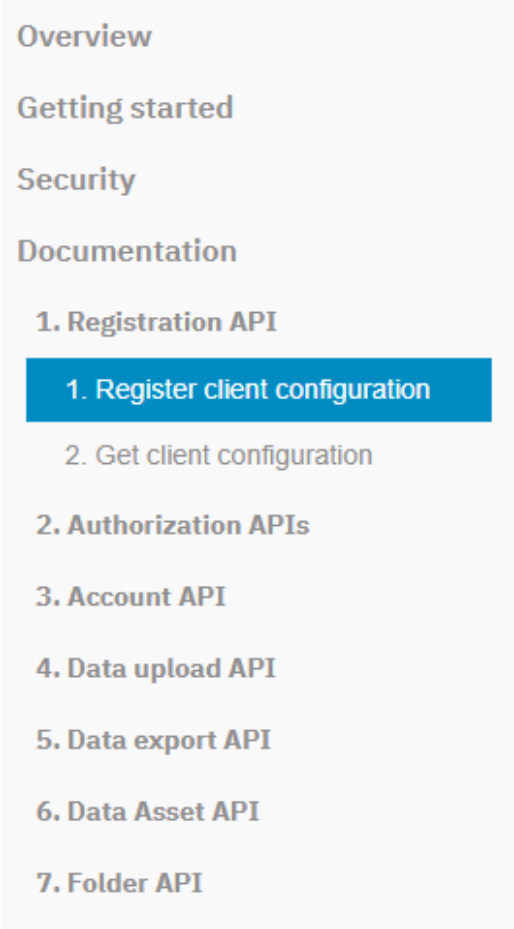

Рисунок 5.5 – Категоріальни розподіл IBM Watson Analytics API

- Account API – категорія запитів до системи IBM Watson Analytics, призначених для отримання інформації про авторизованого користувача.

- Data upload API – категорія запитів до системи IBM Watson Analytics, призначених для завантажження даних. Оскільки розмір завантажуваних данних може перевизувавти допустимі розміри запиту, то присутні два методи звавнтаження: одним сегментом та декількома сегментами.

- Data export API – категорія запитів до системи IBM Watson Analytics, призначених для експорту даних та включають в себе низку основних запитів, таких як запит на початок експорту, отримання статусу експорту та отримання посилання не експортований файл.

- Data Asset API – категорія запитів до системи IBM Watson Analytics, призначених для конфігурування та отримання інформації джерела даних. За допомогою даних запитів можна видалити джерело даних, отримати або оновити його властивості, дозволи тощо.

- Folder API – категорія запитів до системи IBM Watson Analytics, призначених для керування внутрішньою ієрархуєю каталогів. За допомогою даних запитів можна створювати, видаляти, шукати, отримувати властивості каталогів, тощо.

5.3 Структура запитів

Оскільки API надає доступ до персональних даних зареєстрованих користувачів, це потребує особливої структури запитів, щоб запобігти проблемам безпеки данних.

IBM Watson Analytics API оперує двома типами запитів: системними запитами та запитами користувача.

Системні запити – запити, потрібні для функціонування програми, в яку інтегрується Watson. Такі запити повинні включати передачу отриманих раніше ключів доступу для ідентифікації програми в серидовищі IBM. До системних запитів відноситься запит для реєстраціє конфігурації клієнта, структура якого зображена на рисунку 5.6.

```
var settings = {
  "async": true,
  "crossDomain": true,
  "url": "https://api.ibm.com/watsonanalytics/run/oauth2/v1/config",
  "method": "PUT",
  "headers": {
    "x-ibm-client-secret": "string",
    "x-ibm-client-id": "string",
    "x-ibm-wa-data-center": "string",
    "accept": "application/json",
    "content-type": "application/json"
  },
  "processData": false,
  "data": "
{\"clientName\":\"\",\"ownerCompany\":\"\",\"ownerEmail\":\"\",\"ownerName
\":\"\",\"ownerPhone\":\"\",\"redirectURIs\":\"\"}"
}
```

Рисунок 5.6 – Структура системного запиту для реєстрації конфігурації клієнта

Рознлянемо детальніше структуру запиту:

- method – тип запиту[10]. Доступні запити типу GET, PUT, POST, DELETE.

- headers – перелік заголовків http запиту, що дозволяють передати додаткову інофрмацію в клієнт-серверній арзітектурі.

- x-ibm-client-secret – обов’язковий заголовок, який предає клієнтський секрет асоційований з розроблюваним додатком. Клієнтський секрет яквляє собою один із ключів доступу отриманих на попередніх етапах.

- x-ibm-client-id – обов’язковий заголовок, який предає ідентифікатор клієнта асоційований з розроблюваним додатком. Ідентифікатор клієнта яквляє собою один із ключів доступу отриманих на попередніх етапах.

- x-ibm-wa-data-center – необов’язковий заголовок, який передає інформацію про дата-центр з яким працює розроблюваний додаток.

- accept – обов’язковий заголовок, який передає інформацію про допустимі формати данних для передачі.

- content-type - обов’язковий заголовок, який передає інформацію про формат данних для поточного запиту.

- data – тіло запиту в якому передається інформація про клієнта у json форматі, відповідно до вказаного заголовку.

Запити користувача – запити від конкретного користувача, після процесу авторизації, для виконання певного функціоналу. Такі запити повинні включати передачу отриманих раніше ключів доступу для ідентифікації програми в серидовищі IBM та передачу токена, отриманого на етапі авторизації, для ідентифікації користувача. До запитів даного типу відносить запит на створення нового каталога, структура якого зображена на рисунку 5.7.

```
var settings = {
  "async": true,
  "crossDomain": true,
  "url": "https://api.ibm.com/watsonanalytics/run/content/v1/folders",
  "method": "POST",
  "headers": {
    "x-ibm-client-secret": "string",
    "authorization": "string",
    "x-ibm-client-id": "string",
    "accept": "application/json",
    "content-type": "application/json"
  },
  "processData": false,
  "data": "{\"name\":\"\",\"parent\":{\"description\":\"The parent folder
where you want to create the new folder.\",\"items\":
{\"type\":\"string\"},\"type\":\"object\"}}"
}
```

Рисунок 5.7 – Структура запиту користувача для створення нового каталога

Рознлянемо детальніше структуру запиту:

- method – тип запиту. Доступні запити типу GET, PUT, POST, DELETE.

- headers – перелік заголовків http запиту, що дозволяють передати додаткову інофрмацію в клієнт-серверній арзітектурі.

- x-ibm-client-secret – обов’язковий заголовок, який предає клієнтський секрет асоційований з розроблюваним додатком. Клієнтський секрет яквляє собою один із ключів доступу отриманих на попередніх етапах.
- authorization – обов’язковий заголовок, який предає токен, отриманий на етапі авторизації, для ідентифікації користувача.
- x-ibm-client-id – обов’язковий заголовок, який предає ідентифікатор клієнта асоційований з розроблюваним додатком. Ідентифікатор клієнта яквляє собою один із ключів доступу отриманих на попередніх етапах.
- accept – обов’язковий заголовок, який передає інформацію про допустимі формати данних для передачі.
- content-type - обов’язковий заголовок, який передає інформацію про формат данних для поточного запиту.
- data – тіло запиту в якому передається інформація про каталог у json форматі, відповідно до вказаного заголовку.

5.4 Особливості авторизації користувача

Процес авторизації користувача в системі IBM за допомогою API відрізняється від тридиційних підходів, в сторону більш жорсткого контролю доступу.

Після того як конфігурація клієнта зареєстрвана в Watson Analytics можна отримати код авторизації. Це одноразовий код, який сервер може обміняти на токен. Токен доступу використовується для клієнтських запитів у Watson Analytics. Відмінністю від традиційних способів отримання токену є те, що запит для отримання коду авторизації повинен виконуватись із браузера, та мати структуру, зображену на рисунку 5.8.

```
var settings = {
  "async": true,
  "crossDomain": true,
  "url": "https://api.ibm.com/watsonanalytics/run/clientauth/v1/auth?
client_id=string&dc=string&redirect_uri=string&response_type=string&scope=
string&state=string",
  "method": "GET",
  "headers": {
    "accept": "application/json"
  }
}
```

Рисунок 5.8 – Структура запиту для отримання коду авторизації

Розглянемо детальніше:

- method – тип запиту. Доступні запити типу GET, PUT, POST, DELETE.

Оскільки в даному випадку використовується GET-запит, то всі параметри запиту передаються не в тілі, а в URL.

- client_id – обов’язковий параметр, який предає ідентифікатор клієнта асоційований з розроблюваним додатком. Ідентифікатор клієнта яквляє собою один із ключів доступу отриманих на попередніх етапах.

- dc – необов’язковий заголовок, який передає інформацію про дата-центр з яким працює розроблюваний додаток

- redirect_uri – обов'язковий параметр, який являє собою зворотній URL, який викликає сервер авторизації.

- response_type – обов'язковий параметр, значення якого повинно бути встановлене як “code”, та вказує на те, що у відповіді потрібен код авторизації, створений сервером авторизації.

- scope – обов'язковий параметр, значення якого має містити "userContext" (для заюезпечення можливості керування та видалення ресурсів).

- state – необов'язковий параметр, який клієнт може використовувати для аутентифікації самостійно. Сервер авторизації повертає даний параметр дослівно прикріпивши його зворотньої URL-адреси клієнта.

Після виконання даного запиту в браузері, відбувається переадресація користувача на внутрішню сторінку авторизації IBM, зображену на рисунку 5.8.

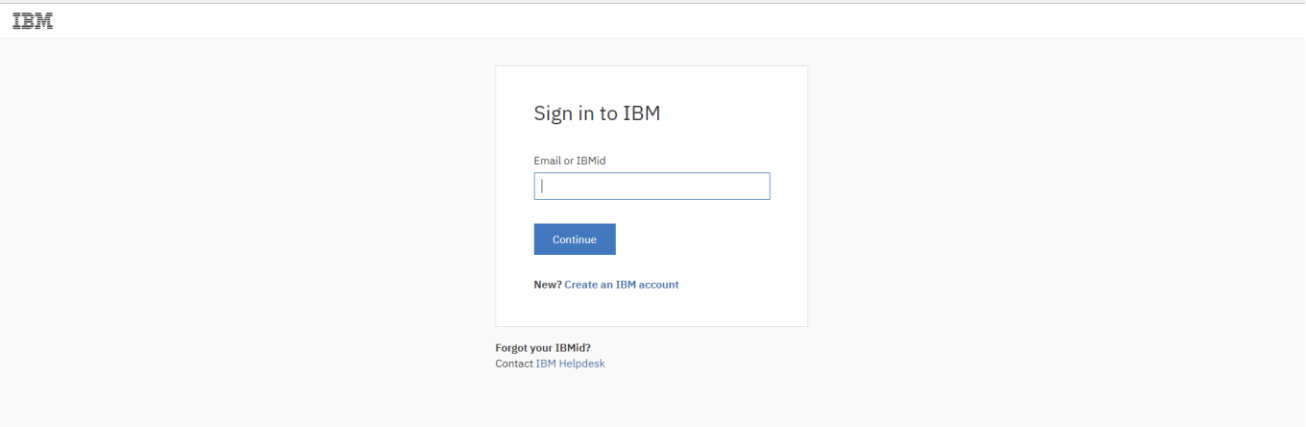

Рисунок 5.8 – Внутрішня сторінка авторизації для API клієнтів

Після введення персональних данних та успішною авторизації, сервер авторизації IBM, в автоматичному режимі робить переадресацію на URL адресу, вказану в параметрі redirect_uri запиту авторизації, доповнюючи цю адресу, своїми структурними параметрами, зображеними на рисунку 5.9. При успішні авторизації серед цих параметрів присутній потрібний код авторизації. У разі помилки серед параметрів присутні код та опис помилки.

```
https//localhost:5443/code?
 code=YOUR_AUTHORIZATION_CODE&
 state=YOUR_OPAQUE_VALUE

https//localhost:5443/code?
 error=ERROR_CODE&
 error_description=ERROR_MESSAGE&
 state=YOUR_OPAQUE_VALUE
```

Рисунок 5.9 – Структура адреси переадресації сервера авторизації IBM

Розроблюваний додаток повинен прйняти та обробити це запит та, у випадку успішної авторизації, виділити код авторизації.

5.5 Отримання токена доступу

Отриманий на попередньому етапі код авторизації валідний на протязі шістдесяти секунд. Наступним кроком авторизації є отримання токена доступу[11] на основі коду авторизації. Для цього використовують запит, структура якого зображена на рисунку 5.10.

```
var settings = {
  "async": true,
  "crossDomain": true,
  "url": "https://api.ibm.com/watsonanalytics/run/oauth2/v1/token?
grant_type=string&refresh_token=string",
  "method": "POST",
  "headers": {
    "x-ibm-client-secret": "string",
    "x-ibm-wa-data-center": "string",
    "x-ibm-client-id": "string",
    "accept": "application/json",
    "content-type": "application/x-www-form-urlencoded"
  }
}
```

Рисунок 5.10 – Структура запиту для отримання токена доступу

Розглянемо детальніше:

- method – тип запиту. Доступні запити типу GET, PUT, POST, DELETE.

Вданому випадку використовується POST-запит, що дозволяє передавати параметри як в тілі запиту так і в URL.

- grant_type – обов’язковий параметр, який повинен мати значення “authorization_code”.

- refresh_token - обов’язковий параметр, що являє собою код авторизації отриманий на попередньому етапі.

- x-ibm-client-secret – обов’язковий заголовок, який предає клієнтський секрет асоційований з розроблюваним додатком. Клієнтський секрет яквляє собою один із ключів доступу отриманих на попередніх етапах.

- x-ibm-client-id – обов’язковий заголовок, який предає ідентифікатор клієнта асоційований з розроблюваним додатком. Ідентифікатор клієнта яквляє собою один із ключів доступу отриманих на попередніх етапах.

- x-ibm-wa-data-center – необов’язковий заголовок, який передає інформацію про дата-центр з яким працює розроблюваний додаток.

- accept – обов’язковий заголовок, який передає інформацію про допустимі формати данних для передачі.

- content-type - обов’язковий заголовок, який передає інформацію про формат данних для поточного запиту.

Результатом цього запиту є токен доступу, за допомогою якого можуть виконуватись подальші клієнтські запити. Токен доступу дійсний на протязі двох годин, після чого він потребує оновлення.

6 РОЗРОБКА СИСТЕМИ АВТОМАТИЗАЦІЇ

6.1 Структура проекту

Структура проекту, зображена на рисунку 6.1, включає наступні компоненти:

- IbmSystems.Domain – бібліотека класів .Net Core, що використовується для збережння основних моделей данних проекту. Дана біблотека виокристовується проектами IbmSystems.WebApi та IbmSystems.WebUI.

- IbmSystems.Web – клієнтська біблотека .Net Core, створеня для зручної взаємодії з IBM Watson Analytics. Дана біблотека виокристовується проектами IbmSystems.WebApi та IbmSystems.WebUI.

- IbmSystems.Plugin – консольний проект .Net Core, призначений для запуску на цільовому комп’ютері, та передачі файлів виходу автотюнера на сервер.

- IbmSystems.Web.Api – проект ASP.Net Core WebAPI, призначений для взаємодії з зовнішніми системами, такими як клієнтський плагін.

- IbmSystems.Web.UI – проект ASP.Net Core MVC, веб-сайт, призначений для взаємодії з клієнтами.

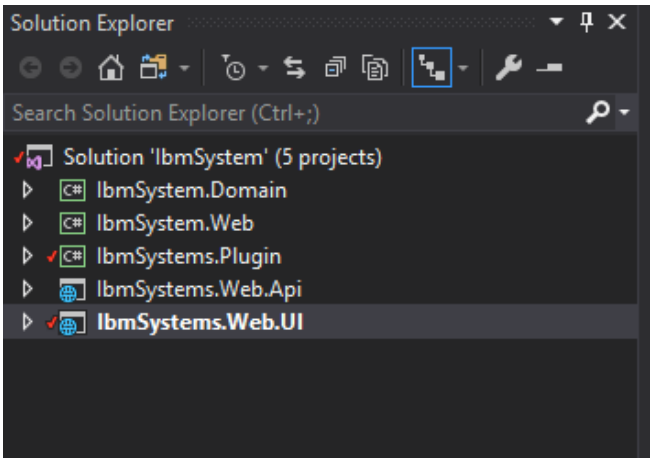

Рисунок 6.1 – Структура проекту

6.2 Розробка клієнтської бібліотеки

Для зручної взаємодії з іншими сервісами, розробкниками IBM було створено ряд клієнтських бібліотек, для декількох популярнийх мов програмування, таких як: Ruby, JavaScript, Python. Оскільки, сервіс IBM Watson Analytics існує доволі коротки проміжок часу та тільки нещодавно вийшов зі стадії бета-тестування, клієнські біблотеки існують далеко не для всіх популярних мов. Тому, одним із етапів реалізації системи автоматизації самоналаштування паралельних програм, є розробка клієнтської бібліотеки відповідно до стеку обраних технологій, а саме для платформи .Net Core.

Структура розробленої бібліотеки розбита на декілька частин, відповідно до структури IBM Watson API. Для кожної секції API створено моделі данних, зовнішній інтерфейс та його базову реалізацію, а вхідні та вихідні параметри запитів реалізовано у вигляді відповідних C# класів, для зручності роботи в серидовищі .Net Core.

Для організації доступу до IBM Watson API використовується популярна сьогодні бібліотека RestSharp[12], яка дозволяє швидко та зручно створювати, модифікувати, та виконувати HTTP-запити в синхронному та асинхронному режимах.

Для організаї процесів серіалізації та десеріалізації данних використовується популярна на сьогодні бібліотека Newtonsoft[13], яка дозволяє задавати особливості процесів серіалізації та десеріалізації моделей завдяки зручній системі атрибутів.

6.2.1 Реалізація сервісу аутентифікації

Для реалізації процесу аутентифікації, під час розробки клієнтської бібліотеки, був створений відповідний інтерфейс та його базова реалізація, зображені на рисунку 6.2.

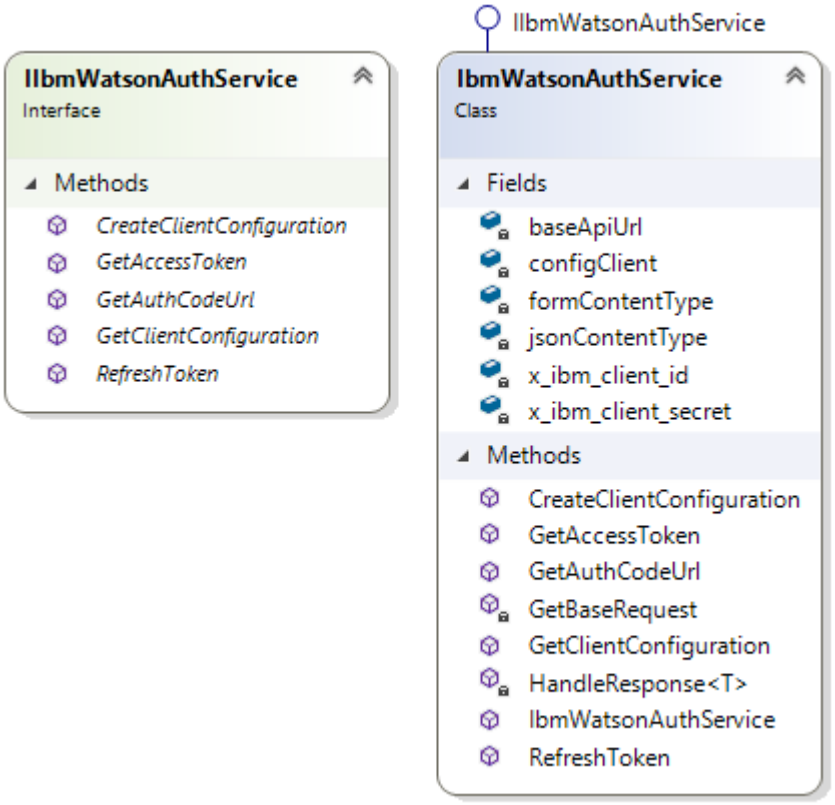

Рисунок 6.2 – Реалізація сервісу аутентифікації

Рознлянемо основні компоненти:

- CreateClientConfiguration – відповідає за створення клієнтської конфігурації на основі ключів доступу x_ibm_client_id, та x_ibm_client_secret.

- GetClientConfiguration – відповідає за отримання створеної клієнтскьої конфігурації на основі ключів доступу x_ibm_client_id, та x_ibm_client_secret.

- GetAuthCodeUrl – відповідає за отримання адреси переадресації, яка в свою чергу потрібна для отримання коду авторизації.
- GetAccessToken – відповідає за отриання токена доступу, що подрібен для подальшої роботи з іншими частинами IBM Watson API.
- RefreshToken – відповідає за оноалення раніше отриманого токена доступу.

Даний сервіс повністю охоплює процес аутентифкації в системі IBM Watson Analytics та отримання данних для подальшої роботи. Базова реалізація інтерфейсу включає реалізацію основних методів, що є віртуальними. Цільова система, за необхідності, може розширити їх функціонал шляхом наслідування та перевизначення потрібних частин.

6.2.2 Реалізація сервісу роботи з ієрархією каталогів

Для реалізації функціоналу роботи з ієрархією каталогів системи IBM Watson Analytics, під час розробки клієнтської бібліотеки, був створений відповідний інтерфейс та його базова реалізація, зображені на рисунку 6.3.

Рознлянемо основні компоненти:

- CreateFolder – відповідає за створення новго каталогу. Для успішного запиту потрібен токен доступу отриманий на етапі аутентифікації.
- DeleteFolder – відповідає за створення новго каталогу. Для успішного запиту потрібен токен доступу отриманий на етапі аутентифікації.
- GetFolderProperties – відповідає за отримання інформації про каталог. Для успішного запиту потрібен токен доступу отриманий на етапі аутентифікації. Інформація про каталог включає: ідентифікатор, назву, власника, батьківський каталог, вміст, дату створення, тип.

- UpdateFolderProperties – відповідає за оновлення інформації про каталог. Для успішного запиту потрібен токен доступу отриманий на етапі аутентифікації.

- GetRootFolder – відповідає за отрияння інформації про кореневий каталог. Для успішного запиту потрібен токен доступу отриманий на етапі аутентифікації. Інформація про каталог включає: ідентифікатор, назву, тип.

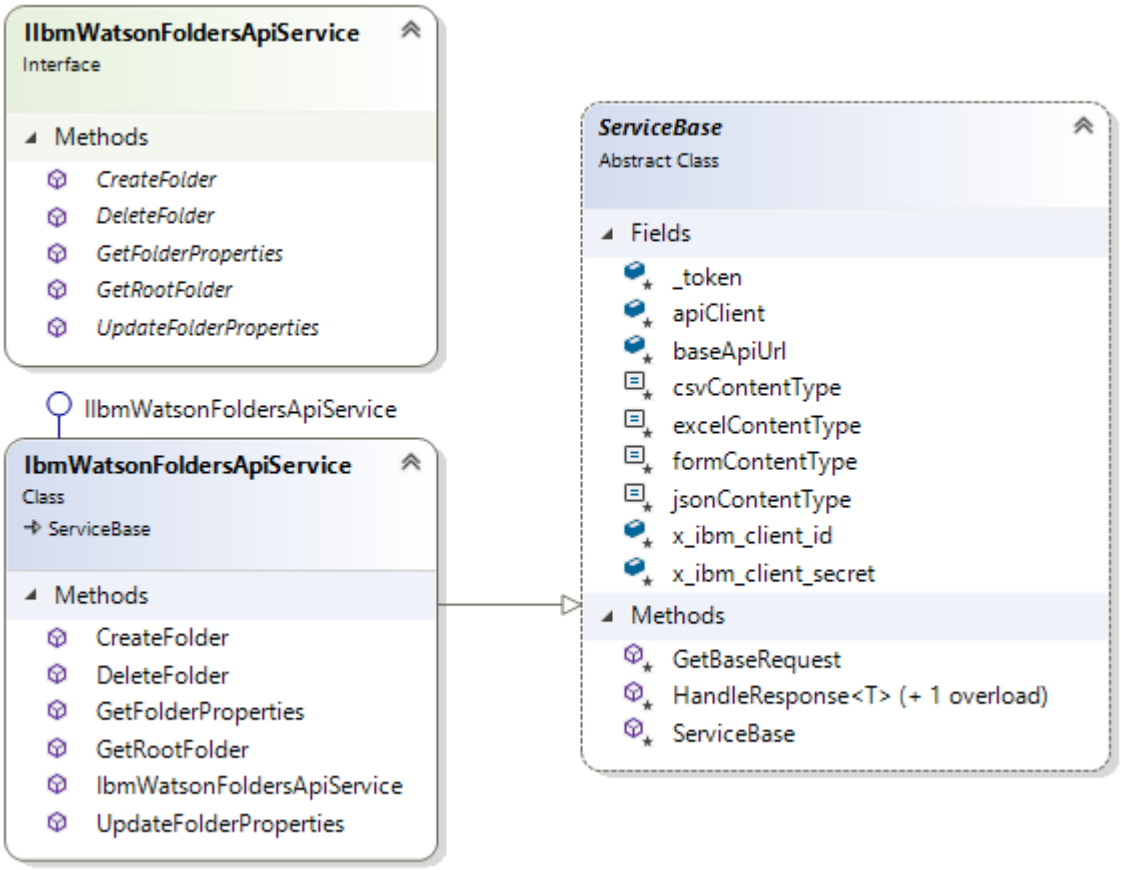

Рисунок 6.3 – Реалізація сервісу роботи з ієрархією каталогів

Даний сервіс повністю охоплює процес роботи з каталогами в системі IBM Watson Analytic. Кожен запит даного сервісу потребує передачі токена доступу для ідентифікації користувача. Базова реалізація інтерфейсу включає реалізацію основних методів, що є віртуальними. Цільова система, за

необхідності, може розширити їх функціонал шляхом наслідування та перевизначення потрібних частин.

6.2.3 Реалізація сервісу роботи з данними

Для реалізації функціоналу роботи з данними в системі IBM Watson Analytics, під час розробки клієнтської бібліотеки, був створений відповідний інтерфейс та його базова реалізація, зображені на рисунку 6.4.

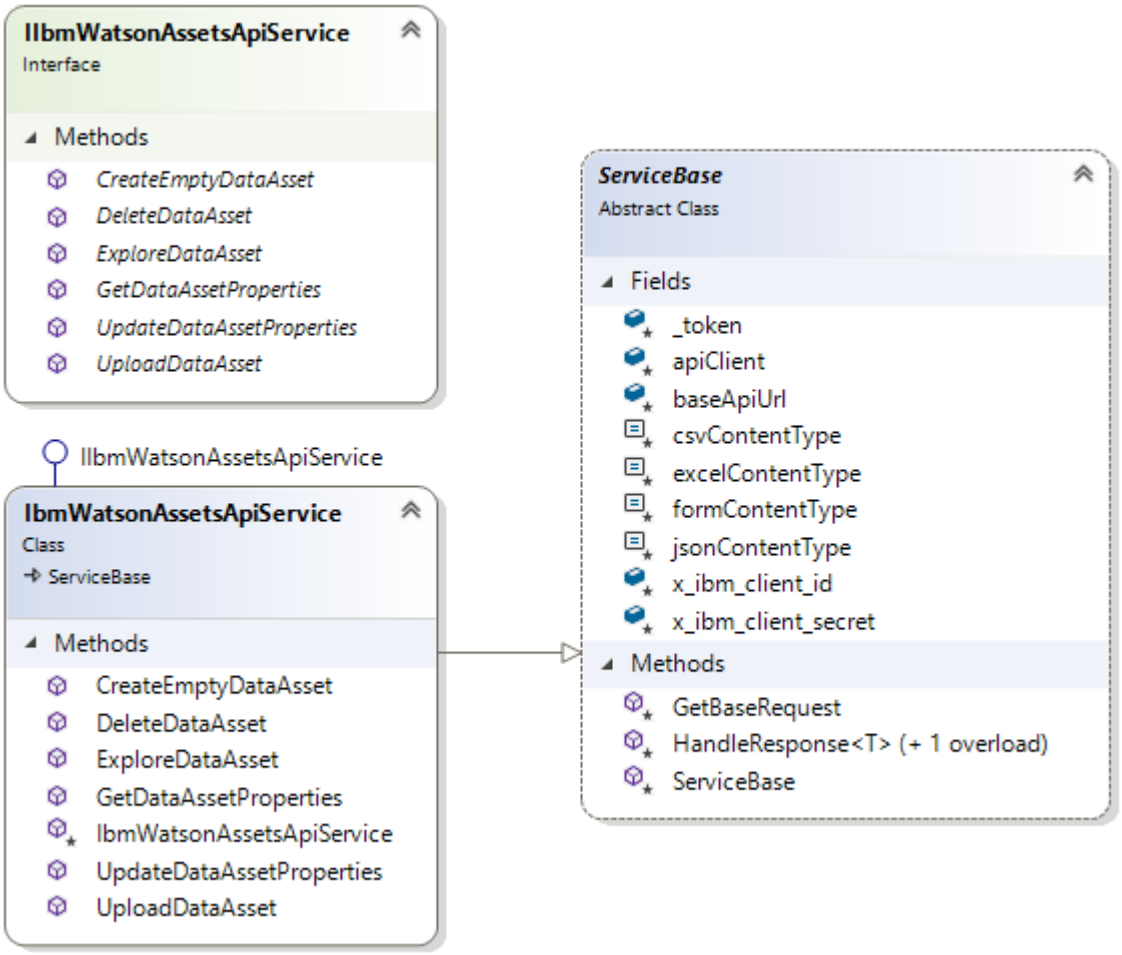

Рисунок 6.4 – Реалізація сервісу роботи з данними

Рознлянемо основні компоненти:

- CreateEmptyDataAsset – відповідає за створення новго, пустого сету данних. Для успішного запиту потрібен токен доступу отриманий на етапі аутентифікації.
- DeleteDataAsset – відповідає за видалення сету данних. Для успішного запиту потрібен токен доступу отриманий на етапі аутентифікації.
- UploadDataAsset – відповідає за завантаження текстових данних у створений сет данних. Для успішного запиту потрібен токен доступу отриманий на етапі аутентифікації.
- GetDataAssetProperties – відповідає за отримання інформації про сет данних, для успішного запиту потрібен токен доступу отриманий на етапі аутентифікації. Інформація про сет данних включає: ідентифікатор, назву, власника, каталог, дату створення, тип.
- UpdateDataAssetProperties – відповідає за оновлення інформації про сет данних. Для успішного запиту потрібен токен доступу отриманий на етапі аутентифікації.
- ExploreDataAsset – відповідає за аналіз сету данних. Для успішного запиту потрібен токен доступу отриманий на етапі аутентифікації.

Даний сервіс повністю охоплює процес роботи з данними в системі IBM Watson Analytic. Кожен запит даного сервісу потребує передачі токена доступу для ідентифікації користувача. Базова реалізація інтерфейсу включає реалізацію основних методів, що є віртуальними. Цільова система, за необхідності, може розширити їх функціонал шляхом наслідування та перевизначення потрібних частин.

6.2.4 Реалізація сервісу експорту данних

Для реалізації функціоналу експорту данних в системі IBM Watson Analytics, під час розробки клієнтської бібліотеки, був створений відповідний інтерфейс та його базова реалізація, зображені на рисунку 6.5.

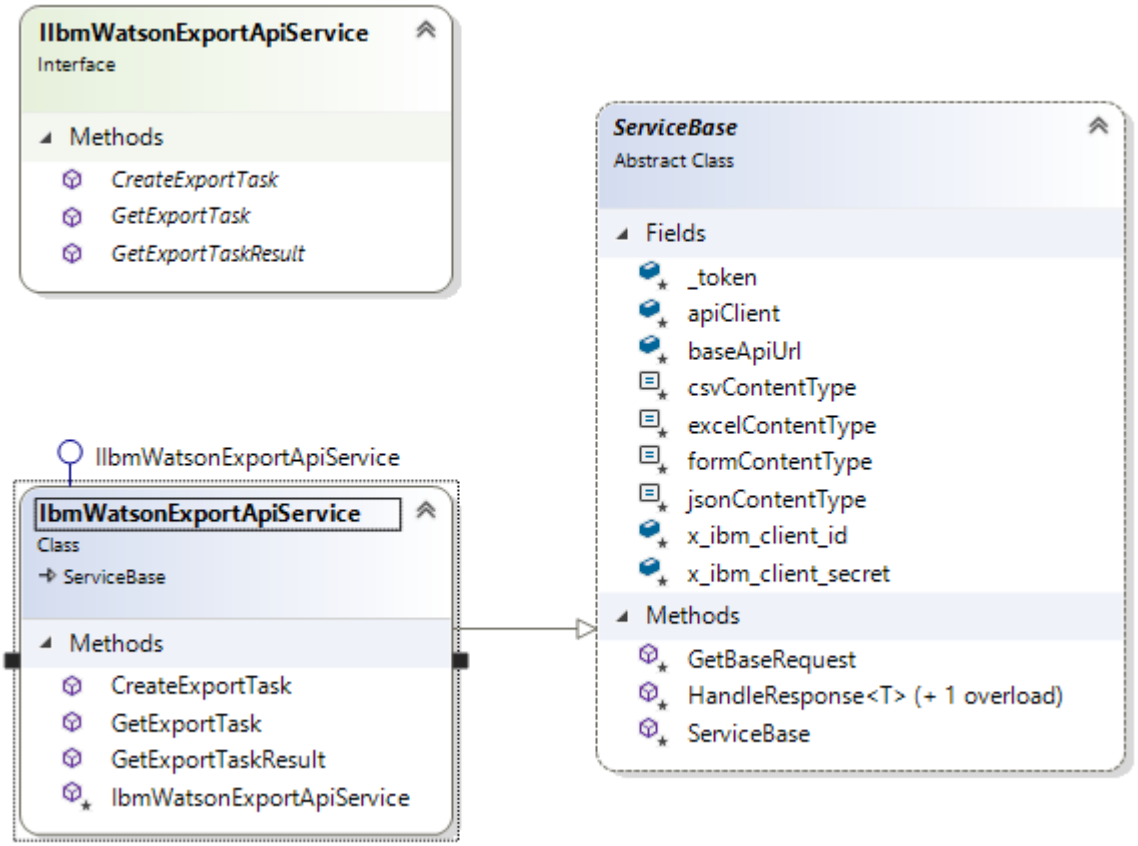

Рисунок 6.5 – Реалізація сервісу експорту данних

Рознлянемо основні компоненти:

- CreateExportTask – відповідає за створення та старту задачі по експорту данних. Для успішного запиту потрібен токен доступу отриманий на етапі аутентифікації.

- GetExportTask – відповідає за отримання інформації про створену задачу експорту. Для успішного запиту потрібен токен доступу отриманий на етапі аутентифікації.

- GetExportTaskResult – відповідає за експорт данних після успішно завершеної задачі експорту. Для успішного запиту потрібен токен доступу отриманий на етапі аутентифікації.

Даний сервіс повністю охоплює процес експорту в системі IBM Watson Analytic. Кожен запит даного сервісу потребує передачі токена доступу для ідентифікації користувача. Базова реалізація інтерфейсу включає реалізацію основних методів, що є віртуальними. Цільова система, за необхідності, може розширити їх функціонал шляхом наслідування та перевизначення потрібних частин.

6.3 Розробка веб-сайту для взаємодії з клієнтами

Відповідно до обраного стеку технологій для реалізації взаємодії з клієнтами створено веб-сайт на базі технології ASP.Net Core MVC.

Проект має структуру, типову для проектів даного типу , зображену на рисунку 6.6. Кожен контроллер відповідає роботу відповідною частиною IBM Watson Analytics API та використовує компоненти створеної на попередньому етапі клієнтської біблотеки. Всі залежності реалізовані на основі інтерфейсів та технології Dependency Injection, що дозволяє легко розширювати та модифікувати компоненти системи.

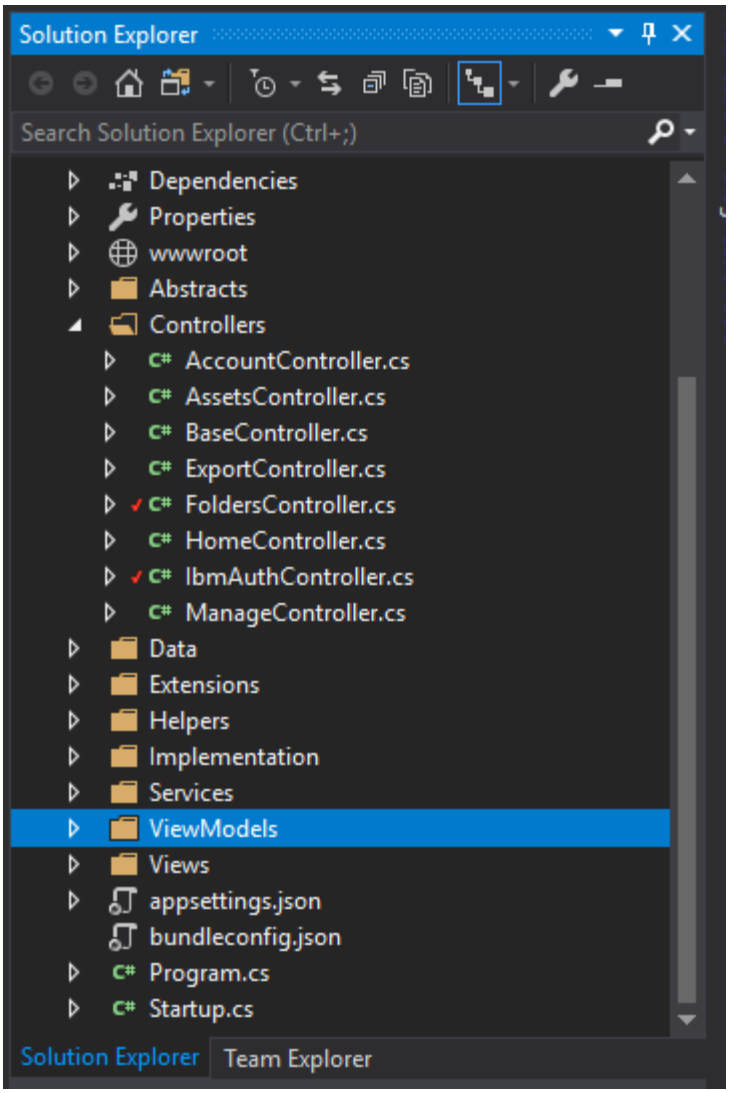

Рисунок 6.6 – Структура MVC проекту

Розглянемо детальніше структуру проекту:

- Abstracts – інтерфейси, використовувані в системі та зареєстровані в DI-контейнері.
- Implementation – реалізації інтерфейсів, використовуваних в системі та зареєстрованих в DI-контейнері.
- Controllers – контроллери, одна з ключових частин MVC проекту, що відповідає за обробкі вхідних запитів та повернення результату. Використовує DI-контейнер для отримання потрібних залежностей.

- Data – набір міграційних файлів, використовуваних для міграцій бізи данних
- Extensions – класи, які надають методи розшираення для потрібних компонентів.
- Helpers – допоміжні класи, для реалізації певних, базових операцій.
- ViewModels – класи, призначені для передачі інформації між контроллером та представленням.
- Views – представлення, призначені для відображення HTML сторінки на основі данних, отриманих з контроллера.

6.4 Взаємодія з базою даних

Взаємодія з базою даних проводиться за допомогою ORM технології Entity Framework. Яка дозволяє автоматично зв’язувати класи мови C# з таблицями реляційної бази даних.

Для взаємодії з базою даних через Entity Framework потрібен спеціалізований клас, наслідуваний від Microsoft.Data.Entity.DbContext – контекст даних. Властивості цього класу з типом DbSet, представляють колекцію об’єктів, яка відповідає певній таблиці в базі даних. При цьому за замовчуванням назва властивості повинна відповідати назві класу моделі у формі множини, згідно з правилами англійської мови.

Одним із найбільш часто використовуваних шаблонів проектування при роботі з даними є шаблон Repository.

Repository призначений для створення абстрактного, проміжного рівня між рівнем доступу до даних та рівнем бізнес логіки. Тобто він дозволяє абстрагуватися від конкретного підключення до джерела даних, з яким працює програма, і являється

проміжною ланкою між класами, що безпосередньо працюють з даними та рештою програми.

Це забезпечує значне полегшення підтримки, зміни чи модернізації програми та збільшує можливість повторного використання коду. Таким чином шаблон Repository додає програмі гнучкості при роботі з джерелами даних.

Схематично принцип дії шаблону в середовищі ASP.Net продемонстрований на рисунку 6.7.

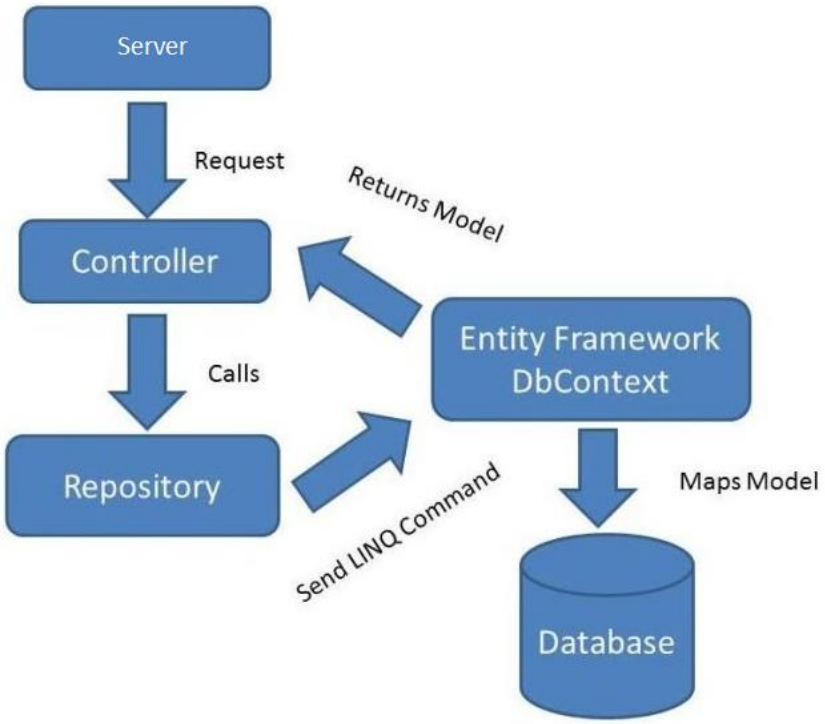

Рисунок 6.7 – Принцип дії шаблону Repository

Для реалізації шаблону Repository створено узагальнений CRUD інтерфейс IRepository<T, TKey> (рисунок 6.8), де T – клас моделі з якою працює репозиторій, TKey – тип первинного ключа.

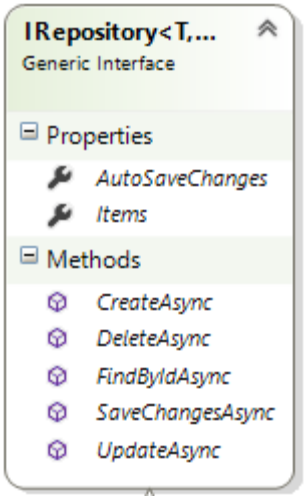

Рисунок 6.8 – Інтерфейс IRepository

CRUD інтерфейс – це інтерфейс, який представляє чотири базові операції для роботи із персистентними сховищами даних: створення, зчитування, редагування, видалення.

7 ПРАКТИЧНИЙ ЕКСПЕРИМЕНТ

Для проведення практичного експерименту експлуатації системи був застований автотюнер, розроблений експертами Національної Академії Наук України, TuningGenie[14], що використовує терми[15] у вихідному коді програми для визначення конфігурації. Тестовий датасет, згенерований на основі роботи автотюнера, являє собою csv-файл, з множиною параметрів конфігурації, виділених автотюнером та заміром часових показників для кожної конфігурації.

Для початку роботи, стартовій сторінці, зображеній на рисунку 7.1, користувач повинен натиснути кнопку Connect та пройти процес авторизації в системі IBM.

Рисунок 7.1 – Стартова сторінка веб-сайту

Для завантаження отриманих автотюнером файлів використовується створений плагін, результат роботи якого зображений на рисунку 7.2.

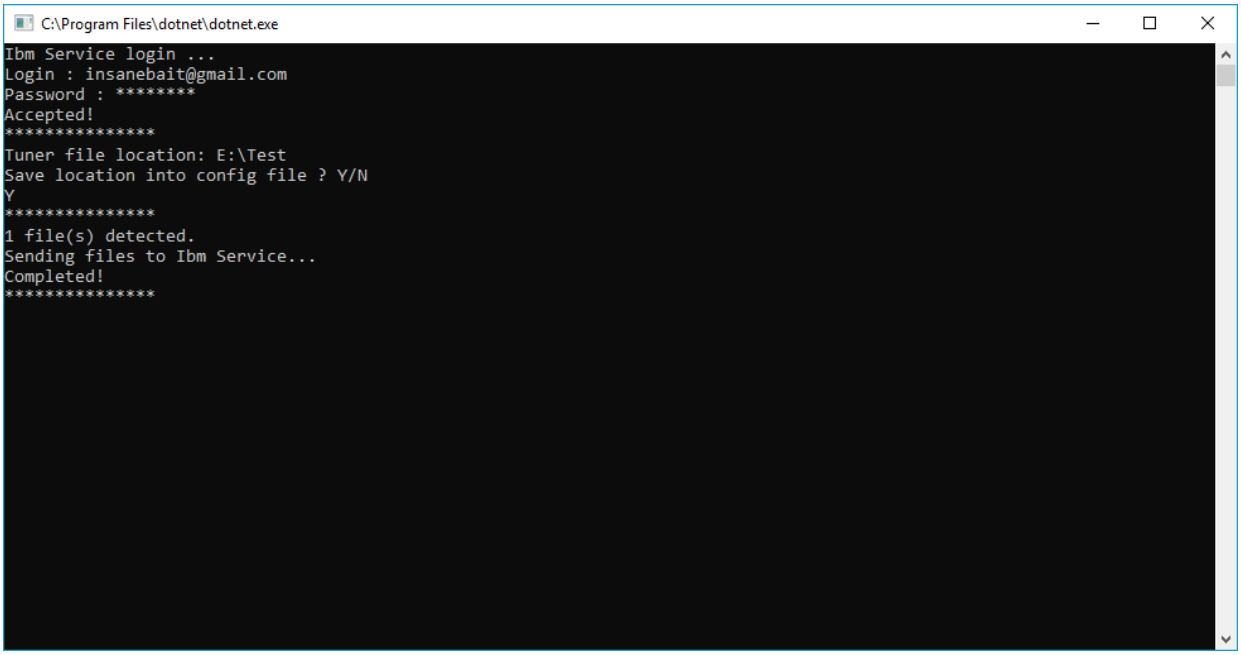

Рисунок 7.2 – Результат роботи плагіну, для завантаження файлів

Після успішної авторизації та успішного завантаження файлів користувач може переглянути вміст свого каталогу, зображений на рисунку 7.3.

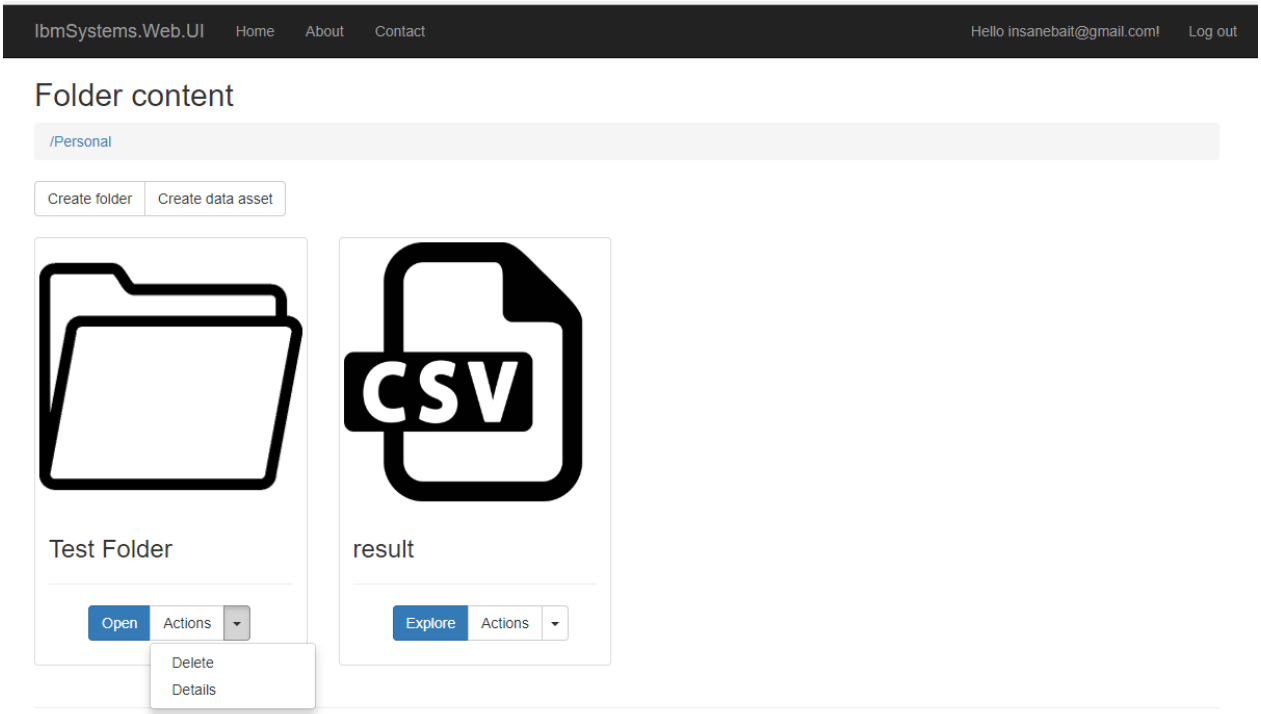

Рисунок 7.3 – Вміст каталогу користувача

Набір кнопок Actions дозволяє виконати певний перелік операцій над папкою чи данними. Наприклад отримати інформацію про набір данних, зображену на рисунку 7.4. Для наборів данних доступна додаткова операція Export, що дозволяє завантажити набір данних у вигляді файлу.

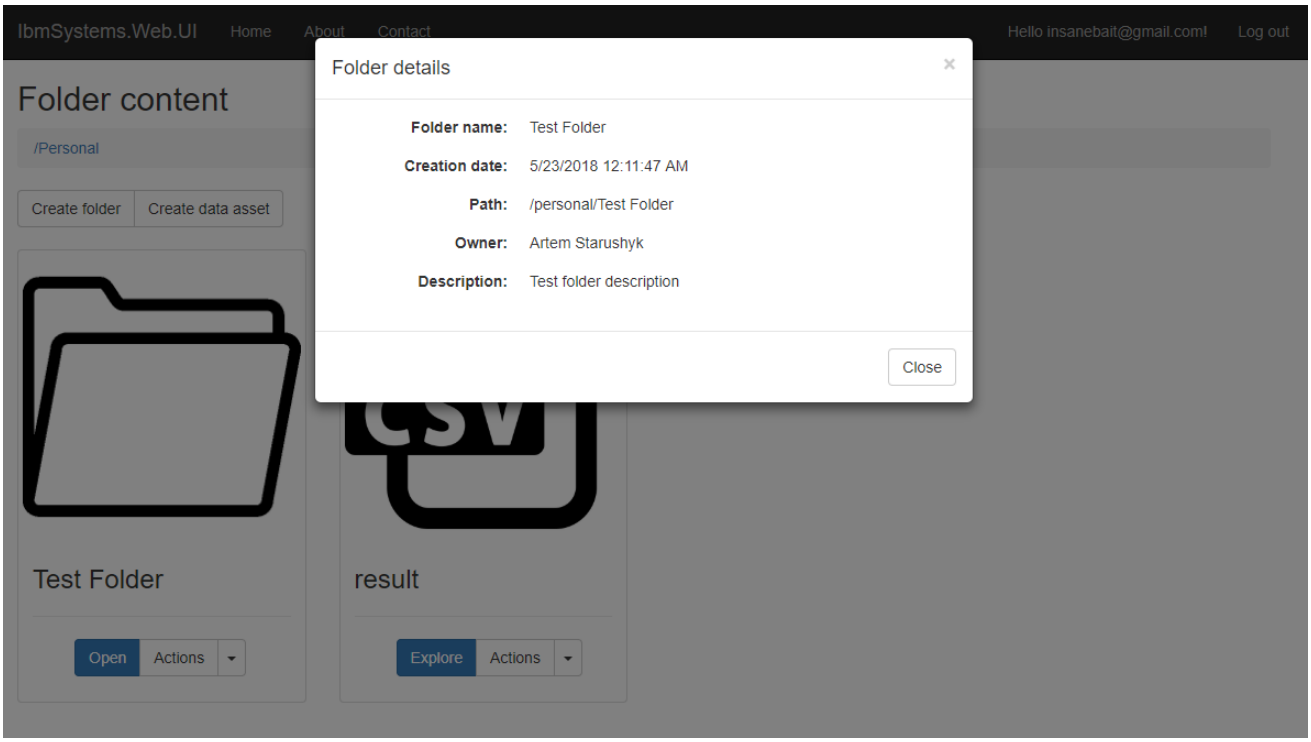

Рисунок 7.4 – Перегляд інформації про каталог

Для початку аналізу данних потрібно натиснути кнопку Explore. Когнітивна взаємодія з IBM Watsons Analytics зображена на рисунках 7.5, 7.6. Відповіді на питання, поставлені природною мовою візуалізовані у вигляді графіків та таблиць, на основы яких можна робити висновки про ефективність оброблених автотюнером конфігурацій.

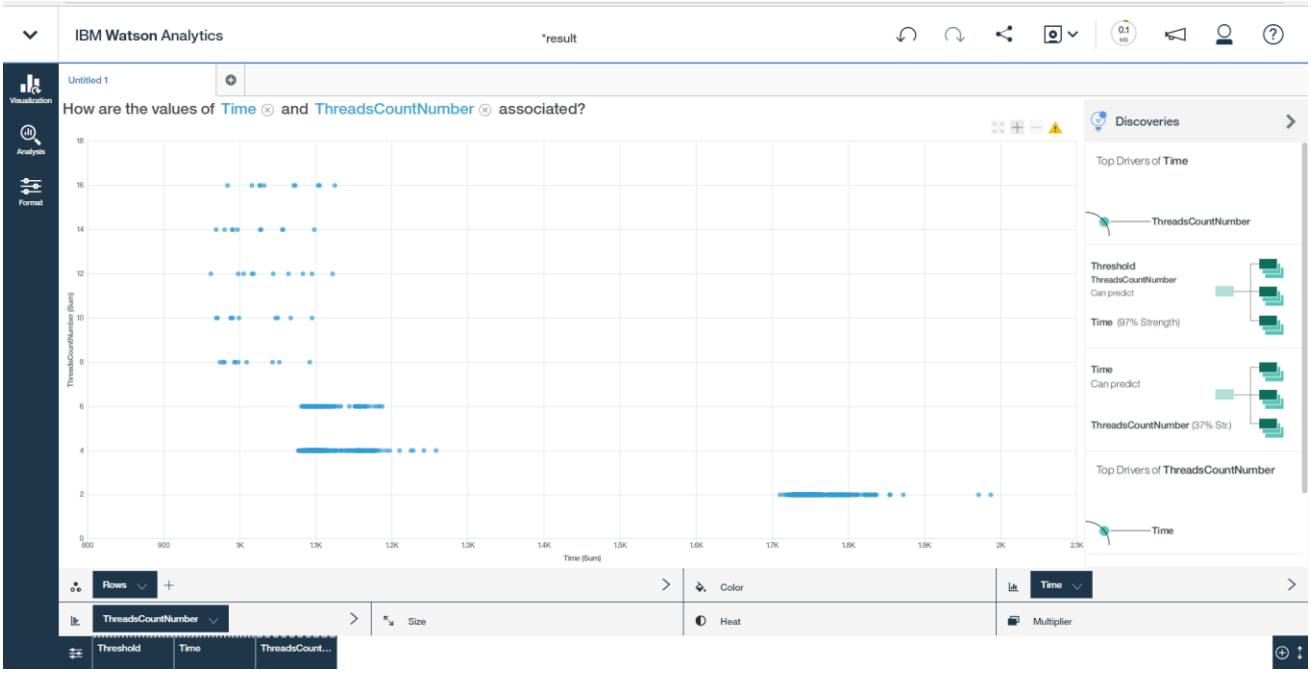

Рисунок 7.5 – Асоція кількості потоків та часу виконання

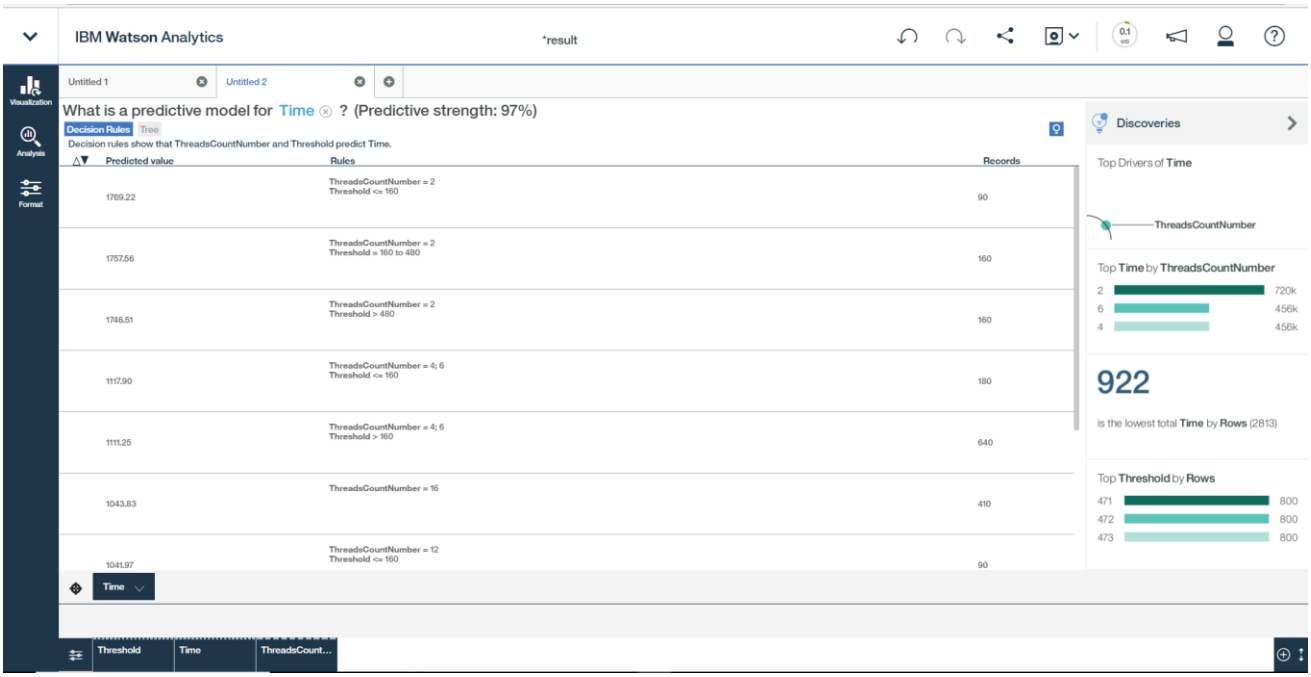

Рисунок 7.6 – Виділений набір правил для датасету

Також, під час роботи IBM Watson Analytics інколи видає підказки про знайдені залежності, як зображено на рисунку 7.7

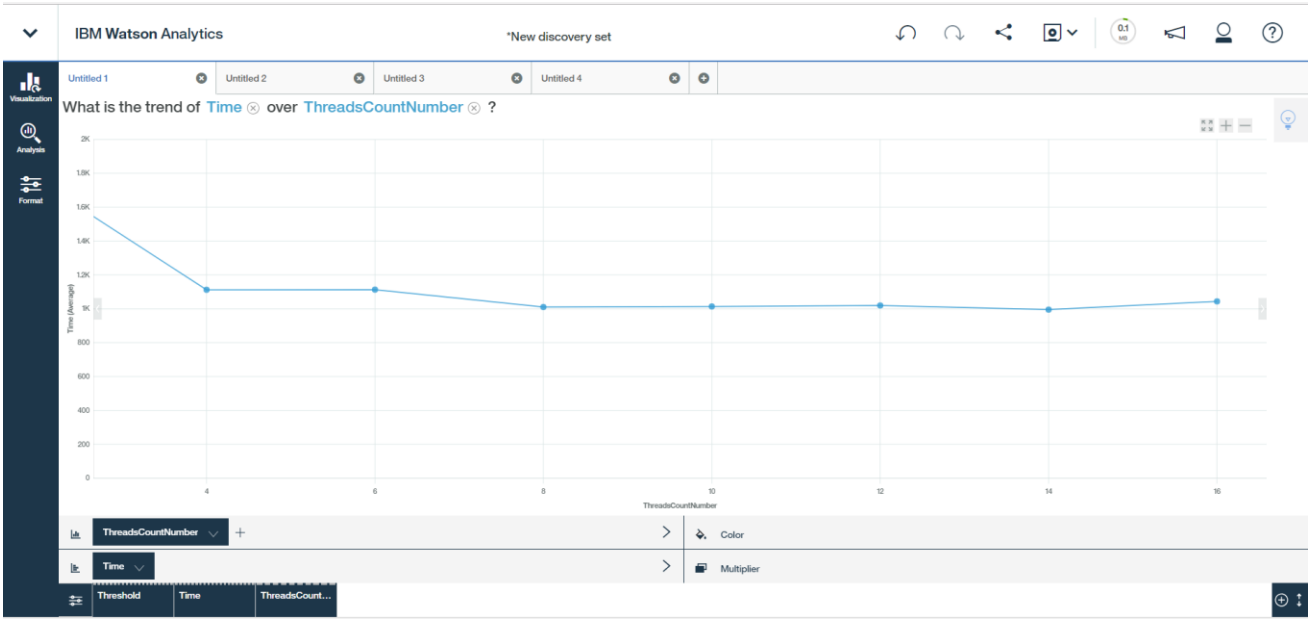

Рисунок 7.7 – Підказка, яка вказує на явну кореляцію часу виконання від кількості потоків

7 СТАРТАП

7.1 Опис ідеї проекту

Метою розділу є розробка стартап-проекту системи автоматизації процесу автотюнінгу паралельних програм для цільової платформи. Зміст ідеї, можливі напрямки застосування, осносні переваги, які зможе отримати користувач предсталено у Таблиці 7.1.1.

Таблиця 7.1.1 – Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
	Розробка програмного забезпечення	Аналіз та виявлення вузьких місць програмного забезпечення на етапі розробки.
	Тестування програмного забезпечення	Аналіз та виявлення вузьких місць програмного забезпечення на етапі тестування.
	Інтеграція програмного забезпечення	Аналіз поведінки програмного забезпечення в цільовому серидовищі дляя поленшення процесу інтеграції.

	Бізнес-аналіз	Збір збереження та аналіз метаданних, для дослідження життєвого циклу та формування кроків подільшого розвитку програмного забезпечення

Система автоматизації процесу автотюнінгу паралельних програм для цільової платформи призначена в основному для підтримки програмного забезпечення на всіх етапах його існування, починаючи від розробки та закінчуючи введенням в експлуатацію. В результаті, за рахунок постійного збору статистичних даних розробник може коригувати відповідні етапи та уникнути можливих проблем у майбутньому.

На ринку конкретні конкуренти відсутні, присутні тільки замінники у вигляді автоматизованих систем аналізу програмного забезпечення. Конкурент 1 - Visual Studio Diagnostic Tool, Конкурент 2 - Google Performance Tool.

Таблиця 7.1.1 – Визначення сильних, слабких та нейтральних характеристик ідеї проекту

№	Техніко-економічні характеристики ідеї	(потенційні) товари/концепції конкурентів		
		Мій проект	Конкурентнт 1	Конкурентнт 2
1	Технічі: швидкість роботи;	Висока щвидкість аналізу роботи за рахунок	Середня швидкість аналізу,	Середня швидкість аналізу, оскільки

	точність результатів; можливість збереження результатів; режими доступу;	використання системи аналізу данних IBM Watsons Analytics. Висока точність за рахунок великої кількості експериментів. Можливість збереження результатів. Доступ через інтернет.	оскільки він проводиться на цільовому комп'ютері. Висока точність. Збереження результатів в рамках однієї сесії. Доступ без інтернету.	він проводиться на цільовому комп'ютері. Висока точність Збереження результатів в рамках однієї сесії. Доступ без інтернету.
2	Економічі: ціна за підписку	Середня в межах галузі	Середня в межах галузі	Середня в межах галузі
3	Надійність	Не обмежений термін діїї	Не обмежений термін діїї	Не обмежений термін діїї

Сильними сторонами системи відносно конкурентів є швидкість аналізу данних, можливість збереження результатів. До слабких сторін можна віднести режим доступу: система доступна тільки при підключенні до мережі інтернет. Конкуренти доступні без підключення до інтернету. Нейтральними сторонами є необмежений термін дії та ціна.

7.2 Технологічний аудит ідеї проекту

В межах даного підрозділу проводиться аудит технологій, за допомогою яких можна реалізувати ідею проекту (технології створення товару).

Визначення технологічної здійсненності ідеї проекту передбачає аналіз таких складових (табл. 5.2.1):

Таблиця 7.2.1 – Технологічна здійсненність ідеї проекту

Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій
	Технологія 1. Самостійна розробка всіх компонентів програми без використання сторонніх бібліотек на мові C#.	Потребує розробки	Немає у відкритому доступі
	Технологія 2. Розробка компонентів програми з використанням функціоналу існуючих	Наявні	У відкритому доступі з відкритим кодом

	відкритих бібліотек для взаємодії з REST API (REST Sharp, Rest Client).		
	Технологія 3. Розробка компонентів програми з використанням функціоналу платформи Microsoft Azure для інтеграції та взаємодії з іншими сервісами.	Наявні	Платні, з закритим кодом

Обрана технологія реалізації ідеї проекту: Технологія 2. Використання даної технології зменшить кількість розроблюваних блоків функціоналу та матеріальні затрати на реалізацю проекту.

7.3 Аналіз ринкових можливостей запуску стартап-проекту

Аналіз попиту: наявність попиту, обсяг, динаміка розвитку ринку представлено у таблиці 7.3.1

Таблиця 7.3.1 – Попередня характеристика потенційного ринку стартап-проекту

№ п/п	Показники стану ринку (найменування)	Характеристика
1	Кількість головних гравців, од	5
2	Загальний обсяг продаж, грн/ум.од	10000 за рік
3	Динаміка ринку (якісна оцінка)	Зростає
4	Наявність обмежень для входу (вказати характер обмежень)	Відсутні
5	Специфічні вимоги до стандартизації та сертифікації	Відсутні
6	Середня норма рентабельності в галузі (або по ринку), %	60

Ринок є привабливий для входу та потребує новітніх засобів автоамтизації процесу автотюнінгу паралельних програм.

Потенційні групи клієнтів, їх характеристика, та орієнтовний перелік вимог до товару для кожної групи представлений в таблиці 7.3.2.

Таблиця 7.3.2 – Характеристика потенційних клієнтів стартап-проекту

Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару

Автоматизація процесу автотюнінгу паралельних програм для цільової платформи	Розробники	Використання в процесі розробки програмного забезпечення для визначення «вузьких місць»	Швидкість роботи, точність результатів, отримання потрібних для розробки данних
	Тестувальники	Використання в процесі тестування для визначення відповідних тест- кейсів	Швидкість роботи, проста інтеграція, отримання потрібних для тестування данних
	Аналітики	Використання в процесі аналітики роботи програмного забезпечення для визначення подальшого шляху розвитку	Швидкість роботи, обсяг інформації, отримання потрібних для аналізу данних

Фактори, що сприяють ринковому впровадженню проекту, та фактори, що йому перешкоджають представлені в таблицях 7.3.3 та 7.4.4 відповідно.

Таблиця 7.3.3 – Фактори загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1	Збільшення ціни на розробку	Збільшення витрат на розробку, розширення технологічної бази.	Зміна цінової політики, додавання ексклюзивного функціоналу, перехід на більш перспективні та ефективні технології
2	Поява нових гравців на ринку	Появав на ринку конкурентів з анлогічними продуктами	Реклама, удосконалення продукту, додавання ексклюзивного функціоналу.

Таблиця 7.3.4 – Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
1	Попит на програмний продукт	Розвиток паралельних систем сприяє збільшенюю попиту на автоматизовані системи автотюнінгу.	Розробка та підтримка унікального нового надійного функціоналу
2	Зменшення ціни на розробку	Ефективніше використання наявних ресурсів.	Збільшення штату компанії, збільшення швидкості розробки.

Загальні риси конкуренції на ринку представлені в таблиці 7.3.5.

Таблиця 7.3.5 – Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
1.Тип конкуренції - олігополія	Домінує мала кількість компаній	Розробка унікального функціоналу, потужна рекламна кампанія.
2. За рівнем конкурентної боротьби : конкурентне серидовище міжнародне	Боротьба ведеться на міжнародному ринку	Локалізація продукту, розвиток рекламної кампанії.
3. За галузевою ознакою - внутрішньогалузева	Продукт використовується у сфері розробки та підтримки програмного забезпечення	Розробка унікального функціоналу, щоб повністю задовольнити користувачів
4. Конкуренція за видами товарів: - товарно-родова	Конкуренція між різними видами систем аналізу паралельних програм.	Розробка унікального функціоналу.
5. За характером конкурентних переваг - не цінова	Ціна не є основним фактором конкуренції. Конкрунтні переваги продукту засновані на його	Впровадження нових функціональних можливостей

	функціональних можливостя.	
6. За інтенсивністю - не марочна	Продукт прив'язаний до марки IBM, та роль цієї прив’язаності незначна, так як використовуються публічні можливості технологій IBM.	Інтеграція нових можливостей та різних систем.

Аналіз конкуренції в галузі за М. Портером представлений в таблиці 7.3.6.

Таблиця 7.3.6 – Аналіз конкуренції в галузі за М. Портером

	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
Складові аналізу	Відсутні	Присутні	Відсутні	Споживачі диктують умови. Аналіз та покращення роботи паралельних програм є важливим аспектом життєвого циклу програмного забезпечення	Присутні, зокрема Конкуренти 1,2.

Висновки:	Прямі конкуренти на ринку відсутні.	Присутні. Вихід на ринок можливий.	Постачальники відсутні, для розробки потрібні спеціалісти сфери розробки програмного забезпечення.	Клієнти диктують умови. Якісний, дешевий, швидкодіючий продукт.	Обмежень для роботи на ринку товари-замінники не представляють

Робота на ринку можлива, відсутні прямі конкуренти, а ринок потребує системи даного типу.

На основі аналізу конкуренції, проведеного в таблиці 7.3.6, а також із урахуванням характеристик ідеї проекту (табл. 7.1.2), вимог споживачів до товару (таблиці 7.3.2) та факторів маркетингового середовища (таблиці №№ 7.3.3-7.3.4) визначено перелік факторів конкурентоспроможності. Представлених в таблиці 7.3.7.

Таблиця 2 – Обґрунтування факторів конкурентоспроможності

№ п/п	Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)
1	Швидкість аналізу показників	Аналіз проводиться за допомогою системи IBM Watson Analytics, що значно пришвидшує роботу.
2	Різносторонній аналіз показників	Система IBM Watson Analytics проводить різносторонній аналіз

		показників для визначення прихованих кореляцій.
3	Точність результатів	Висока точність результатів за рахунок великої кількості експериментів
4	Збереження резльтатів	Збереження результатів роботи як з поточної, так і з минулих сесій.

За визначеними факторами конкурентоспроможності (табл. 7.3.7) проведений аналіз сильних та слабких сторін стартап-проекту представлений в таблиці 7.3.8., де К – конкурент, П – стартап-проект.

Таблиця 7.3.8 – Порівняльний аналіз сильних та слабких сторін проекту

№ п/п	Фактор конкурентоспроможності	Бал	Рейтинг товарів-конкурентів						
			–3	–2	–1	0	+1	+2	+3
1	Швидкість аналізу	20			К2	К1			П
2	Різносторонній аналіз	19				К2	К1	П	
3	Точність результатів	18				К1	К2		П
4	Збереження результатів	17		К1	К2				П
5	Доступність	15			П			К1	К2

SWOT-аналіз (матриця аналізу сильних (Strength) та слабких (Weak) сторін, загроз (Troubles) та можливостей (Opportunities) на основі виділених ринкових загроз та можливостей, та сильних і слабких сторін приведено в таблиці 7.3.9.

Таблиця 7.3.93 – SWOT- аналіз стартап-проекту

Сильні сторони: швидкість аналізу,	Слабкі сторони: доступність

точність аналізу, можливість збереження результатів.	
Можливості: збільшення попиту на товар, зменшення коштів на розробку.	Загрози: збільшення ціни на розробку, нові гравці на ринку

На основі SWOT-аналізу розроблено альтернативи ринкової поведінки для виведення стартап-проекту на ринок та орієнтовний оптимальний час їх ринкової реалізації з огляду на потенційні проекти конкурентів, що можуть бути виведені на ринок. Результати приведені в таблиці 7.3.10.

Таблиця 4.10 – Альтернативи ринкового впровадження стартап-проекту

№ п/п	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	Розробка базової версії продукту, з забезпеченням запланованого функціоналу.	висока	До року
2	Розробка версії продукту з використанням власної системи аналізу даних.	низька	До трьох років
3	Створення оффлайн версії продукту	середня	До року

4	Впровадження найновіших ефективних технологій	низька	До дев'яти міяців

Обрана альтернатива під номером один, оскільки її реалізація є найбільш вірогадна та з малими сроками.

7.4 Розробка ринкової стратегії проекту

Розроблення ринкової стратегії першим кроком передбачає визначення стратегії охоплення ринку. Опис цільових груп потенційних споживачів представлений в таблиці 7.4.1.

Таблиця 5.1 – Вибір цільових груп потенційних споживачів

Опис профілю цільової групи потенційних клієнтів.	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
Компанії, що спеціалізуються на розробці програмного забезпечення.	Споживачі готові сприйняти продукт.	Попит високий в цільовому сегменті	Конкуренція з товарами-замінниками	Продукт простий у впровадження його на ринок

Обранастратегія концетрованого маркетингу, оскільки присутній один цільовий сегмент.

Визначення базової стратегії розвитку представлено в таблиці 7.4.2

Таблиця 7.4.26 – Визначення базової стратегії розвитку

Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
Розробка базової версії функціоналу, з подальшим покращенням.	Стратегія розширення первинного попиту	Конкурентами є лише товари-замінники. Розроблювана система має переваги в швидкості, точності та збереженню данних.	Стратегія диференціації

Вибір стратегії конкурентної поведінки представлений в таблиці 7.4.3.

Таблиця 7.4.37. Визначення базової стратегії конкурентної поведінки

Чи є проект «першопрохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару	Стратегія конкурентної поведінки*

		конкурента, і які?	
Так	Шукати нових та забирати споживачів у конкурентів	Ні	Стратегія лідера

На основі вимог споживачів з обраних сегментів до постачальника (стартап-компанії) та до продукту (таблиця 7.3.2), а також в залежності від обраної базової стратегії розвитку (таблиця 7.4.2) та стратегії конкурентної поведінки (таблиця 7.4.3) розроблена стратегія позиціонування представлена в таблиці 7.4.4.

Таблиця 7.4.4 – Визначення стратегії позиціонування

№ п/п	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)
	Швидкість, точність, унікльний функціонал	Стратегія диференціації	- швидкість - точність - збереження результатів	Швидкість, точність, сучасність

7.5 Розроблення маркетингової програми стартап-проекту

Таблиця 7.5.18 – Визначення ключових переваг концепції потенційного товару

№ п/п	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити
1	Швидкодія	Швидкодія за рахунок використання системи аналізу данних IBM Watsons Analytics.	Процес аналізу швидший ніж у конкурентів.
2	Точність	Точність за рахунок великої кількості експериментів.	Диференціація експериментів за певними показниками та визначення найкразих по певному показнику.
3	Збереження результатів	Збереження результатів різної кількості сессій використання.	Конкурентні рішення зберігають дані тільки однієї сесії роботи, або кількох останніх.
4	Різносторонній аналіз	IBM Watsons Analytics дозволяє знайти приховані кореляції.	Конкрунти працюють тільки з часовими показниками.

Трирівнева маркетингова модель товару представлена в таблиці 7.5.2.

Таблиця 7.5.2 9– Опис трьох рівнів моделі товару

<table>
<tr><td>Рівні товару</td><td colspan="3">Сутність та складові</td></tr>
<tr><td>І. Товар за задумом</td><td colspan="3">Автоматизована система автотюнінгу паралельних програм для цільової платформи.
Можна виділити наступні вигоди від використання продукуту:
- Швидкість аналізу данних;
- Різносторонній аналіз;
- Точність;
- Збереження попредніх результатів;</td></tr>
<tr><td rowspan="7"></td><td>Властивості/характеристики</td><td>М/Нм</td><td>Вр/Тх /Тл/Е/Ор</td></tr>
<tr><td>1. Висока швидкодія</td><td>М</td><td>Тх/Тл</td></tr>
<tr><td>2. Висока точність</td><td>М</td><td>Тх/Тл</td></tr>
<tr><td>3. Різносторонній аналіз</td><td>М</td><td>Тх/Тл</td></tr>
<tr><td>4. Довговічність</td><td>М</td><td>Вр/Тл</td></tr>
<tr><td colspan="3">Якість: відповідає нормам розробки програмного забезпечення.</td></tr>
<tr><td colspan="3">Пакування: Продукт представлений у вигляді веб-сайту та плагіну. На сайті міститься:
• загальна назва продукту, власна назва;
• опис продукту;
• функції, які представлені;
• інструкція для корестування;</td></tr>
</table>

	• контакти для зв’язку з розробником; • підтримка для клієнтів
	Марка: ПП «TuneWare»

Проект буде захищений як інтелектуальна власність.

Наступним кроком є визначення цінових меж, якими необхідно керуватись при встановленні ціни на потенційний товар. Результат представлений в таблиці 7.5.3.

Таблиця 7.5.310 – Визначення меж встановлення ціни

Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар
15 тис. грн	Відсутні	600 тис. грн	12 тис. грн – 20 тис. грн

Визначення оптимальної системи збуту представлено в таблиці 7.5.4.

Таблиця 7.5.4 – Формування системи збуту

Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
Реєстрація на сайті, оплата продукту, відсилання на електрону пошту ключа активації.	Доставка на електрону пошту ключа активації.	Виробни - споживач	Web-сайт

Останньою складової маркетингової програми є розроблення концепції маркетингових комунікацій, що спирається на попередньо обрану основу для позиціонування, визначену специфіку поведінки клієнтів. Результат представлений в таблиці 7.5.5.

Таблиця 7.5.5 – Концепція маркетингових комунікацій

Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
Ретельно обирають систему	Веб-сайт, телефон, месенджер	Підтримка, індивідуальний підхід, постійне оновлення	Донести переваги продукту до клієнта	Автоматизований автотюнінг паралельних програм.

ВИСНОВКИ

Різноманіття та складність сучасних паралельних архітектур практично унеможливлює створення паралельних програм, ефективних на будь-якій із них. Кожен паралельний програмний комплекс потребує певної конфігурації для конкретного серидовища виконання для досягнення максимальної ефективності роботи.

Методологія автотюнінгу дозволяє значно скоротити етап оптимізації паралельноих програм, тим самим покращити якість програмного забеспечення та зекономити час його розробки.

Емпіричний аналіз результатів системою IBM Watsons Analytics дозволяє знаходити явні та не явні кореляції між множинами параметрів та результатами швидкодіїі, та подати ці дані у зрозумілому, зручному форматі. Також мінімізується процес аналізу даних в загальному, оскільки, як спеціалізована платформа, IBM Watsons Analytics використовує ефективні алгоритми аналізу та значні обчислювальні ресурси. В свою чергу зручний та функціональний відритий API дозволяє використовувати можливості системи для створення власних додатків.

Реалізована система автоматизації автотюнінгу паралельних програм для цільової програми з використанням системи аналізу даних IBM Watson Analytics є швидкою, надійною та точною. Веб-розміщення дозволяє отримати доступ до системи з будь якого комп’ютера, що має доступ до мережі Інтернет. В свою чергу технічні характеристики системи знаходяться на найвищому рівні, оскільки використовуються найновітніші технології веб-стеку .Net.

ПЕРЕЛІК ПОСИЛАНЬ

1. Automatic Parallelization with Intel Compilers [Електронний ресурс] – Режим доступу до ресурсу: http://software.intel.com/en-us/articles/automatic-parallelization-with-intel-compilers

2. Naono K., Teranishi K., Cavazos J., Suda R. Software automatic tuning from concepts to state-of-the-art results – New York: Springer, 2010. – 240 с.

3. Asanovic K. The Landscape of Parallel Computing Research: A View From Berkeley. Technical Report, University of California, Berkeley, 2006

4. Дорошенко А.Ю, Іваненко П.А., Новак О.С. Гібридна модель автотьюнінгу з використанням статистичного моделювання // журнал «проблеми програмування», 2016

5. Mannila, Heikki. Data mining: machine learning, statistics, and databases // Eighth International Conference on Scientific and Statistical Database Systems, 1996.

6. IBM Watson Analytics [Електронний ресурс] – Режим доступу до ресурсу: https://www.ibm.com/ru-ru/marketplace/watson-analytics

7. Analytics of data-mining and data-science methodology [Електронний ресурс] – Режим доступу до ресурсу: https://www.kdnuggets.com/polls/2014/analytics-data-mining-data-science-methodology.html

8. Mass M. REST API Design Rulebook / Mark Mass., 2011. – 116 p.

9. Watson Analytics Developer Center [Електронний ресурс] – Режим доступу до ресурсу: https://developer.ibm.com/api/view/id-160:title-Watson_Analytics

10. Richardson L. RESTful Web APIs: Services for a Changing World / Leonard Richardson., 2015. – 355 с.

11. Auth0 documentation [Електронний ресурс] – Режим доступу до ресурсу: https://auth0.com/docs/api-auth/why-use-access-tokens-to-secure-apis

12. RestSharp documentation [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/restsharp/RestSharp/wiki.

13. Newtonsoft documentation [Електронний ресурс] – Режим доступу до ресурсу: https://www.newtonsoft.com/json/help/html/Introduction.htm.

14. Ivanenko P.A, Doroshenko A.Y., Zhereb K.A. TuningGenie: Auto-Tuning Framework Based on Rewriting Rules // 10th International Conference, ICTERI 2014, Kherson, Ukraine, June 9-12, 2014, P. 139–158.

15. Иваненко П.А., Дорошенко А.Ю. Метод автоматической генерации автотюнеров для параллелных программ // Кибернетика и системный анализ. – 2014. – № 3. – С. 75–83.