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

Факультет інформатики та обчислювальної техніки

Кафедра технічної кібернетики

«На правах рукопису»
УДК 004.043

«До захисту допущено»
Завідувач кафедри
__________ І.Р. Пархомей
(підпис)
“___”_____________2018 р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 121 «Інженерія програмного забезпечення»

на тему: Система навчання та аналізу навантаження планувальника завдань

Виконав: студент другого курсу, групи ІТ-74мп
(шифр групи)

Ромащенко Павло Сергійович __________ __________
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник доцент, к.т.н., доцент Пасько В.П. ____ __________
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант ___________ ____________________________ __________
(назва розділу) (науковий ступінь, вчене звання, , прізвище, ініціали) (підпис)

Рецензент ____ __________
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _______________
(підпис)

Київ – 2018 року

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»**

Факультет інформатики та обчислювальної техніки

Кафедра технічної кібернетики

Рівень вищої освіти – другий (магістерський)

Спеціальність 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
__________ І.Р. Пархомей
(підпис)

«___»___________ __2018 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Ромащенко Павло Сергійович
(прізвище, ім'я, по батькові)

1. Тема дисертації «Система навчання та аналізу навантаження планувальника завдань»

Науковий керівник дисертації к.т.н., доцент Пасько В.П.,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «07» листопада 2018 р. № 4112-с

2. Термін подання студентом дисертації ____________________________

3. Об'єкт дослідження – система аналізу даних планувальника завдань, засоби навчання та розподілення ресурсів.

4. Предмет дослідження – методи, засоби та алгоритми аналізу даних, збір та навчання системи коректного розподілення ресурсів.

5. Перелік завдань, які потрібно розробити:

1. Аналіз існуючих алгоритмів балансування навантаження на сервер.
2. Огляд існуючих рішень роботи вирішення навантаження на сервер.
3. Розробка алгоритму балансування навантаження.
4. Розробка нейронної мережі для аналізу мережі по різним параметрам та навчання системи обробки задач.
5. Розробка загального методу відправки повідомлень про стан системи.
6. Реалізація даної системи та випробування в умовах реальної роботи.

6. Орієнтовний перелік ілюстративного матеріалу:

1. Схема побудови повідомлень.
2. UML–діаграма послідовності відправки повідомлень.
3. Схема навчання нейронної мережі.
4. UML–діаграма діяльності роботи нейронної мережі.
5. Структурна схема моделі класів.
6. UML–діаграма діяльності роботи в системі.

7. Орієнтовний перелік публікацій:

1. Ромащенко П.С Алгоритми балансування навантаження на сервер/ Павло Сергійович Ромащенко. // Актуальные научные исследования в современном мире. – 2018. – №9(41).
2. Ромащенко П.С, Хижняк М.Д Міжнародна конференція Modern scientific challenges and trends Warsaw – The use of self-organizational neural network for planning tasks in distributed systems.

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

9. Дата видачі завдання ______________________________

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Аналіз предметної області	01.09.2018 р.	
2	Постановка задачі	10.09.2018 р.	
3	Аналіз інформаційного забезпечення	15.09.2018 р.	
4	Аналіз алгоритмічного забезпечення	25.09.2018 р.	
5	Розробка алгоритмів	15.10.2018 р.	
6	Розробка процесу розгортання	20.10.2018 р.	
7	Розробка додатку	01.11.2018 р.	
8	Налагодження додатку	07.11.2018 р.	
9	Маркетинговий аналіз стартап-проекту	10.11.2018 р.	
11	Висновки	15.11.2018 р.	

Студент ____________ (підпис) Ромащенко П.С. (ініціали, прізвище)

Науковий керівник дисертації ____________ (підпис) Пасько В.П. (ініціали, прізвище)

АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка магістерської дисертації складається з 7 розділів, містить 36 рисунків, 19 таблиць, 3 додатки, 21 джерело.

Магістерська робота присвячена створення комплексної системи, яка аналізувала вхідні дані з різних джерел, опрацьовувала їх та могла коректно та вірно розподілити навантаження на сервери.

Цілі розробки — підвищення швидкості та продуктивності роботи серверів або кластеру, стабілізація роботи та безвідмовність за рахунок правильного розподілення навантаження на сервери.

Задача розробки — створення комплексної системи для збору даних, в мережі, якій вона працює, аналіз та опрацювання та в подальшому використовуючи алгоритм нейронної мережі — багатошаровий перцептрон, для того, щоб система самостійно навчалась та могла коректно розподілити ресурси на сервери таким чином, щоб кожен з них не простоював без роботи та не було ситуації перевантаження процесорних можливостей сервера.

В рамках магістерської дисертації проводиться дослідження різних видів алгоритмів балансування навантаження на сервер, для порівняння та вибору найкращого, а також порівняння алгоритмів нейронних мереж для аналізу та навчання. Дана система являє собою комплексне рішення та має в собі під системи різного характеру, а саме: система повідомлень роботи використовуючи сучасні системи обміну повідомленнями та графічного опрацьовування роботи та слідкування за системою.

Ключові слова: балансування, навчання, аналіз даних, алгоритми балансування навантаження, мікросервіси, розгортання, нейронна мережа, багатошаровий перцептрон.

ABSTRACT

Structure and scope of work. The explanatory note of the master's dissertation consists of 7 sections, containing 36 drawings, 19 tables, 3 annexes, 21 sources.

The master's work is devoted to the creation of a comprehensive system that analyzed input data from different sources, worked out them, could correctly, and correctly distribute the load on servers.

The development goals are to increase the speed and performance of the servers or cluster, stabilize the work and failures due to the correct allocation of the load to the servers.

The task of development is the creation of a comprehensive system for data collection, on the network it operates, analysis and processing, and in the future using the algorithm of the neural network — the multilayer perceptron, in order for the system to study independently and able to correctly allocate resources to servers in this way, so that each one does not idle without work and there was no overloading of the processor capabilities of the server.

As part of the master's thesis, various types of load balancing algorithms for the server are conducted, for comparison and selection of the best, as well as comparison of algorithms of neural networks for analysis and training. This system is a comprehensive solution and includes a system of different types: a system of message work using modern messaging systems and graphic workflow and monitoring system.

Keywords: balancing, training, data analysis, load balancing algorithms, microservices, deployment, neural network, multilayer perceptron.

Пояснювальна записка
до магістерської дисертації

на тему: «Система навчання та аналізу навантаження планувальника завдань»

Київ – 2018 року

ЗМІСТ

ПЕРЕЛІК ТЕРМІНІВ ТА СКОРОЧЕНЬ 9
ВСТУП 11
1 ЗАДАЧА СИСТЕМИ НАВЧАННЯ ТА АНАЛІЗУ НАВАНТАЖЕННЯ ПЛАНУВАЛЬНИКА ЗАВДАНЬ 13
1.1 Мета роботи 13
1.2 Задача системи 13
1.3 Цілі системи 15
1.4 Методологія дослідження 16
1.5 Характеристика об’єкту автоматизації 16
Висновок до розділу 17
2 ОГЛЯД ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ 18
2.1 Системи навчання планувальника завдань 18
2.2 Планувальник завдань для хмарних обчислень з нейронною мережею 18
2.3 Аналіз алгоритмів планування роботи в області хмарних обчислень 21
2.4 Планувальник процесів з використанням нейромережевих технологій 31
Висновок до розділу 37
3 ОПИС АРХІТЕКТУРНОЇ РЕАЛІЗАЦІЇ ТА ЗАСОБІВ РОЗРОБКИ 38
3.1 Архітектура програмного забезпечення 38
3.2 Мікросервісна та модульна архітектура додатку 39
3.3 Засоби реалізації 48
3.4 Засоби розгортання 53
3.5 Архітектура обміну повідомленнями 57
Висновок до розділу 62
4 ОПИС ВИКОРИСТАНИХ АЛГОРИТМІВ 63
4.1 Алгоритм Round Robin 63

4.2 Алгоритм Weighted Round Robin 65

4.3 Задача комівояжера 66

Висновки до розділу 67

5 ОПИС НЕЙРОННОЇ МЕРЕЖІ 69

5.1 Існуючі нейронні мережі 69

5.2 Багатошаровий перцептрон 75

5.3 Реалізація алгоритму багатошарового перцептрона 82

5.4 Тестування нейронної мережі 85

Висновок до розділу 87

6 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ 88

6.1 Реалізація роботи з ботами для сповіщення 88

6.2 Реалізація логування 91

Висновки до розділу 93

7 СТАРТАП ПРОЕКТ 94

7.1 Опис ідеї проекту 94

7.2 Технологічний аудит ідеї проекту 97

7.3 Аналіз ринкових можливостей запуску стартап-проекту 99

7.4 Розроблення ринкової стратегії проекту 105

7.5 Розроблення маркетингової програми стартап-проекту 106

Висновки до розділу 106

ВИСНОВКИ 108

ПЕРЕЛІК ПОСИЛАНЬ 110

ДОДАТКИ 112

ПЕРЕЛІК ТЕРМІНІВ ТА СКОРОЧЕНЬ

ОС	Операційна система
Фреймворк	Інфраструктура програмних рішень, що полегшує розробку складних систем.
VS 2017	Спеціалізоване середовище розробки програмного забезпечення різного рівня та складності
FIFO	Спосіб організації та маніпуляції за даними щодо часу та пріоритетів. Даний вираз описує принцип технічної обробки черги шляхом упорядкування процесу за принципом: "перший прийшов – перший зроблений"
Use-Case	(з англ. варіант використання) відображає варіанти взаємодії користувача із програмою.
LXC	(з англ. Linux Containers) система віртуалізації на рівні операційної системи для запуску декількох ізольованих примірників Linux на одному комп'ютері
API	Інтерфейс прикладного програмування, опис методів (набір класів, процедур, функцій, структур або констант), за допомогою яких одна комп'ютерна програма може взаємодіяти з іншою програмою.
SOLID	абревіатура складена з перших літер п'яти базових принципів об'єктно-орієнтованого програмування та дизайну
Guid	статистично унікальний 128-бітний ідентифікатор
TDD	(з англ. Test-driven development) технологія розробки програмного забезпечення, яка використовує короткі ітерації розробки, що починаються з попереднього написання тестів, які визначають необхідні покращення або нові функції. Кожна ітерація має на меті розробити код, який пройде ці тести
Бібліотека	Перелік просторів імен, класів та їх властивостей і

	функцій, що об’єднані в одну dll бібіотеку для поставки клієнтам.
Cross-platform	У комп'ютерних програмах крос-платформене програмне забезпечення (також багатоплатформове програмне забезпечення або незалежне від платформи програмне забезпечення) це комп'ютерне програмне забезпечення, яке реалізується на кількох обчислювальних платформах.

ВСТУП

Кожного дня люди користуються різними сервісами, додатками та сайтами, які дають можливість знаходити інформацію, обмінюватись нею. Всі користувачі хочу, щоб всі системи на мобільному пристрої або комп'ютері працювали швидко, коректно та якісно. На сьогоднішній день кількість користувачів сягає мільйони. Це люди різного віку та з різними пристроями, які мають різні операційні системи та всі вони працюють злагоджено та виконують свої функції. На сьогодні у світі зареєстровано дуже багато різної техніки та різні статистичні дані показують на відмітку у мільярди пристроїв.

Але не всі системи стабільно та коректно працюють. Навіть великі корпорації, котрі мають велику кількість розробників, менеджерів та адміністраторів не можуть зробити систему ідеальною та налагодженою таким чином, щоб працювати та вирішувати задачі автономно. Всі системи зазнають невдачі у вигляді перевантаження ресурсами системи, виділення малих ресурсів для обробки великих задачі, котрі потребують значних витрат.

Деякі компанії намагаються вирішити дану задачу та проблему шляхами додавання ресурсів до системи або зміною алгоритму обробки, а не в цілому. Задача магістерської дисертації зробити огляд таких проблем та поглянути на проблему з іншого боку та вирішити її кардинально унікальною та універсальною системою.

Актуальність теми дослідження. Поширення систем, сервісів. Створення великих компаній, котрі прагнуть стабільності в роботі інфраструктурної системи для того, щоб користувачі були повністю задоволені роботою. Прагнучі до швидкості відклику та затрати мінімального часу, щоб обробити дані та відати їх клієнту. Для стабільності роботи додатків, програмних комплексів та систем в цілому. Зважаючи на всі вище наведені фактори обґрунтовують актуальність створення даної системи та впровадження в компаніях.

Об’єктом дослідження є інтелектуальні системи, які можуть опрацьовувати великі об’єми даних, задачі на виконання планувальником задачі, робити аналіз та використовуючи набуті дані проводити навчання системи.

Предметом дослідження виступають алгоритми роботи та методи опрацювання задач в системі.

Постановка проблеми. Для досягнення мети виконання магістерської дисертації необхідно вирішити такі задачі:

1. Аналіз існуючих інтелектуальних систем для балансування навантаження та аналізу разом з навчанням системи;
2. розробка алгоритму балансування навантаження;
3. розробка нейронної мережі для аналізу мережі по різним параметрам та навчання системи обробки задач;
4. розробка загального методу відправки повідомлень про стан системи;
5. програмна реалізація даної системи та випробування в умовах реальної роботи.

Наукова новизна даної магістерської роботи дуже значна в загалом та націлена на створення нової системи, яка полегшить роботу планувальника завдань та роботу сервера в цілому. Покращить швидкість та інтелектуальність обробки задач.

Основними галузями застосування системи в подальшому розвитку повинні бути компанії будь якого об’єму та будь якій галузі, навчальні заклади, заводи, котрі прагнуть стабільності в роботі всіх програмних комплексів та сервісів без перебоїв та перевантажень.

Система повинна в цілому вирішити проблему збоїв роботи планувальників задач та серверів.

1 ЗАДАЧА СИСТЕМИ НАВЧАННЯ ТА АНАЛІЗУ НАВАНТАЖЕННЯ ПЛАНУВАЛЬНИКА ЗАВДАНЬ

1.1 Мета роботи

Мета магістерської роботи — створення програмної системи навчання та аналізу навантаження планувальника завдань використовуючи сучасні системи розробки, розгортування, алгоритми балансування навантаження та створення, розробку та навчання нейронної мережі для опрацювання даних та коректно розподілення навантаження на сервери.

Створення саме даної інтелектуальної системи допоможе вирішити задачу різних типів навантажувачів на сервер, які балансують навантаження на сервер, але не беруть до уваги велику кількість параметрів для оброки та аналізу. Створена система буде обробляти дані різного типу та характеру завдяки чому покращиться робота балансувальника та завдяки нейронній мережі, яка буде навчатися на основі даних, які надходять до системи, буде їх опрацьовувати та аналізувати та розмежовувати роботи на більш незавантажені сервери для правильної та швидкої роботи.

1.2 Задача системи

Одна з основних задач роботи — створення унікальної та інтелектуальної системи розподілення роботи на сервери.

Велика кількість сервері знаходяться в кластері та обробляють данні користувачів в будь якій компанії. Деякі компанії використовують хмарні обчислення, а інші мають власні кімнати, в яких зберігаються великі шафи з серверами обробки даних. Кількість користувачів в будь якій компанії росте с кожним роком, таким чином навантаження на дані сервери також росте. Основна задача лягає на сервери баз даних та багато компаній стикалися з падінням сервера або перевантаження в результаті чого потрібно було виключати та перезавантажувати один сервер, а коли це кластер то це дуже болісний процес, бо всі сервіси та додатки перестають працювати.

«Падіння» або перевантаження сервера (а це відбувається зазвичай несподівано, в самий неочікуваний момент часу та роботи) загрожує дуже серйозними наслідками. Для початку проблеми з продуктивності сервера в зв'язку зростанням кількості користувачів, а втім та навантажень можна вирішувати шляхом додавання потужності до сервера, або ж оптимізацією різних алгоритмів, які використовуються в системі, програмних кодів. Але інколи таких заходів виявляються недостатніми і тому потрібно з цією проблемою щось робити.

Компанії борються різними способами, інколи вдаються до кластеризації: кілька серверів об'єднуються в кластер серверів та навантаження між ними ділиться за допомогою комплексу спеціальних алгоритмів. Крім вирішення цих проблем кластеризація також допомагає забезпечити резервування серверів один на одного. Ефективність даного методу залежить від того, як розподіляється (балансується) навантаження між різними елементами кластера. Балансування навантаження може здійснюватися різними способами та методами, наприклад за допомогою апаратних або програмних інструментів.

Інколи такий термін, які реплікація дуже допомагає по вирішенні даної проблеми. Саме використання механізму синхронізації вмісту декількох копій об'єкта на декілька сервери або на кластери серверів дає своє переваги. Але даних підхід швидко перестає працювати, бо немає механізму розподілення запитів на дані сервери і тому даний підхід не має великого профіту від реалізації. Тому використовують різні балансувальники навантаження на сервери, які мають різні алгоритми, тим самим даний підхід дає вирішення проблеми та задачі розподілення навантаження.

Задача даної роботи — це реалізація інтелектуального балансувальника на сервер, який буде аналізувати велику кількість даних, опрацьовувати їх та на основі даних робити висновки та розподілення роботи на кластер серверів.

Система навчання та аналізу навантаження планувальника повинна реалізувати наступні поставлені задачі перед нею:

1. реалізація системи навчання планувальника завдань відповідно до обчислювальним можливостям інфраструктури;
2. реалізація аналізу навчання та перерозподіл навантаження відповідно до обчислювальним можливостям інфраструктури;
3. реалізація системи відображення всіх параметрів роботи планувальника завдань в інфраструктурі;
4. вдосконалення роботи планувальника завдань на різних операційних системах;
5. реалізація нейронної мережі для розподілення навантаження на сервер за допомогою навчання;
6. реалізація універсальної системи розгортування використовуючи сучасні інструменти для роботи на будь якій операційній системі;
7. реалізація системи сповіщення роботи планувальника відповідно до різних рівнів повідомлення (інформація, повідомлення, попередження, помилка, критична помилка) за допомогою використання сучасних додатків оброблення повідомлень;
8. реалізація системи відображення основної інформації та статусу роботи планувальника завдань на різних пристроях та операційних системах.

1.3 Цілі системи

Даний додаток вирішує поставлені перед собою цілі:

1. підвищення ефективності обчислювальних ресурсів шляхом використання нейронної мережі для планувальника завдань;
2. оптимізація роботи алгоритмів балансування навантаження планувальника завдань з використання нейронної мережі;
3. швидке та зручне сповіщення користувачів та адміністраторів при будь якій зміні стану системи або певних помилках.

1.4 Методологія дослідження

Для розробки даної системи, було оброблено велику кількість матеріалу та виокремлено основні позиції, які саме дали підґрунтя для методологічної основи дослідження:

1. аналогічні системи, які стали основою для розробки даної системи;
2. алгоритми будування нейронних мереж;
3. підходи розробки на кросплатформеній технології .net core;
4. методи та правила будування веб застосувань за допомогою кросплатформеній технології .net core;
5. методи побудови архітектури розгортування за допомогою сучасних інструментів;
6. методи використання мікросервісної архітектури;
7. методи використання патернів розробки якісного та тестованого коду;
8. підходи та шаблони розробки масштабованого коду.

1.5 Характеристика об’єкту автоматизації

В даній роботі автоматизується та покращується цілий процес розподілення роботи на сервер. Вона повинна надати розробникам та системним адміністраторам полегшити роботу та надати універсальний інструмент для роботи з розроблюваним даним програмним рішенням.

Визначено зацікавлені сторони в даній системі:

— розробники — зацікавленість полягає у використанні універсального, сучасного та робочого інструменту при розробці великих та масштабованих додатків та не звертати велику увагу на серверні можливості, які є та будуть використовуватись для роботи проекту, бо є програмне рішення для розподілення навантаження;
— системні адміністратори — зацікавленість полягає спрощення налаштування балансувальника та легкість у використанні в роботі;
— менеджери — швидкість при роботі в системі та викладення нових релізів в системі без втрачання продуктивності системи та не зависання в цілому.

Висновок до розділу

В даному розділі описано мету даної роботи та поставлена задача, яку повинен вирішити дана магістерська дисертація. Було описано пункти, які система повинна вирішити та реалізувати пед собою.

Також було виділено складову методологію дослідження, яка допомагала при створенні додатку, а це різні алгоритми, різні підходу по розробці тестованого коду, методи побудови архітектури розгортування за допомогою сучасних інструментів, прийоми та методи використання мікросервісної архітектури в великих системах та взаємодіє між сервісами.

В наступному розділі було зроблено детальний опис та характеристика вже існуючих програмних рішень системи навчання планувальника завдань. В кожній із систем є наявні алгоритми мереж, які було детально описані та розкладені для розуміння та внесення ясноті роботи інших систем в цілому.

2 ОГЛЯД ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ

2.1 Системи навчання планувальника завдань

В розділі було детально досліджено різні методи та технології, які використовуються для навчання планувальника завдань. Було досліджено різні реалізації хмарних обчислень з використання нейронних мереж для навчання системи.

2.2 Планувальник завдань для хмарних обчислень з нейронною мережею

"Job Scheduling Cloud Computing Using Neural Networks" — назва статті в академічному журналі "Scientific Research", яка написана авторами Mahmoud Maqableh, Huda Karajeh, Ra'ed Masa'deh — містить в собі інформації, щодо створення планувальника завдань для хмарних обчислень використовуючи технології нейронних мереж [1].

В даній статті йдеться про те, що хмарні обчислення спрямовані на максимальне використання переваг розподілених ресурсів та їх сукупність досягти більшої пропускної здатності для вирішення великомасштабних обчислювальних завдань. Також вони описують, що планувальник завдань є однією з найбільших проблем у хмарі, бо планувальнику потрібно правильно розділити ресурси на запити, які до нього надходять та виконати їх за мінімальний час. Основним завданням системи планування є пошук найкращих ресурсів, беручи до уваги деякі статистичні дані та динамічні обмеження параметрів ресурсів, які має система [1].

Для вирішення цих та інших задач, в даній статті автори впроваджують в хмарні обчислення алгоритм штучних нейронних мереж. Автори в своїй роботі використовували різні підходи та алгоритми для пошуку оптимального розподілу ресурсів. Тому в даній статті автори пропонують реалізацію штучних нейронних мереж, щоб оптимізувати результати планування роботи в хмарі [1].

В даній статті йдеться про різні види хмар, які використовується та переваги їх у використанні. Крім того, пояснюється, що система хмарних

обчислень складається з двох основних частин, які підключені один до одного через інтернет: backend, frontend частини. Хмарні обчислення спрямовані на те, щоб максимально використовувати розподілені ресурси, а їх поєднувати, щоб вони могли вирішити великомасштабні обчислювальні завдання [1].

Також в статі, описуються генетичні алгоритми — алгоритми пошуку, що імітують процеси природного відбору та природної генетики, що використовують для розрахунку оціночних рішень складних проблем, на рис. 2.1 [1].

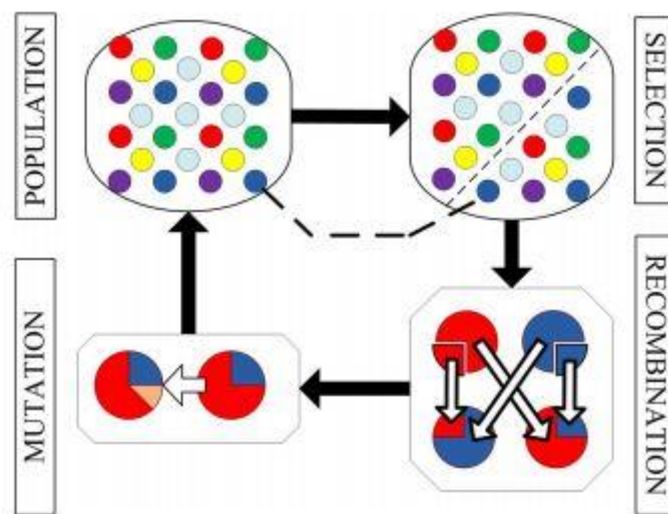


Рисунок 2.1 – Простий генетичний алгоритм

Генетичний алгоритм популяції конкурує один з одним для перетворення кандидата на вирішення проблеми, який буде обраний на основі функції фітнесу. Основними етапами генетичних алгоритмів є початкова популяція, функціональність, вибір, мутація. Початкове населення складається з усіх осіб, які використовуються в генетичному алгоритмі, щоб з'ясувати оптимальне рішення. Кожне рішення в населенні називається індивідуальним.. Особи вибираються з початкового населення і деякі операції застосовуються для тих, хто генерує наступне покоління. Операція вибору поєднання хромосом, ґрунтується на деяких специфічних критеріях. Функція фітнесу використовується для вимірювання якості обраних осіб з населення відповідно, а конкретна ціль оптимізації. Функція фітнесу може бути різною, в деяких

випадках може бути функція фітнесу ґрунтуючись на максимізації деяких факторів, в інших випадках він може базуватися на мінімізації інших чинників [1].

Генетичні алгоритми були використані в деяких аспектах дизайну нейронних мереж, оскільки вони можуть знайти оптимальне рішення. Можна розглядати як спосіб планування роботи, який базується на біологічній концепції генерації населення.

Штучна нейронна мережа — це парадигма обробки інформації, яка імітує нейронний мозок людського мозку, зображено на рис. 2.2. Він розроблений, щоб імітувати спосіб, що людський мозок виконує конкретне завдання або функцію, певну задачу [1].

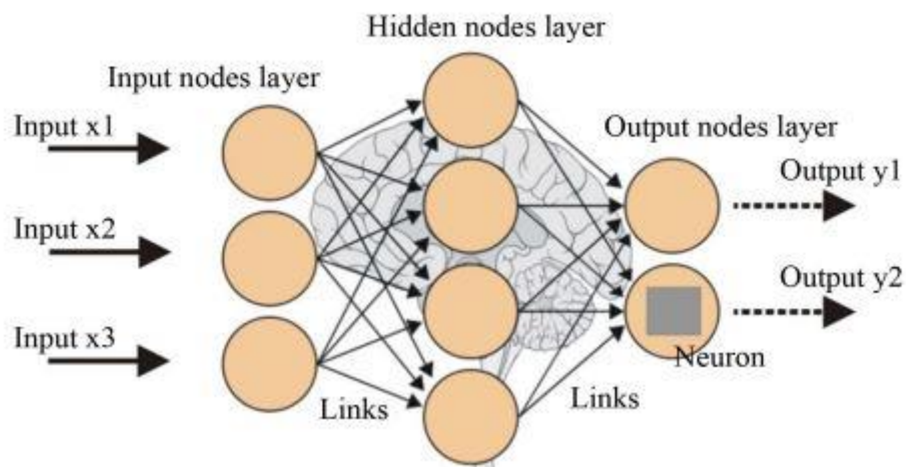


Рисунок 2.2 – Базова структура штучної нейронної мережі

Адаптивний характер цієї мережі вважається однією з найважливіших рис, де "навчання за прикладом" використовується для вирішення завдань [19]. Таким чином, ця модель використовується для вирішення складних або неоднозначних системних завдань, класифікація та розпізнавання моделі. Ці проблеми будуть дуже важко видобути з використанням багатьох інших комп'ютерні техніки. Нейрона мережа може дати дуже хороший результат, коли він використовується в складних системах, які не є повністю зрозумілими. Модель нейронної мережі має три основні проблеми:

— топологія мережі;

— функція передачі;

— алгоритм навчання.

Нейронна мережа складається з процесорних одиниць, зважених з'єднань та правила навчання. Нейронні мережі складаються з трьох або більше шарів, і кожен шар має підрозділ обробки, який називається нейронами. Він має вхідний шар, вихідний шар і приховані шари. Мережа зв'язує вхідні шари з вихідними шарами, що використовують приховані шари з функцією нелінійної трансформації та зважених з'єднань. Штучні нейронні мережі можуть мати різну кількість шарів і різну кількість вузлів. Характер проблеми і ступінь складності контролюють кількість прихованих шарів і їхніх нейронів [1].

Навчання мережі здійснюються за допомогою різних курсів навчання, параметрів та різних методів розповсюдження, таких як feedforward та зворотне поширення. Вона вивчається шляхом зміни зав’язків між вхідними та вихідними шарами. Точність виведення в мережі залежить від параметрів, які використовуються для навчання системи. Продуктивність мереж залежить від кількості шарів, кількості вузлів та алгоритмів навчання. Кращі результати можна досягти, використовуючи архітектуру нейронної мережі з належним вибором вхідної змінної та навчального набору [1].

2.3 Аналіз алгоритмів планування роботи в області хмарних обчислень

В іншій статі "Analysis of Job Scheduling Algorithms in Cloud Computing" – – написана Rajveer Kaur, Supriya Kinger та видана в журналі "International Journal of Computer Trends and Technology (IJCTT)" йдеться також про використання алгоритму нейронної мережі у планувальнику задач [2].

В ній йде опис сучасності хмарних обчислення та вирішення проблеми планування задачі. Автори роблять наголос, що хмарні обчислення — це одна з найближчих новітніх комп'ютерних парадигм, де надаються послуги та послуги з передачі даних по інтернету. Хмарні обчислення - це сервіс, націлений надавати

якісні та інформаційні послуги за допомогою плати за використання моделі, в якій гарантії пропонуються постачальниками хмарних сервісів. Хмарне обчислення — це платформа, націлена надавати спільним даним своїм клієнтам одночасно. Cloud Computing пропонує різні моделі обслуговування. Це може бути програмне забезпечення як модель обслуговування, пропонуючи програмне забезпечення на одному платформа або може бути платформа як модель обслуговування, яка пропонує платформу, звідки знаходиться програмне забезпечення. Або це може бути інфраструктура як служба, яка забезпечує безпеку та резервну копію ваших сервісів [2].

Планування автори вважають — це процес розподілу завдань наявні ресурси на основі якості завдань і потрібно. Головна мета планування – це збільшення використання ресурсів без, що впливають на послуги, надані хмарою. Є два типи планування, тобто ресурс планування та планування роботи. Нижче перераховані деякі потреби в плануванні в хмарних обчисленнях:

— розподіл ресурсів — розподіл ресурсів найвигіднішим чином, щоб економити та використовувати все те, що потрібно для програмного забезпечення;
— QOS — ресурси та робочі місця заплановані в таким чином, щоб забезпечити якість послуг;
— використання ресурсів — це ступінь до які ресурси системи є використаний. Якісний алгоритм планування забезпечує максимальне використання ресурсів;
— споживання електроенергії — це ступінь до які ресурси системи є споживаний, алгоритм планування економить споживання енергії [1].

Також автори дають вдалу характеристику для планувальника задач:

— планування роботи є глобально централізованим. Оскільки обчислення на хмарних ресурсах є обчислювальною моделлю, яка поставляє централізований ресурс дзеркальною службою для декількох розподілених програм, і це розгортання дзеркального відображення може зробити

простіше здійснення неоднорідних процедур, пов'язаних з виконанням взаємодії, що важко було вирішити. Тому віртуалізовані технології та служби віддзеркалення роблять завдання планування хмарних обчислень досягти глобального централізованого планування. Вирішується одна з задач розподілення ресурсів на різні сервери, саме розгортування дзеркального відображення сервера на інші сервери. Краще за все — це робити з використанням сучасних технологій, а саме контейнерів, які допомагають вирішити цю задачу найлегшим способом. В нашій розробленій системі, як раз використовується дана технологія;

- кожен вузол в хмарі незалежний. У хмарних обчисленнях внутрішнє планування кожного хмарного вузла є автономним, а планувальники в хмарі не будуть перешкоджати політиці планування цих вузлів — це дає можливість вдало контролювати роботу сервісу, бо по різним причинам деякі вузли можуть на певний час відключатись або бути не доступні через проблеми в мережі або інше. Таким чином навантаження переходить на інший вузол і система продовжую працювати належним чином, що користувачі не помічають даних проблем та помилок;
- масштабованість — масштаб постачання ресурсів від постачальника хмар може бути обмежений на ранніх стадіях. Завдяки додаванню різноманітних обчислювальних ресурсів розмір абстрактних віртуальних ресурсів може стати великим, а попит на додаток продовжує зростати. У хмарі планування завдань повинно відповідати функціям масштабованості, так що пропускна здатність планування завдання в хмарі може бути не надто низькою;
- планувальник задач може бути динамічно само адаптивним — розширення та скорочення додатків у хмарі можуть знадобитися, залежно від вимоги. Віртуальні обчислювальні ресурси в хмарній системі також можуть одночасно розширюватися або зменшуватися. Ресурси постійно змінюються, деякі ресурси можуть не працювати, нові ресурси можуть приєднатися до хмар або перезапускатися. Таким чином планувальник

задач в хмарі корегується самостійно та починає працювати залежно від виділених ресурсів та можливостей в системі;

— набір планування виконання задач — планування завдань складається з двох частин: один з них використовується як уніфікований розподіл ресурсів, і в першу чергу відповідає за планування програм та хмарного API; інший — для уніфікованого планування ресурсів порту в хмарі, наприклад, планування завдання MapReduce. Проте, кожне планування складається з двох двосторонніх процесів, які планувальник орендує ресурс за допомогою хмари та планувальника відкликання запитаних ресурсів після використання. Перший процес — це стратегія планування, а друга — стратегія зворотного виклику. Комбінація стратегії ресурсів планування та зворотного виклику — це набір планування завдання [2].

В статі йде приклад алгоритмів, які були використані авторами.

Перший вперше обслуговує — алгоритм планування — він також відомий як First in First out, але не потрібно путати з алгоритмом оброки черги FIFO, бо трохи різні області використання алгоритмів та характеристики, але основна задача одна. Найкоротша робота є вигідною через її простоту і тому, що вона мінімізує середню кількість часу, кожен процес повинен зачекати до завершення його виконання. Це один із найпростіших алгоритмів планування, який ми маємо, виділяємо процесор у тому порядку, в якому приходить процес. Передбачається, що готова черга керується спочатку першим, що означає, що перша робота буде оброблена спочатку без інших уподобань [2].

Алгоритм FCFS:

— ініціалізація задачі та всіх необхідних ресурсів для виконання задачі разом з налаштуванням;
— перша задача призначена до черги — черга починає свою роботу по виконанні задач, разом з цим до черги додаються задачі n;
— додається наступна задача на останню позицію в черзі.

FCFS (Example)

Process	Duration	Oder	Arrival Time
P1	24	1	0
P2	3	2	0
P3	4	3	0

Gantt Chart :

P1 waiting time : 0
P2 waiting time : 24
P3 waiting time : 27

The Average waiting time :

(0+24+27)/3 = 17

Рисунок 2.3 – Приклад коду алгоритму SJF

На рис. 2.3 показано роботу планувальника задач з використанням алгоритму FCFS. Бачимо, що перша задача ставиться у чергу та починає свою роботу. P1-P3 — це процеси в системі. За даним алгоритмом, після виконання першої задачі, буде додана наступна. В цілому кожен процес буде чекати час виконання попередніх задач, якщо це не перша задача в черзі. Середній час очікування кожним процесом – 17 секунд.

Найкоротший алгоритм першого планування — найкоротший робочий час (SJF), також відомий як Shortest Job Next (SJN) або Shortest Process Next (SPN) – – це метод планування, який обирає завдання з найменшим виконанням часу. Задачі розташовуються в черзі з найменшим робочим часом, розміщеним на першому місці, і задача з найдовшим терміном виконання виконується останньою і має найнижчий пріоритет. Цей алгоритм планування має справу з різним підходом у цьому алгоритмі. Центральний процесор виділяється на процес, що має найменший час розриву. Даний алгоритм має свої переваги та недоліки. Одним з основним недоліком є планування пріоритетів для задач та їх

виконання. Наприклад в системі потрібно виконати дуже важливу та ресурс затратну задачу з використання великої кількості часу, то дана задача по даному алгоритму буде поставлена в кінець черги на виконання [2].

Алгоритм SJF:

```
for i = 0 to i < main queue-size
    o if task i+1 length < task i length then
        ▪ add task i+1 in front of task
          i in the queue
    o end if
    o if main queue-size = 0 then
        ▪ task i last in the main
          queue
    o end if
end for
```

Рисунок 2.4 – Приклад коду алгоритму SJF

З даного алгоритму бачимо цикл по всім задача в черзі, він буде працювати доки в черзі є не виконані задачі. Якщо задача, яка надходить до черги менше по часу виконання, аніж що зараз є то саме вона буде додана до черги перед тою задачею, що є в черзі. Таким чином до черги додаються задачі відповідно до їх часу виконання.

Алгоритм планування Round-Robin — це один з найстаріших, найпростіших, найрозумніших і найбільш широко використовуваних алгоритмів планування, розроблених спеціально для систем розділення задач. Визначається невелика одиниця часу, що називається скиданнями квантом. Всі запущені процеси зберігаються в круговій черзі. Планувальник процесора обходить цю чергу, розподіляючи процесор на кожний процес за часовий інтервал одного кванту. Нові процеси додаються до хвоста черги. Планувальник процесора вибирає перший процес із черги, встановлює таймер для переривання після одного кванту і відправляє процес. Якщо процес все ще працює в кінці кванту, процесор перемагається, і процес додається до хвоста з черги Якщо процес

завершується до кінця кванта, процес сам відпускає процесор добровільно [2].

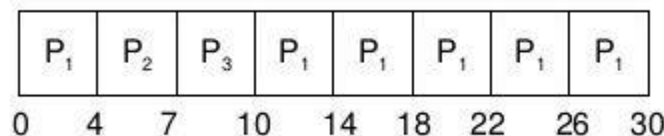
Алгоритм Round Robin:

— черга FIFO очищена та готова приймати процеси для обробки задач;
— нові процеси додаються до хвоста черги;
— планувальник вибирає перший процес з готової черги, встановлює таймер для переривання після 1 часового інтервалу та відправляє процес;
— планувальник починає опрацьовувати процес протягом одного кванту часу та закінчує даний процес, після цього планувальник переходить до наступного процесу в готовій черзі;
— в іншому випадку, якщо для виконання процесу потрібно більше ніж один часовий період, тоді таймер вимкнеться і призведе до переривання процесу. Буде виконано контекстний перемикач, і процес буде поставлений у кінці готової черги. Планувальник процесора потім вибере наступний процес у готовій черзі [2].

Round Robin

Process	Burst Time
P1	24
P2	3
P3	3

- Quantum time = 4 milliseconds
- The Gantt chart is:



- Average waiting time = {[0+(10-4)]+4+7}/3 = 5.6

Рисунок 2.5 – Приклад роботи алгоритму Round Robin

З рис. 2.5 бачимо, в даному прикладу за квантовий час було взято 4 мілісекунди та три процеси, які потрібно виконати, кожний процес має певний свій час на виконання, з рисунку бачимо — це burst time. Кожний процес виконується свій певний час, перший процес виконується найдовше — 24 мілісекунди. Тому робота по даному алгоритму буде така:

— взяти перший процес та виконати його за квант часу, так як він повністю не виконав свій процес він буде поставлений в кінець черги;
— черга бере наступний другий процес та виконує його за квант часу, так як процес виконується за 3 мілісекунди, то за квант часу він повністю буде виконано, він не буде доданий до кінця черги бо повністю виконаний;
— далі черга бере наступний процес та виконує його повністю, бо процес виконується також за вибраний період часу;
— далі в черзі залишаться виконати перший процес. Черга буде виконувати його кожний період часу, до поки не буде повністю виконаний;
— черга пуста, тобто всі процеси були успішно виконані.

Алгоритм планування за пріоритетом — даний алгоритм планування є переважним, в якому всі речі засновані на пріоритеті в цьому алгоритмі планування, кожен процес в системі базується на пріоритеті, тоді як робота найвищого пріоритету може виконуватися спочатку, тоді як завдання з низьким пріоритетом можна очікувати, найбільша проблема в цьому алгоритмі — це ситуація коли для задач, які будуть надходити до планувальника будуть ставитись однакові високі пріоритети, то задача, яка має нижчий пріоритет майже ніколи не виконається або виконається через довгий час [2].

З рис. 2.6 бачимо код для даного алгоритму. Даний алгоритм дуже простий та базується на пріоритеті. В циклі проходиться по всіх задачах, які знаходяться в черзі на виконання та береться пріоритет задачі, якщо наступна задача має більше пріоритет, аніж дана задача, то саме наступна задача додається до черги та буде опрацьована планувальником задач.

Алгоритм PSA:

```
for i = 0 to i <main queue-size
    o  if priority (task i+1 ) > priority (task i
       ) then
           ▪  add task i+1 in front of task
              i in the queue
    o  end if
end for
```

Рисунок 2.6 – Приклад коду алгоритму PSA

Та останній алгоритм, який описується в статті авторів це — генетичний алгоритм (GA) — це метод вирішення проблем, який використовує генетику як модель рішення задач. Це пошукова техніка для пошуку оптимізованого рішення. GA обробляє населення можливим рішенням. Кожне рішення представлено через хромосому. Генетичний алгоритм — це метод планування, в якому завдання призначаються ресурсами відповідно до індивідуальних рішень (які називаються графіками в контексті планування), що говорить про те, який ресурс слід призначити для якого завдання. Генетичний алгоритм базується на біологічній концепції генерації населення. У генетичному алгоритмі початкове породження формується випадковим чином. Генетичний алгоритм — це метод випадкового пошуку [2].

За рис. 2.7 можна виділити основні етапи роботи генетичного алгоритму:

— ініціалізація: генерування початкової випадкової кількості населення;
— фітнес функція;
— цикл роботи – поки критерій припинення дії не буде справджуватись;
— відбір популяції;
— кросовер – функція кросовера, генерує нове потомство, змінюючи його на кращий стан;
— мутація – призначений для зменшення часу процесора, які чекають даних від інших процесорів;
— знову робота фітнес функції;

— якщо дані по критерію справджуються то повернути дані.

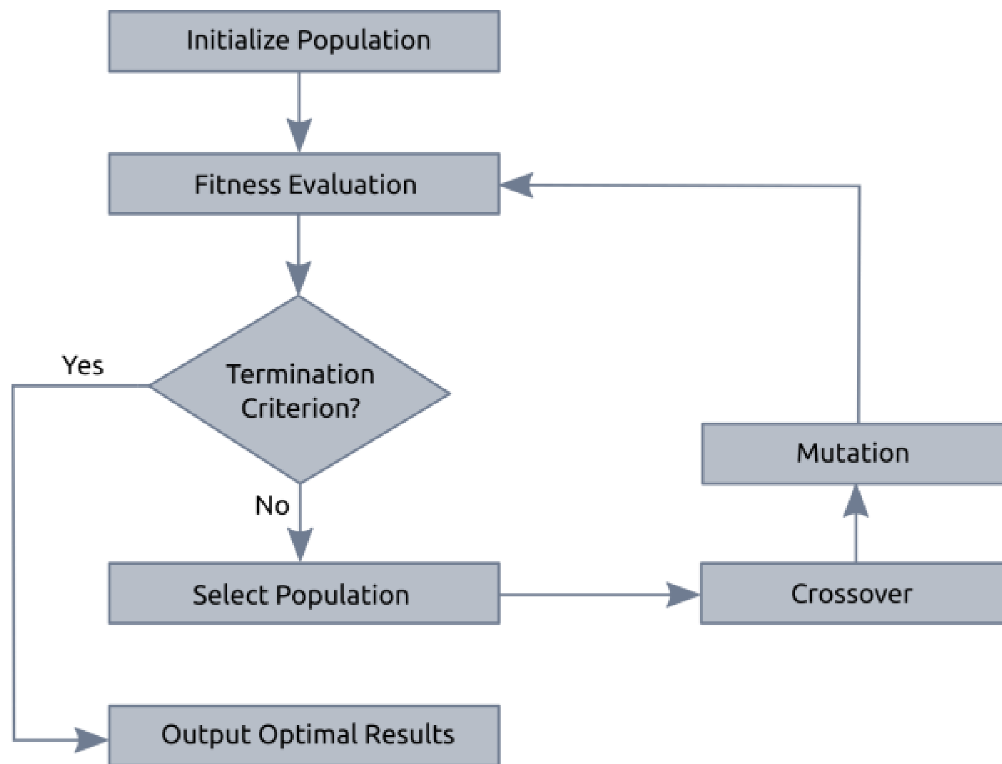


Рисунок 2.7 – Блок-схема алгоритму роботи GA

Після опису кожного алгоритму в статті робиться порівняльний аналіз.

Таблиця 2.1 – Порівняння різних алгоритмів

Алгоритм	Складність	Розподіл	Час очікування	Ти системи
FCFS	Найпростіший алгоритм планування	ЦП виділяється в тому порядку, в якому відбуваються процеси	Більше	Підходить для системи пакетного обміну даними
SJF	Важкий для розуміння та в розробці	Процесор розподіляється на процес, який має найменший час виконання	Менше, ніж алгоритм FCFS	Підходить для системи пакетного обміну даними

Таблиця 2.1 – Порівняння різних алгоритмів (закінчення)

Алгоритм за пріоритетами	Важкий для розуміння	Виходячи з пріоритету, найперша задача може працювати першою	Меньше, ніж алгоритм FCFS	Підходить як для системи пакетного, так і для часового обміну
Round Robin	Продуктивність сильно залежить від розміру квантового часу	Перехід на наступну задачу відбувається після фіксованого інтервалу часу	Більше за усі алгоритми	Підходить як для системи часового обміну
Генетичний алгоритм	Складність залежить від задачі, яку слід зробити	Це жадібний алгоритм і найкраща робота для вибору центрального процесора	Час очікування менше	Краще за все, коли простір пошуку великий

2.4 Планувальник процесів з використанням нейромережевих технологій

В даній науковій роботі університету Неймегена — «Radboud Universiteit Nijmegen» описується імплементація, розробка та тестування системи с використанням нейронної мережі. Автор даної роботи — Peter Bex.

В даній статі йдеться про різні алгоритми для використання планувальника задача. Зробимо опис з роботи деяких ключових алгоритмів, які ще не описувались.

Дерево прийняття рішень — це спосіб побудови дерева рішень (або дерева класифікації, у нашому випадку) з повного набору прикладів, які виражаються у дискретних значеннях. Алгоритм рекурсивно шукає предикат, який би поділив набір як можна рівніше в розглянутій галузі. Ці предикати не повинні бути двійковими. Цей процес створює дрібне, збалансоване дерево з предикатами на

кожному вузлі і остаточне рішення (або клас) на кожному листі. Дерево рішень можна використовувати для класифікації нових проблеми [3].

Алгоритм дуже простий і легко реалізований. Найбільший недолік для використання його як функції пріоритетного призначення є те, що вона вимагає дискретної цінності для роботи. Це можна зробити лише, дивлячись на вхід і, наприклад, Ще один можливий недолік є той факт, що гранулярність дискретних значень має бути визначена заздалегідь. Якщо в цільовому сценарії використання змінної коливається навколо невеликого діапазону значень, інші можливі значення, які змінна може досягти менш важливі, ніж невеликі відмінності в цьому діапазоні. Це має бути відображено таким чином, значення перетворюються в дискретні діапазони. Проблему можна виключити, попросивши користувача, який будує дерево, надати діапазони, але це також ускладнює використання системи [3].

Область застосування дерева рішень в даний час широка, але все завдання, які вирішуються цим апаратом можуть бути об'єднані в наступні три класи:

— опис даних: дерева рішень дозволяють зберігати інформацію про дані в компактній формі, замість них ми можемо зберігати дерево рішень, яке містить точний опис об'єктів;
— класифікація: дерева рішень відмінно справляються з завданнями класифікації, тобто віднесення об'єктів до одного з заздалегідь відомих класів. Цільова змінна повинна мати дискретні значення;
— регресія: якщо цільова змінна має безперервні значення, дерева рішень дозволяють встановити залежність цільової змінної від незалежних (вхідних) змінних. Наприклад, до цього класу належать задачі чисельного прогнозування (передбачення значень цільової змінної) [3].

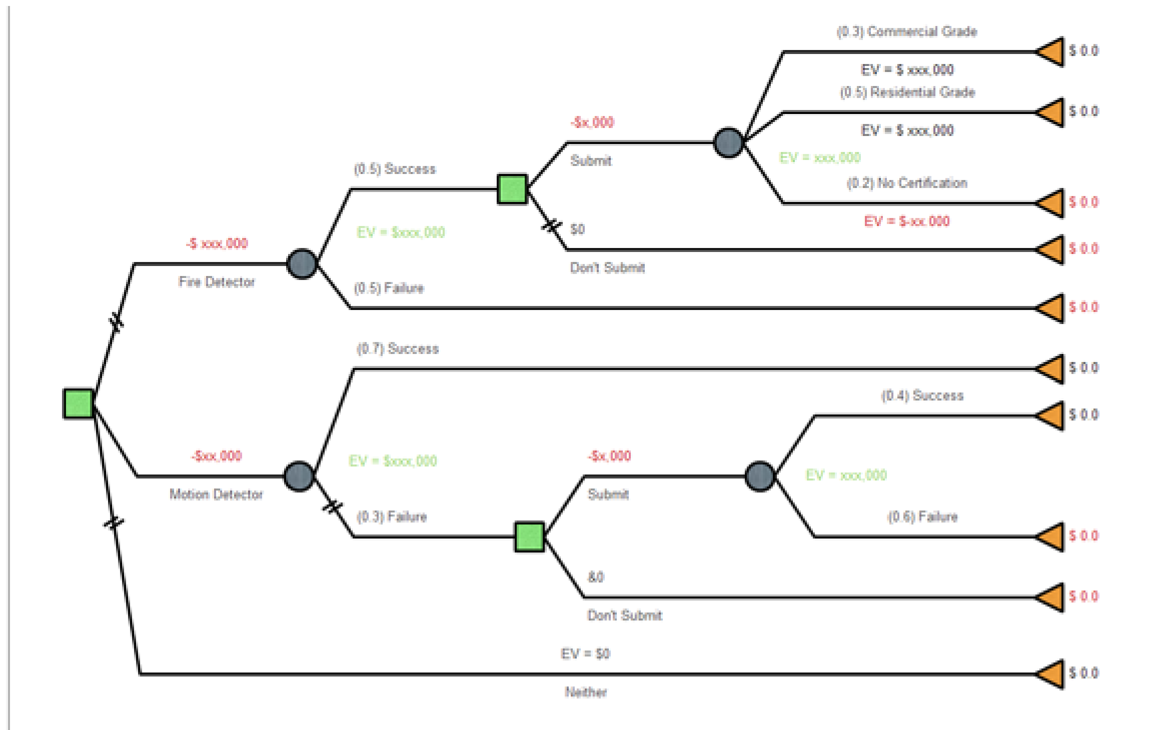


Рисунок 2.8 – Приклад алгоритму дерева рішень

Далі йдеться про систему алгоритмів основаних на певних правилах. Ця техніка в основному являє собою простий набір правил, якщо, то, далі. Це має більшість недоліків, як вивчення дерева рішень, і це також означає, що правила повинні бути ручними. Це зробить всю систему досить статичною і буде складно створювати нові набори правил. Це рішення різко знижує кількість людей, які можуть створювати планувальники задач, які добре працюють. Лише програмісти можуть зробити це, взагалі кажучи [3].

В наступному розділі дослідник виділяє проблему виділення процесу певний проміжок часу та проставлення коректної пріоритетності по задачам.

Зазвичай в мережах, що мають пам'ять, ця пам'ять використовується для зберігання значень функції на входах за попередніми етапами часу. У випадку коли входи на будь-якому етапі часу не мають нічого спільного з попереднім. Кожна мережа циклу вибирається новий процес, а його функції затискаються на вхідні вузли. Загалом, процеси не пов'язані. Таким чином, в один момент часу ми маємо характеристики одного процесу, закріпленого на вході, але пам'ять буде містити значення, пов'язане з входами попередніх, незв'язаних процесів. Це

перешкоджатиме навчанню, а не полегшить його. Вхідний сигнал у мережевому циклі t повністю не пов'язаний з входом при циклі t-1. Якщо є k процесів, то вхід на циклі t пов'язаний з тим, що відбувається в циклі t-k, але лише тоді, коли не було створено або знищено процеси проміжок часу. Це надто складно, занадто швидко. Глобальна історія, яка відображає взаємозв'язок між процесами, також важлива для планування, але ці проблеми вивантажуються на існуючого планувальника. Це означає, що планувальник нейронної мережі може зосередити увагу на основних питаннях розрахунку пріоритету для кожного процесу на основі їх ізольованої поведінки [3].

Рішення цієї проблеми це створення історії введення для кожного процесу шляхом перегляду вузлів пам'яті. Це означає, що коли функції процесу подаються в мережу, затискаючи їх на блоки введення, пам'ять та процес також буде затиснута до блоків пам'яті. Ця логіка фіксується псевдокоду в алгоритмі на рис. 2.9.

```
for all proc in proclist do
  gather_features(proc) {Reset counters, perform preprocessing}
  for i = 0 to nfeatures do {Clamp input and memory}
    network.input[i] ⇐ proc.features[i]
    network.memory[i] ⇐ proc.memory[i]
  end for
  run_network(network, proc)
  proc.nice ⇐ output_to_nice(network.output[0])
  for i = 0 to nfeatures do {Remember new memory values}
    proc.memory[i] ⇐ network.memory[i]
  end for
end for
```

Рисунок 2.9 – Приклад алгоритму для підрахунку пам’яті

Після запуску системи, вузли пам’яті зберігаються у процесі блоку керування для наступного циклу. Таким чином можливо виділити нейрону мережу для наступного процесу.

Для того, щоб виміряти накладні витрати планувальника нейронної мережі, було проведено декілька тестових тестів за допомогою програми benchmarking hackbench, яка була спеціально розроблена для тестування алгоритмів планування. Ця програма тестування створює ряд груп процесів, які спілкуються через сокет. Вона створює однакову кількість відправників і одержувачів у кожній групі. Всі відправники в групі відправляють сотню байтів всім одержувачам у своїй групі, що повторюються сто разів. Розмір групи — 40 процесів (тобто 20 одержувачів та 20 відправників) [3].

Накладні витрати виконуються шляхом вимірювання загального часу виконання або часу настінного годинника сеансу збірки. Вираховуючи час закінчення сеансу з часу початку, отримується результат. Ці результати можна порівняти між різними ядрами, щоб побачити, скільки виходить накладних витрат за розкладом, вираховуючи загальне час ядра тієї ж кількості груп з іншого загального часу роботи цих груп. Різниця — додатковий час, витрачений всередині планувальника, враховуючи, що все інше рівне. Ці тести проводились в так званому одно користувацькому режимі, який є режимом роботи, в якому не запускаються жодні сервіси, за винятком тих, які є необхідними для забезпечення роботи системи. Це було зроблено для мінімізації перешкод від інших процесів, які інакше могли б працювати (наприклад, cron, який може виконувати щоденний сценарій, який виконує пошук досить важкого диску). Кожен тест проводився 5 разів, а результати усереднювали для зменшення подальших аномалій. Після кожної партії з 5, система була перезавантажено, щоб попередні тести не впливали на пізні випробування. Результати відображаються на рис. 2.11 [3].

Система була протестована з використанням сокетів. На рис. 2.12 зображено, як сокет читає та записує для процесів firefox, mplayer та ftp в системі.

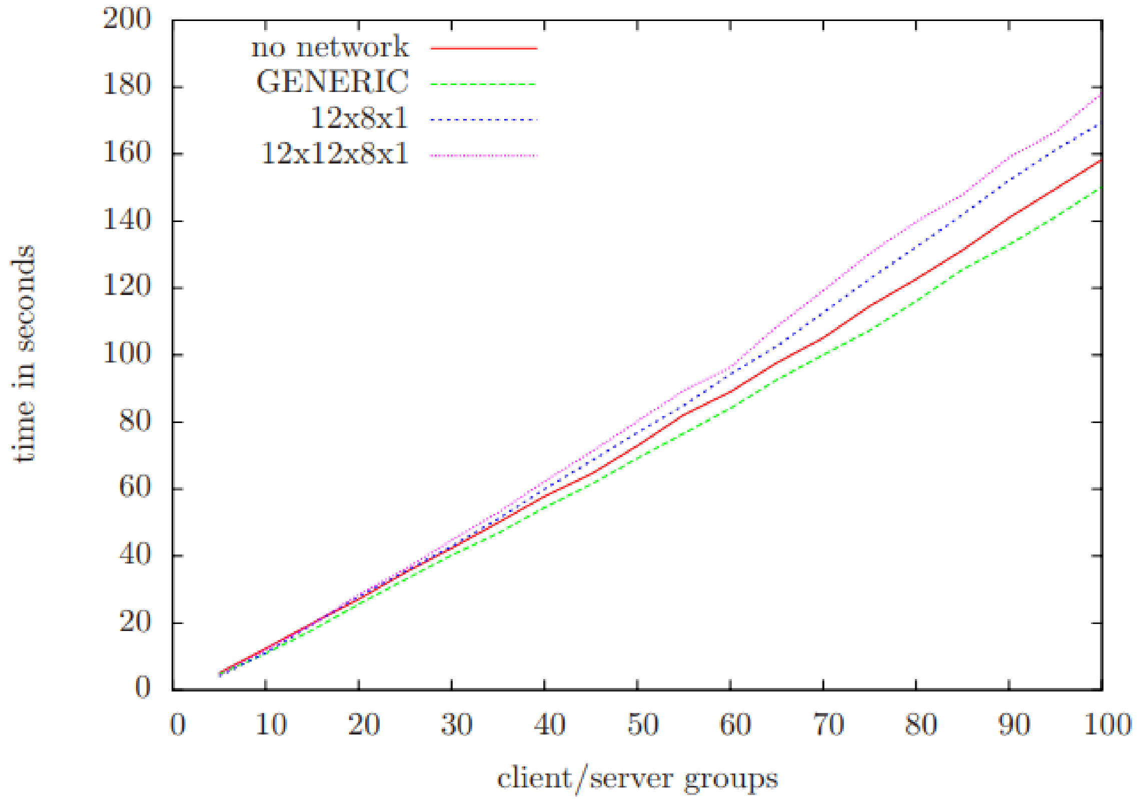


Рисунок 2.11 – Графік тестування з різними ядрами та мережами

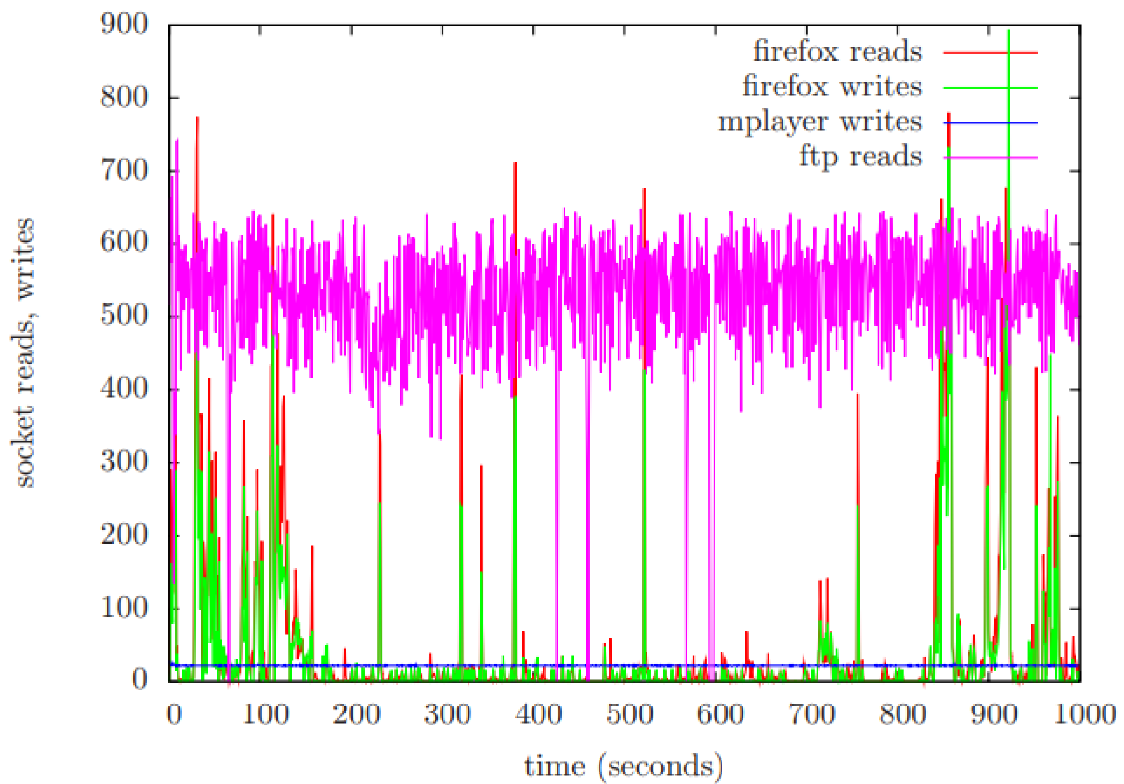


Рисунок 2.12 – Графік тестування з використанням сокетів

Висновок до розділу

В цьому розділі було зроблено опис та характеристику різних наукових докладів та дописів в журналах, можна зробити висновки, що еталонної системи в даній тематиці немає. Кожен автор використовує певний алгоритм та намагається експериментувати та навчати нейронну мережу спираючись на певні дані. Кожна система май свої успішні навчання нейронної мережі – таким чином автори робили висновки та намагались вдосконалити навчання змінивши алгоритм або його переробити на більш оптимальний. Але в результаті кожний розробник мережі описував вдалі спроби та приріст роботи планувальника завдань використовуючи обрані алгоритми.

В наступному розділі проведемо опис вибраної архітектури. Зробимо аналіз архітектурних рішень та виберемо підходящу до нашої системи. Також опишемо інструменти для розробки системи, розгортування та встановлення на різних серверів.

3 ОПИС АРХІТЕКТУРНОЇ РЕАЛІЗАЦІЇ ТА ЗАСОБІВ РОЗРОБКИ

3.1 Архітектура програмного забезпечення

Розробка будь якої системи починається з проектування в цілому. Процес визначення архітектури, контейнерів, інтерфейсів, модулів, процесу розгортування та встановлення на сервері та інші характеристики системи дає можливість виокремити певні задачі, які потрібно реалізувати перед початком розробки.

Після процесу аналізу та проектування програмного забезпечення було виділено основні моменти програмного додатку, для реалізації функціоналу та виконання всіх потреб, які були поставлені перед системою, такі як-от:

1. розробка програмного засобу використовуючи підхід, як мікросервіси;
2. архітектурна реалізація стабільного та гнучкого програмного забезпечення для подальшого розвитку додатку та додавання нового функціоналу;
3. використання сучасних технологій для розгортування програмного забезпечення на різних серверах;
4. використання технологій аналізу та навчання за допомогою нейронної мережі.

Аналіз всіх вимог показав, що найбільш краще та повно задовольняє цим критеріям мікросервісний принцип. Для побудови архітектури системи було взято та використано SOLID принципи, шаблони проектування та об’єктно-орієнтоване програмування. За допомогою цих принципів та правил розробки додатку можна легко вирішувати поставлені задачі перед замовником і тим самим досягти стабільної роботи програмного забезпечення.

Тому архітектура додатку було виконана з використанням модульної та мікросервісної архітектури з різними принципами програмування, які в результаті створюють стабільний, тестований веб-додаток.

3.2 Мікросервісна та модульна архітектура додатку

Мікросервісна архітектура додатку були вибрані для покращення розробки та впровадження нової функціональності в додаток тим самим не перешкоджаючи роботи всім іншим сервісам додатку.

Мікросервіси — архітектурний стиль за якого єдиний застосунок будується як сукупність невеличких сервісів кожен з яких працює у своєму власному процесі і зв’язується з рештою використовуючи легковагові механізми, зазвичай HTTP. Дані сервіси будуються навколо бізнес потреб, які представляє замовник проекту та розгортаються незалежно один від одного з використанням повністю автоматизованого середовища. Вони можуть бути написані з використанням різних мов програмування та технологій обробки та зберігання даних. Мікросервісна архітектура добре підходить для процесу безперервної поставки, на відміну від сервіс-орієнтовної архітектури мікросервісна архітектура направлена на створення одного застосунка в той час як сервісна-орієнтована система — являє собою множину застосунків які взаємодіють між собою [4][5].

Кожен мікросервіс розгортається окремо. Тож якщо ви змінюєте щось в одному з них, ви можете розгорнути ці зміни, не чіпаючи інших мікросервісів, які можуть продовжувати працювати. Можна вносити будь-які зміни настільки часто, наскільки потрібно, щоб додаток завжди відповідав вашим потребам. В монолітній архітектурі все інакше — будь-яка зміна потребує розгортання цілої складної системи. Мікросервіси в цілому невеликі за обсягом коду. Завдяки даній характеристиці, розробників легше їх розуміти, розроблювати та підтримувати — звісно, якщо мікросервіси створені правильно. Також легше повністю покривати тестами [6].

На рис. 3.1 показано гарний приклад мікросервісної архітектури додатку. Програма електронної комерції, яка приймає замовлення від клієнтів, перевіряє інвентар та доступний кредит, а також супроводжує їх. Додаток складається з декількох компонентів, включаючи StoreFrontUI, який реалізує

користувальницький інтерфейс, а також деякі служби підтримки для перевірки кредиту, ведення замовлень на інвентар та доставку. Додаток складається з безлічі сервісів, вони взаємодіють один з одним та не перешкоджають працювати іншим сервісам в системі [6].

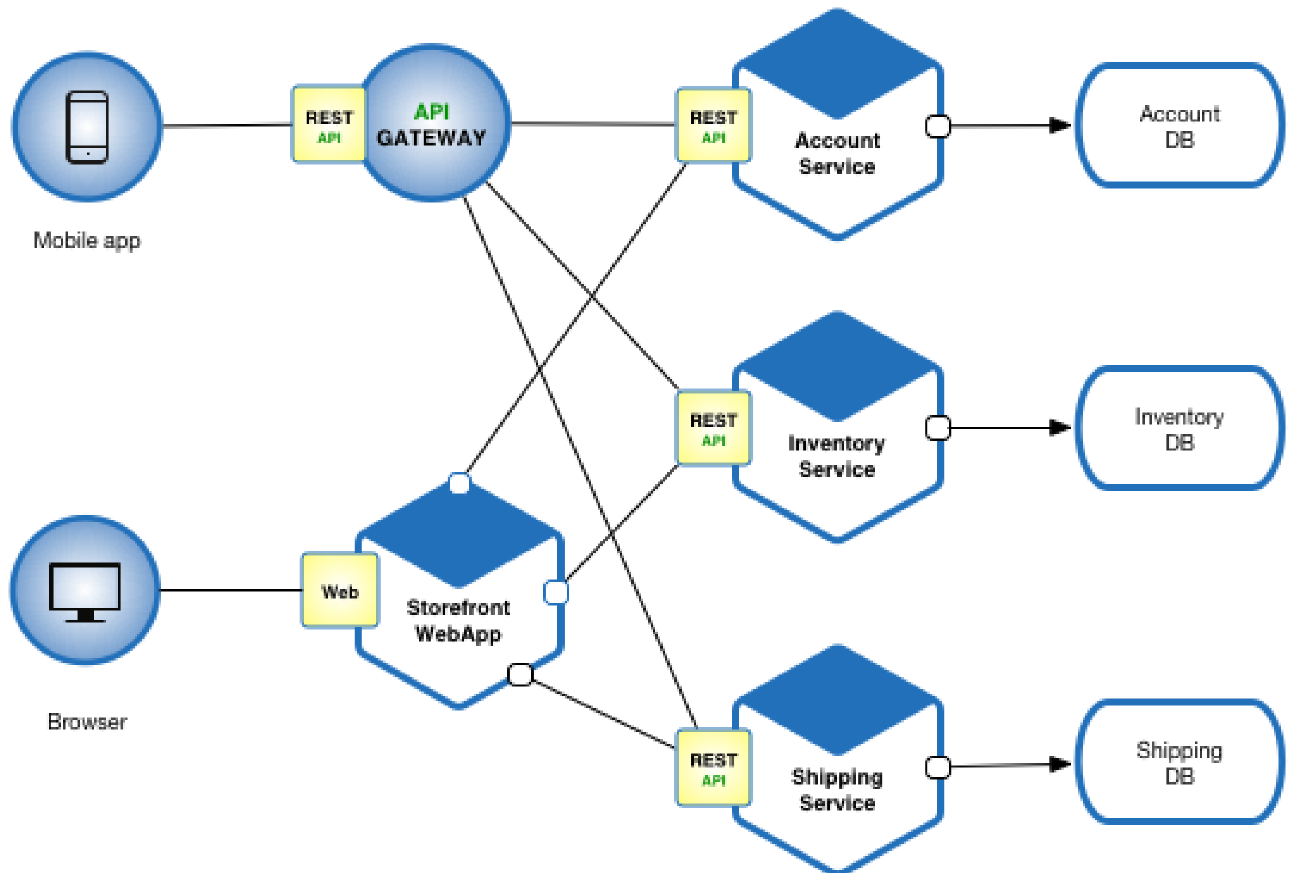


Рисунок 3.1 – Приклад архітектури мікросервісного додатку

Дане рішення має ряд переваг від інших типів архітектурних рішень:

— дозволяє безперервну доставку та розгортання великих, складних додатків — це реалізується за допомогою сучасних інструментів розгортування, наприклад: контейнерів;

— краще проходить етап тестування кожного сервісу;

— дає змогу організувати зусилля щодо розробки кількох команд. Кожна команда володіє і несе відповідальність за одну або декілька окремих сервісів. Кожна команда може розробляти, розгортати та масштабувати свої послуги незалежно від усіх інших команд;

— розподіл програмного забезпечення на дрібні, чітко визначені модулі дає командам можливість використовувати функції для кількох цілей. Сервіс, написаний для певної функції, може бути використана як будівельний блок для іншої функції. Це дозволяє додатку завантажуватися самостійно, оскільки розробники можуть створювати нові можливості без написання коду з нуля [7];

— мікросервіси дозволяють самостійно масштабувати кожний сервіс, щоб задовольнити попит на підтримку програми. Це дає змогу командам обрати потрібну інфраструктуру, точно виміряти вартість роботи сервісу та підтримувати доступність, якщо сервіс зазнає попиту [7];

— додаток запускається швидше, що робить розробку продуктивнішим та прискорює розгортання;

— кожен мікросервіс вимірює та записує свої показники роботи та моніторингу, таким чином розробники краще розумують стан кожного сервісу окремо та можуть своєчасно реагувати на помилки;

— незалежність служби збільшує стійкість програми до завершення роботи додатку від певної помилки. У монолітній архітектурі, якщо один компонент не працює, це може призвести до повного завершення роботи всієї програми. За допомогою мікросервісів програми відбивають повну відмову служби від зниження рівня функціональності, а не збої всієї програми [7];

— розробники можуть для кожного сервісу налаштувати балансування навантаження, таким чином знизити навантаження на сервер та розподілити його. В даній роботі — описуються підходи, алгоритми та розробка додатку для аналізу та навчання і в результаті правильно та коректно розподіляти навантаження;

— усуває будь-які довготривалі зобов'язання щодо технології. Розробляючи новий сервіс, можна вибрати новий стек технологій. Аналогічним чином, при внесенні значних змін до існуючого сервісу можна переписати його, використовуючи нову технологію.

Також кожна сучасна технологія має недоліки:

- розробники повинні мати справу з додатковою складністю створення розподіленої системи;
- розробники повинні впроваджувати механізм взаємодії між службами.
- впровадження задач коли потрібно використовувати декілька сервісів, що охоплюють кілька сервісів без використання розподілених транзакцій, є складним завданням для розробника;
- реалізація задач з використанням, що охоплює кілька сервісів, вимагає ретельної координації між командами;
- складність розгортання. У виробництві також існує оперативна складність розгортання та управління системою, що складається з багатьох різних етапів;
- збільшення споживання пам'яті. Мікросервісна архітектура замінює N монолітні прикладні програми з екземплярами служб NxM. Якщо кожен сервіс запускається у власному середовищі, яка необхідна для ізоляції екземплярів, то є накладні витрати в M разів більше. Більше того, якщо кожний сервіс запускається на власній віртуальній машині (наприклад, екземпляр EC2), як і у випадку з Netflix, витрати на них навіть вищі [6].

На рис. 3.2 показано архітектурний стиль додатку, яку використовує компанія Microsoft для створення власних додатків. Вони пояснюють даний тип архітектури, як сукупності невеликих автономних послуг. Кожна служба є самодостатньою і повинна реалізовувати єдину ділову спроможність [7].

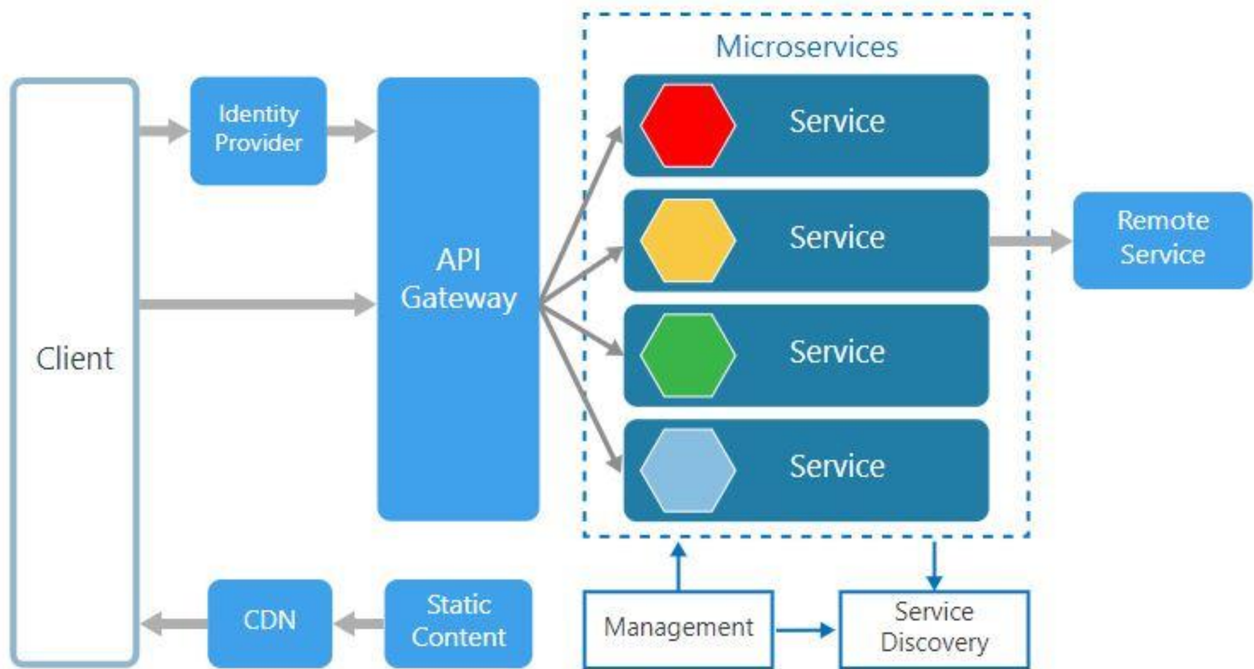


Рисунок 3.2 – Приклад мікросервісного додатку

Зробимо порівняння різних архітектур, які використовуються розробниками для проектування систем.

На рис. 3.3 ми бачимо гарний приклад монолітного та мікросервісного додатку та їх будову.

Монолітний додаток побудований як одне ціле. Додатки великого значення та обсягу за звичай в собі містять три частин, а саме:

1. клієнт — це графічний додаток, яким користувач користується та бачить перед собою у вікні монітору. По шаблонам проектування та принципам розробки програмного забезпечення, клієнтський додаток не має містити підключення та доступ до бази даних, не повинен бути навантаженим великою кількістю контролів, інтерфейс користувача повинен бути ясно зрозумілим та прозорим. Інтерфейс користувача показується основна інформація та виводиться основна та легка бізнес логіка для того, щоб користувач мав можливість швидко розібратись з додатком та навчитись в ньому працювати;
2. сервер застосунків розташовується вже на другому рівні. На цьому рівні написано більша частина бізнес-логіки. В даному випадку – на цьому

рівні закладена вся логіка роботи додатку та користувача, різні можливі функції додатку та особливості;

3. бази даних це вже третій рівень. Зазвичай це стандартна реляційна або об'єктно-орієнтована база даних. [8] [9].

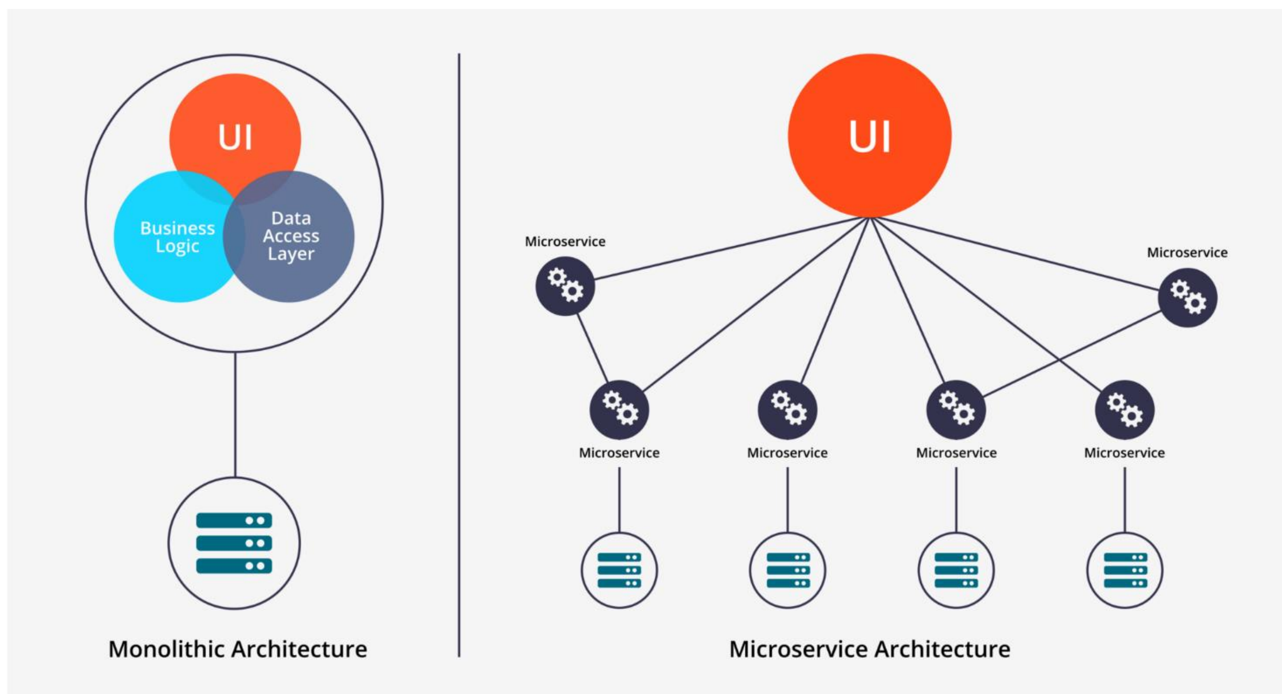


Рисунок 3.3 – Монолітна та мікро сервісна архітектура додатку

Монолітний сервер — це досить очевидний спосіб побудови подібних систем. Вся логіка по обробці запитів виконується в єдиному процесі, що саме показано на рис. 3.3 (ліва частина). Тим часом, як мікросервіси виділяють кожну певну функціональність в окремий сервіс рис. 3.3 (права частина).

Більше детально мікросервісну архітектуру можна зрозуміти з рис. 3.4. Кожний сервіс відповідає за певну одну функціональність [8].

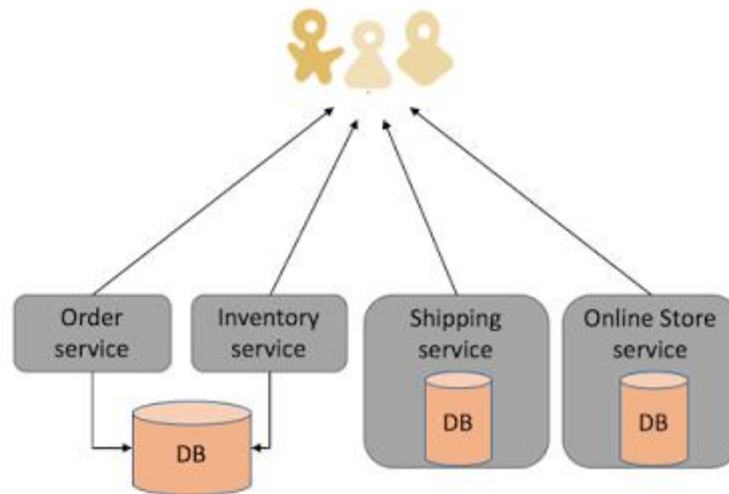


Рисунок 3.4 – Мікросервісна архітектура

Також мікросервісну архітектуру часто порівнюють з сервісно-орієнтованої архітектурою, але при порівнянні потрібно розуміти, що сервісно-орієнтована архітектура (SOA) включає в себе мікро сервісну (microservices) як ми бачимо з рис. 3.5 [8].

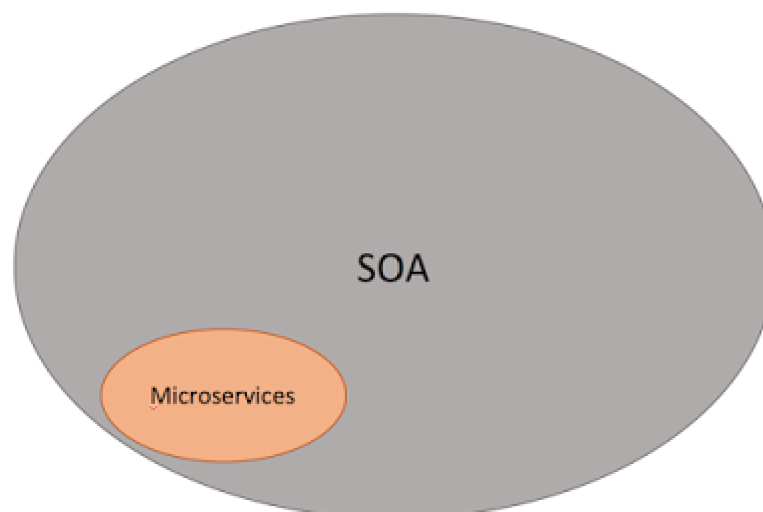


Рисунок 3.5 – SOA and Microservices

Сервіно-орієнтована архітектура може складатися з багато маленьких мікросервісів [10][11]. За словами досвідченого розробники та архітектора програмного забезпечення, що сервісно-орієнтована архітектура – це надбудова для розробки мікросервісної архітектури [11].

Рис. 3.6 якраз показує в ілюстративній формі різницю між монолітним, сервісно-орієнтованим та мікросервісною архітектурами.

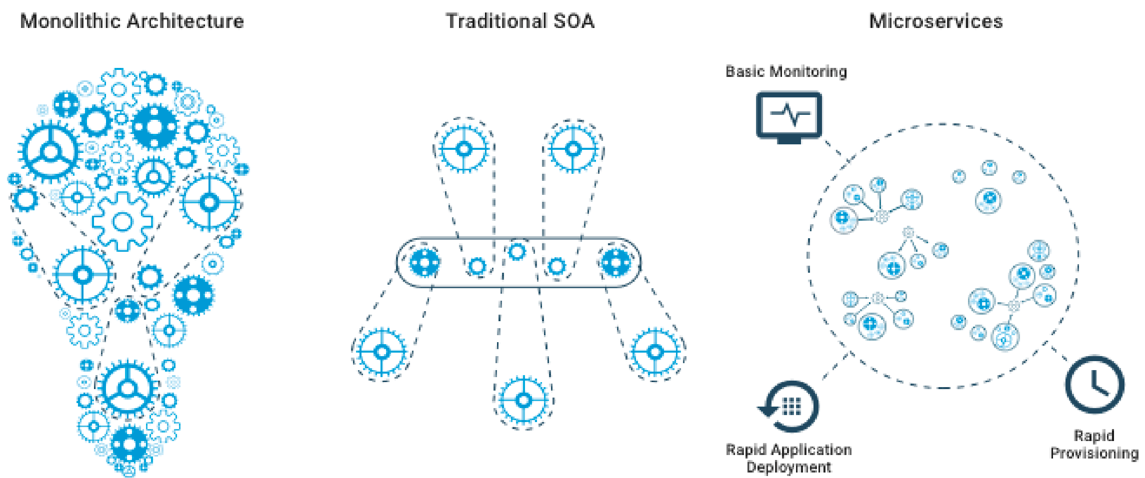


Рисунок 3.6 – Monolithic and SOA and Microservices

Мікросервісна архітектура має певні переваги поміж інших реалізацій програмного забезпечення, тому зробимо порівняльну характеристику мікросервісної, сервісна-орієнтованої та монолітної систем. Порівняння між архітектурами відображено в табл. 3.1.

Таблиця 3.1 – Порівняння різних архітектур

Характеристика	Monolithic	SOA	Microservices
Тип передачі повідомлення	-	Синхронний, очікування до з'єднання	Асинхронний: опублікувати сервіс та підписатися
Моніторинг	Потрібно розроблювати або підключати сторонні бібліотеки	Потрібно розроблювати або підключати сторонні бібліотеки	Кожний сервіс має свою моніторингову систему

Таблиця 3.1 – Порівняння різних архітектур (закінчення)

Стан системи	Зберігає та слідкує за станом системи	Зберігає та слідкує за станом системи	Без відслідковування стану системи
База даних	Масштабні реляційні (SQL)	Масштабні реляційні (SQL)	Масштабні не реляційні (NoSQL) та реляційні
Засоби збільшення та розвитку	В цілому розвивається та збільшується	Кожен сервіс еволюціонує і стає більше	Кожен сервіс незмінний і може бути залишений або виключений.
Засоби розгортування	Розгортання цілого додатку	Розгортання цілого сервісу	Розгортання окремих потрібних для змін сервісів
Засоби для зміни	Зміна монолітного додатку	Зміна всього сервісу додатку	Створення нового сервісу
Масштабування	Оптимізація моноліту	Оптимізація моноліту	Масштабування окремого сервісу
Підтримка (рядки коду обслуговування)	Тисячі і більше	Сотні, тисячі	Сотні або менше
Розуміння системи в цілому	Менш зрозуміло весь додаток	Менш зрозуміло кожен великий сервіс	Більш зрозумілий кожний невеликий сервіс

Отже з наведеної таблиці ми можемо зробити висновки того, що мікросервіси більш зрозумілі в розробці, вони не зберігають кожний раз стан системи, що дає менш використання пам’яті при збереженні певного стану. Така система легка у підтримці, бо кожен окремий сервіс містить свою невелику логіку для розуміння та невелику кількість коду, а також легка у масштабуванні. Саме цю архітектуру було обрано при розробці програмного забезпечення.

3.3 Засоби реалізації

— Операційна система

При плануванні, проектуванні та розробці системи, було зроблено наголос на універсальність системи та роботи на будь якій платформі. Дана система повинна злагоджено працювати на будь якому сервері та ОС. Було вирішено створювати саме комплекс сервісів та додатків для кроссплатформенної роботи на всіх платформах.

— Основна мова розробки сервісів системи

Для розробки різних сервісів в цілому додатку — була обрана мова програмування C#. Це об'єктно-орієнтована мова програмування з безпечною системою типізації. Вона має велику кількість бібліотек, можливостей та переваг, також дана мова програмування високорівнева, що дає можливості створювати додатки високого рівня розробки з легкістю та простотою. Дана мова дуже популярна серед розробників і тому в інтернеті можливо знайти дуже багато допоміжної літератури та прикладів по розробці додатків.

Для розробки додатку було обрано середовище розробки VS 2017 (рис. 3.7)

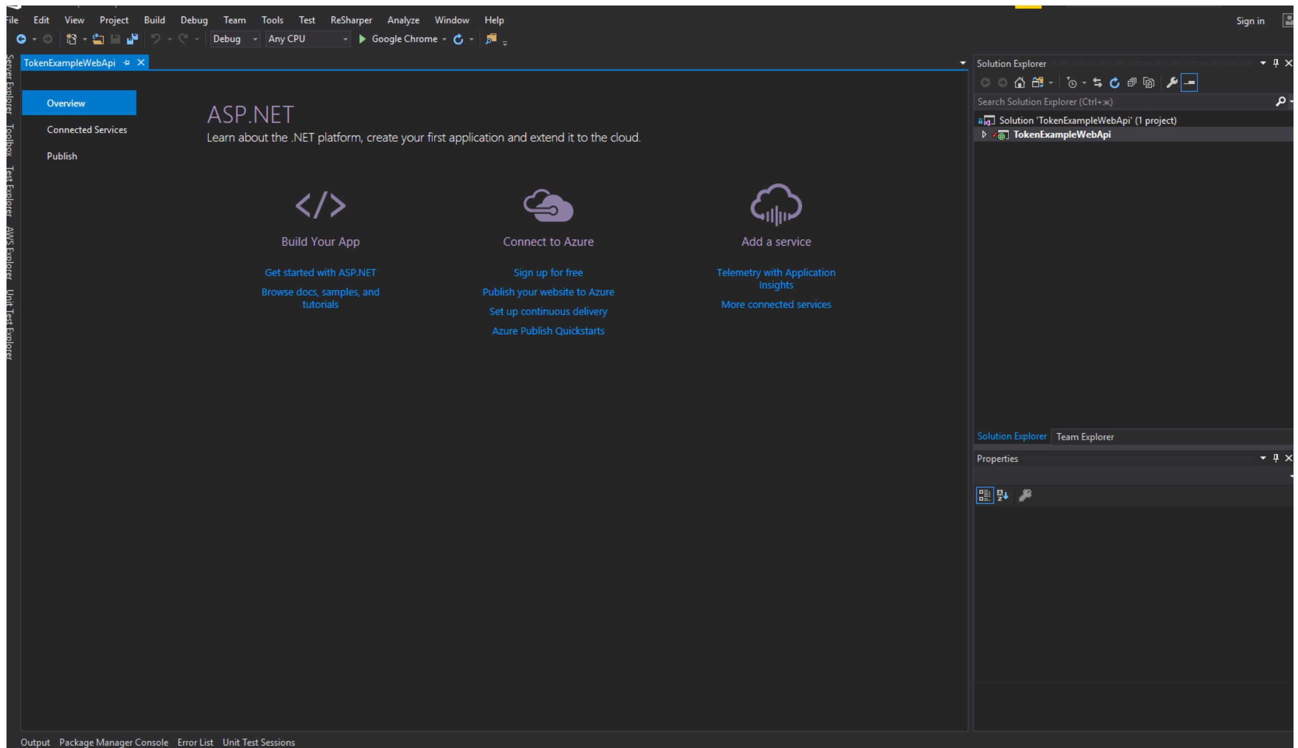


Рисунок 3.7 – Засіб розробки visual studio 2017

— Допоміжні компоненти реалізації сервісів

Для реалізації сервісів було обрано набір бібліотек .NET Core. NET Core – – це практично повне перезавантаження стека .NET Framework. З нової платформи з різних причин був виключений ряд технологій. Слід розуміти, що платформа .NET Core розрахована в першу чергу на розробку для серверних і хмарних рішень. Для десктопних додатків краще підходять класичний .NET для Windows (з підтримкою WPF і Windows Forms) і Mono для Linux і Mac OS X (з підтримкою Windows Forms). Мобільні проекти можна створювати, використовуючи Xamarin. На рис 3.8 показано, як технології розподілені всередині різних реалізацій .NET.

Зате інструменти для розробки консольних і веб-додатки отримали новий етап розвитку. При розробці більшість необхідних компонент додатки можуть завантажуватися як окремі модулі через пакетний менеджер NuGet. Це дозволяє зменшити кількість надлишкових залежностей і загальний розмір готового продукту. Саме тому вибір пав на дану технологію та набір бібліотек.

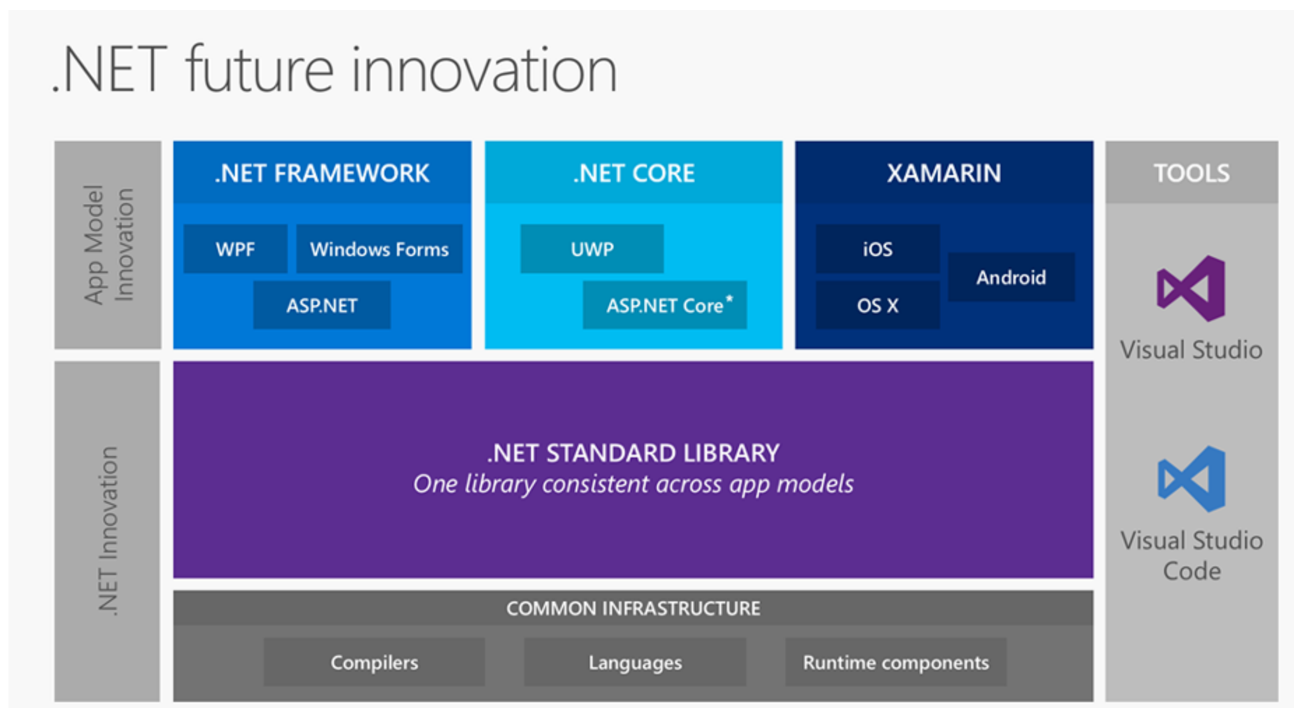


Рисунок 3.8 – Технології стеку .net framework

Виокремимо основні переваги NET Core.

- платформа з відкритим вихідним кодом — даний набір бібліотек є частиною .NET Foundation, який існує для побудови спільноти і впровадження інновацій в рамках розвитку фреймворку. Використання цієї платформи має багато переваг: ви отримуєте більше свободи в контролі і зміни проекту, прозорість коду може забезпечити вас інформацією і послужити натхненням для ваших власних проектів на базі .NET Core. Статус відкритості також дає велику стійкість, оскільки на відміну від пропрієтарного програмного забезпечення, яке часто буває забуте творцями, код, що лежить в основі інструментів цієї платформи, завжди буде залишатися загальнодоступним [12];
- створений багатьма зусиллями — у зв'язку з тим, що проект був спроектований відповідно до принципів відкритого ПЗ, платформа .NET Core побудована за допомогою близько дуже багатьох розробників. Їх внесок включав pull request'и, а також відгуки про все: від дизайну і UX до продуктивності. Впроваджуючи кращі пропозиції і побажання, команда розробників перетворила дану бібліотеку в платформу, засновану на спільнотах, що робить її більш доступною і ефективною для спільноти розробників, ніж якби вона була створена виключно всередині компанії. Платформа продовжує вдосконалюватися завдяки співпраці, оскільки вона підтримується спільнотою .NET, Microsoft і GitHub. Ви як розробник маєте можливість впливати на майбутнє просування DotNet Core, працюючи з кодом та залишаючи відгуки [12];
- ключові характеристики: NET runtime, яка надає основні сервіси, включаючи систему типів, збирач сміття, вбудований interop і збірка проекту. Примітивні типи даних, типи композиції додатків і основні утиліти надаються набором бібліотек фреймворка, також відомим як «CoreFX». Хост додатки «Dotnet» вибирає і запускає виконуючу середу, дозволяючи запускати додатки. У міру розробки ви отримаєте доступ до DotNet Core як пакет розробки програмного забезпечення (SDK). Він

включає в себе інструменти командного рядка і драйвер dotnet - все, що необхідно для створення програми або бібліотеки;

- кросплатформеність — ця унікальна програмна платформа вже працює на Windows, Mac OS X і Linux. Хоча це може здатися дивним кроком для Microsoft, але це важливо в технологічному світі, який все більше орієнтується на гнучкість і сегментування, коли справа доходить до операційних систем і платформ. Доступність на інших платформах, відмінних від Windows, робить його кращим кандидатом для використання всіма розробниками, включаючи розробників Mac і Linux, а також дає всієї платформи [12];
- ядро оснащено Agile Development Capability, в рамках крос-сумісності платформа для розробки додатків включає в себе модульну інфраструктуру. Вона видається через NuGet, і ви можете отримати доступ до пакетним функцій, а не до однієї великої збірці. Як розробник ви можете створювати легкі додатки, що містять тільки необхідні пакети NuGet, що зробить вашу програму безпечніше і продуктивніше. Модульна інфраструктура також дозволяє швидше оновлювати платформу .NET Core, оскільки порушені модулі можуть оновлюватися і випускатися окремо. Акцент на гнучкість і швидкі релізи поряд з вищезазначеним співпрацею позитивно позиціонує в рамках руху DevOps [12];
- вбудований командний рядок. Microsoft заявляє, що командний рядок слугує для того, щоб «всі сценарії продукту могли виконуватися в командному рядку». Інтерфейс командного рядка (CLI) є основою для інструментів високого рівня, таких як інтегровані середовища розробки, які використовується для створення додатків. Подібно платформі .NET Core, CLI є кросплатформним, тому ви зможете використовувати його на будь-який підтримуваної платформі. CLI .NET Core є основою для портативності додатків [12];
- схожість разом з net framework. Хоча бібліотека була розроблений як кроссплатформенная версія .NET Framework з відкритим вихідним кодом,

між ними існують і інші відмінності, які йдуть далеко за рамки цих двох ключових властивостей. Багато з цих порівнянь є результатом дизайну та новизни .NET Core. Моделі додатків, засновані на технологіях Windows, не підтримуються, але консольні і ASP.NET моделі додатків підтримують як .NET Core, так і .NET Framework. API Core менше, ніж в .NET Framework, але він у міру його розвитку він буде збільшуватися. Крім того, він реалізує тільки деякі підсистеми .NET Framework, щоб підтримувати спрощений і гнучкий дизайн платформи. Ці відмінності можуть в якійсь мірі обмежити Core, однак переваги кросплатформенного дизайну з відкритим вихідним кодом переважують будь-які обмеження в міру подальшого розширення платформи [12];

— в розробці — характер цієї платформи робить її незавершеною, постійно вдосконалюється як командою Microsoft, так і розробниками по всьому світу. Версія 1.1 призначалася для забезпечення платформи більшою функціональністю. Одна з імовірних особливостей - поліпшена підтримка API на рівні BCL. Те ж стосується і версії 2.0: постійний розвиток також підтримує Visual Basic. З бібліотеки DotNet Standard можна реалізувати посилання на бібліотеку PCL або ж на іншу бібліотеку DotNet Standard.

Саме ці та інші переваги даної бібліотеки, дали ясно розуміти, який інструмент розробки потрібно вибрати в нашому випадку для створення додатку.

— Основна мова розробки frontend частини системи

Для створення зовнішнього вигляду сторінок та саме частини frontend було обрано такий набір інструментів інструмент HTML, CSS, та JS. Також для гарного та красивого вигляду сторінку було обрано Bootstrap. Він допомагає швидше та простіше розробляти зовнішній вигляд веб-сторінок. Він підходить для людей з будь-яким рівнем досвіду, для пристроїв будь-яких форматів, та проектів будь-якого розміру.

— CI сервер

За основу CI (continuous integration) серверу було взято два сервери. Travis CI та Appveyor CI. Дані сервіси дають можливість проводити зборку всього

додатку, його розгортання на певні визначені машини, проводити тестування різних типів та робити аналіз помилок в коді за допомогою статичних аналізаторів коду.

Даний підхід розробка сервісів дуже спрощую багато різних етапів. Такий підхід дає переваги та можливість виявити певні помилки ще на стадії розробки. Даний сервіс починає робити коли було зроблено коміт в систему контролю версій. Таким чином він робить збірку та виконання тестів. Якщо тести не пройшли успішно, то система одразу сповіщає про помилки.

Було обрано обидві системи, бо кожна з них на своїй базі має різну операційну систему та різні версії встановлених пакетів та модулів. Appveyor CI — на базі операційній системі використовує Windows 10 та системну архітектуру x64, а сервіс Travis CI Linux ubuntu.14.04 та архітектуру x64.

Для злагодженої роботи використовується файл конфігурації для того, щоб сервіс знав як проводити збірку проекту та виконання тестів або його розгортання. Для сервісу Appveyor CI вся конфігурація знаходиться у файлі, який лежить в корні проекту appveyor.yml. Для іншого сервісу використовується файл .travis.yml. Код налаштування даних сервісів можна знайти у додатках.

— Git — система контролю версій

За основу, при розробці систем, було взято саме Git як розподілену систему керуванням версіями файлів. Дана система зручна та легка у використанні, а також має великий та потужній функціонал для роботи з кодом.

В цілому — це набір утиліт, які стежать і фіксують зміни в файлах в програмних кодах. З його допомогою з проектом можливо робити певні зміни та не турбуватись за стан коду, бо в будь який час можливо відкотитись до попередній версії, злити гілки або інші дії.

3.4 Засоби розгортання

Для нашого сервісу, який має велику кількість модулів в системі потрібно було опрацювати схему розгортання на різні машини та сервери та взаємодію

між ними. Було обрано вибрати продукт, який міг працювати та мати логіку роботи з контейнерами.

З даною задачею гарно справляться комплекс програмного забезпечення, а саме:

— docker;

— kubernetes;

Як контейнери Linux — незалежні середовища виконання з власними центральним процесором, пам'яттю, блоком введення-виведення і мережевими ресурсами, які використовують ядро гостьової ОС. На виході ми отримуємо щось на зразок віртуальної машини, яка працює як надбудова гостьовий операційній системи – саме завдяки даним властивостям, даний інструмент використовують для реалізації системи та архітектури розгортування [13].

У широкомасштабної системи, коли працюють віртуальні сервери, це зазвичай означає, що ви використовуєте безліч дублікатів однієї і тієї ж ОС і багато зайвих завантажувальних томів. У середовищі додатків до вимог веб-масштабування контейнери — більш привабливе середовище, ніж традиційна серверна віртуалізація [13].

Оригінальна технологія контейнерів Linux називається Linux Containers, або LXC. LXC — це метод віртуалізації на рівні операційній системи призначений для того, щоб запускати безліч ізольованих систем Linux на одному хості. Контейнери відокремлюють додатки від операційних систем. Це означає, що у користувачів є чиста мінімальна Linux система, і можна запускати всі процеси в одному або декількох ізольованих контейнерах. Так як операційна система відокремлена від контейнерів, можливо переміщати контейнер на будь-який Linux-сервер, який підтримує операційну середу контейнера [13].

Docker, який почався як проект, щоб будувати LXC-контейнери під один додаток, серйозно змінив LXC і зробив контейнери більш портативними і гнучкими.

Docker — це відкрита платформа для розробки, доставки і експлуатації додатків. Використовуючи контейнери, ви можете розгортати, копіювати, переносити і робити резервні копії інформації швидше і легше, ніж за допомогою віртуальної машини. В принципі дає хмароподібну гнучкість в будь-яку інфраструктуру, яка може працювати на контейнерах [13].

Docker обмежує контейнери, змушуючи їх працювати як єдиний процес. Якщо ваша серед додатків складається з X одночасних процесів, Система запустить X контейнерів, кожен зі своїм процесом. На відміну від докер, LXC контейнери можуть запускати безліч процесів, на цю відмінність наголошується, бо розроблювана система має багато сервісів, які запускаються в окремих процесах.

Одно процесорні контейнери мають багато переваг, включаючи прості і більш дрібні поновлення. Вам не потрібно вбивати процес баз даних, коли ви хоче оновити тільки веб-сервер. Також в даній системі з одним процесом контейнерів ефективна архітектура для того, щоб будувати додатки, засновані на мікросервісах. Одно процесорні контейнерів також є обмеження. Наприклад, ви не можете запускати агенти, срипти реєстрації або автоматично запускаються SSH-процеси всередині контейнера. Також нелегко оновлювати контейнер на рівні додатку. Доведеться запускати новий оновлений контейнер [13].

Найважливіша перевага Docker над LXC, те що більше відокремлює мережеві ресурси, сховище і деталі операційної системи, ніж LXC. З докер додаток дійсно не залежить від налаштувань цих низькорівневих ресурсів. Коли ви переміщаєте контейнер від одного хоста до іншої машини з докер, то він гарантує, що навколишнє середовище для додатка залишиться незмінною. Пряме перевага цього підходу — це те, що докер допомагає програмістам створювати локальні середовища розробки, які виглядають як продакшн-сервер. Коли програміст закінчує писати і починає тестувати код, він може обернути його в контейнер, опублікувати безпосередньо на сервері або в приватному хмарі, і він відразу буде працювати, так як це одне і та ж середовище. CLXC програміст

може запустити щось на своїй машині, але виявити, що код не працює належним чином при розгортанні на сервері. Середа сервера буде інша, і програмісту доведеться витратити багато часу, щоб полагодити цю різницю і виправити проблему [13].

Kubernetes – це ще один сервіс, який допомагає вирішувати різнопланові задачі для керування контейнерами. Це інструмент для роботи над контейнерами, які находяться в кластері. Зазвичай з багатьох контейнерів формують кластер та з ним працюють, бо таким чином простіше слідкувати за статусом та станом кластеру. Також можливо встановлення на різних серверах. Даний інструмент вирішує, в якій частині кластера контейнер буде запущений, стежить, щоб його запитана конфігурація («запущений», «п'ять реплік») виконувалася, щоб у контейнера була мережа, налаштований доступ ззовні (якщо потрібно), поновлення приходили в потрібному порядку.

Компанія Google користується контейнерної технологією вже більше десяти років. Вона починала з запуску більш 2 млрд контейнерів протягом одного тижня. За допомогою проекту Kubernetes компанія ділиться своїм досвідом створення відкритої платформи, призначеної для масштабування запуску контейнерів.

Проект переслідує дві мети. Якщо розробники користуються контейнерами Docker, виникає наступне питання про те, як масштабувати і запускати контейнери відразу на великій кількості хостів, а також як виконувати їх балансування. У проекті пропонується високорівнева API, що визначає логічне групування контейнерів, що дозволяє визначати пули контейнерів, балансувати навантаження, а також задавати їх розміщення.

Також саме наша робота проекту полягає в тому, що правильно проаналізувати навантаження на контейнер, який працює, навчити нейронну мережу правильно обробляти та розподіляти запити по різним машинам – для того, щоб стабілізувати систему та розподілити навантаження.

3.5 Архітектура обміну повідомленнями

В даному підрозділі основна мета — це проведення аналізу додатків для обміну повідомлень, характеристика кожного з них, опис сильних та слабких сторін. Система повинна сповіщати адміністраторів системи та користувачів про певні зміни або надати характеристики мережі, сповіщати про певні неполадки або надавати корисну інформацію.

Даний програмний засіб повинен вирішувати такі поставлені задачі:

1. бути легким у використані (привабливий інтерфейс користувача);
2. можливість для встановленням на будь яку операційну систему та використання з будь яким девайсом;
3. наявність відкритого API та належного опису документації;
4. швидкий у робі;
5. можливість передавання графічних зображень та посилань на різні веб ресурси;
6. можливість інтегрування з різними сервісами для роботи (Github, trello, jira, google drive);
7. можливість створити власного бота для обміну системними повідомленнями.

Для кращого розуміння зробимо порівняння найкращих додатків для обміну повідомлення у вигляді таблиці (табл. 3.3).

Таблиця 3.3 – Порівняння додатків для обміну повідомлень

Характеристика	Slack	Telegram	Facebook messenger	Viber	Skype
Підтримка операційних систем (1 – windows; 2 – Linux; 3 – mac os x; 4 – android; 5 – ios)	1) + 2) + 3) + 4) + 5) +	1) + 2) + 3) + 4) + 5) +	1) + 2) + 3) + 4) + 5) +	1) + 2) + 3) + 4) + 5) +	1) + 2) + 3) + 4) + 5) –
Зручність використання	Так	Так	Так	Так	Так

Таблиця 3.3 – Порівняння додатків для обміну повідомлень (закінчення)

Шифрування повідомлень	Так	Так	Так	Так	Так
Можливість відправка файлів	Так	Так	Так	Так	Так
Підтримка ботів	Так	Так	Так	Ні	Так
Відкрите API для використання	Так	Так	Так	Так	Так
Розширена документація API	Так	Так	Так	Ні	Ні
Можливість інтеграція з іншими сервісами (Github, trello)	Так	Так	Ні	Ні	Ні
Можливість створювати незалежні канали (команди)	Так	Ні	Ні	Ні	Ні
Для користуванням не потрібно сім карту	Так	Ні	Так	Ні	Так

З великої кількості основних додатків для отримання повідомлень ми використали основні та зробили порівняльний аналіз.

Всім затребуваним вимогам відповідали два додатки — це Slack та Telegram. Вони мають такі переваги:

1. взаємодія з іншими сервісами напряму (github, trello, helpScout, google drive, dropbox, heroku);
2. можливість створення ботів для сповіщення користувачів;
3. швидкі у роботі;
4. привабливий дизайн інтерфейсу;
5. можливість поширення файлів різного розширення та рисунків;
6. можливість роботи з вставками коду та їх синтаксичного підсвітлення ключових слів;
7. доступність роботи на всіх платформах;
8. ідеально підходить для командної роботи;

9. робота з JSON форматом для відправки повідомлень та сповіщень користувачам;
10. відкритий доступ до API та повна документація для роботи.

Важливим перевагами даних додатків перед іншими додатками так це вільний доступ до API та велика кількість прикладів коду, як саме правильно використовувати додаток для обміну повідомленнями.

На рис. 3.5 можемо побачити графічний інтерфейс додатку.

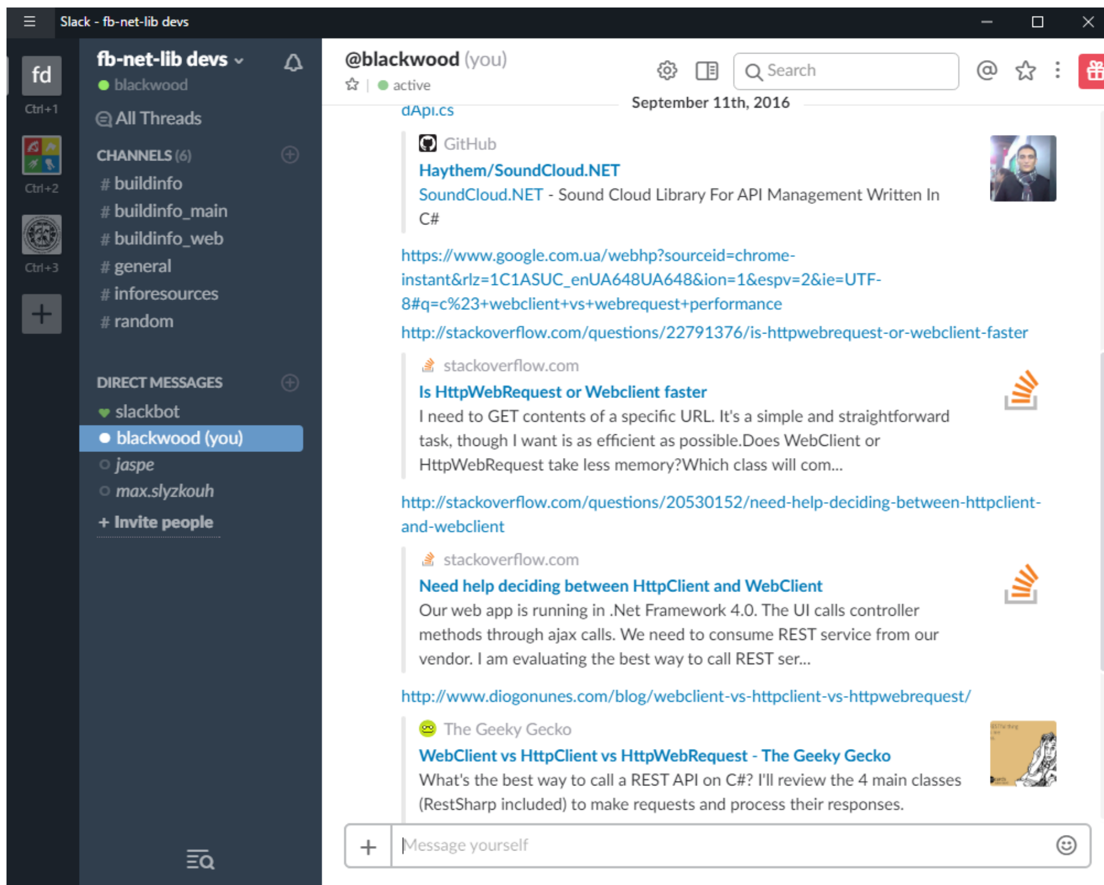


Рисунок 3.5 – Графічний інтерфейс додатку Slack

Slack для авторизації використовую протокол безпечності OAuth 2.0. Він дозволяє авторизації клієнтського застосування запитувати особисті дані в обліковому записі користувача Slack без отримання пароля. Він використовує 6 етапів для авторизації:

1. клієнтський додаток робить запит на отримання доступів;
2. сервіс Slack перевіряє наявність та коректність всіх введених полів в запит, який приходить та відправляє дозвіл на доступ;

3. клієнтський додаток робить запит на саме отримання потрібних доступів в додатку;
4. сервіс отримує запит на отримання доступів та відправляє унікальний токен;
5. тепер додаток має можливість відправляти певні запити на отримання повідомлень або інші запити разом з унікальним токеном;
6. сервіс Slack перевіряє на актуальність токен та відправляє потрібний запит користувачу.

На рис. 3.6 можемо побачити схему авторизації клієнтського додатку з додатком Slack для коректної роботи.

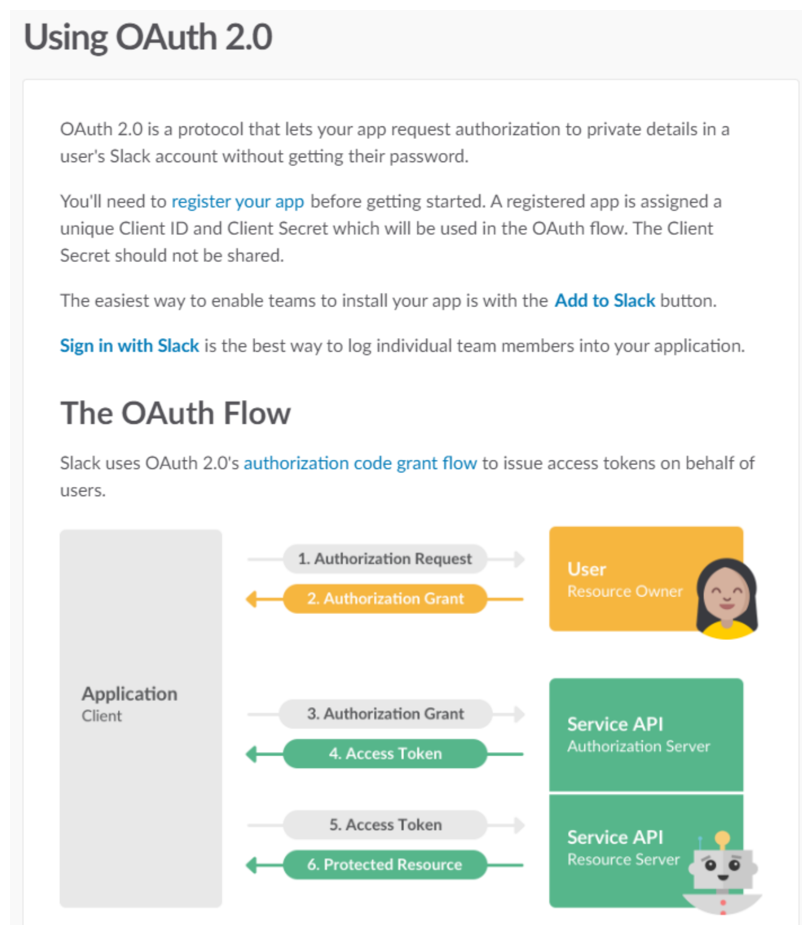


Рисунок 3.6 – Схема авторизації клієнтського додатку з Slack

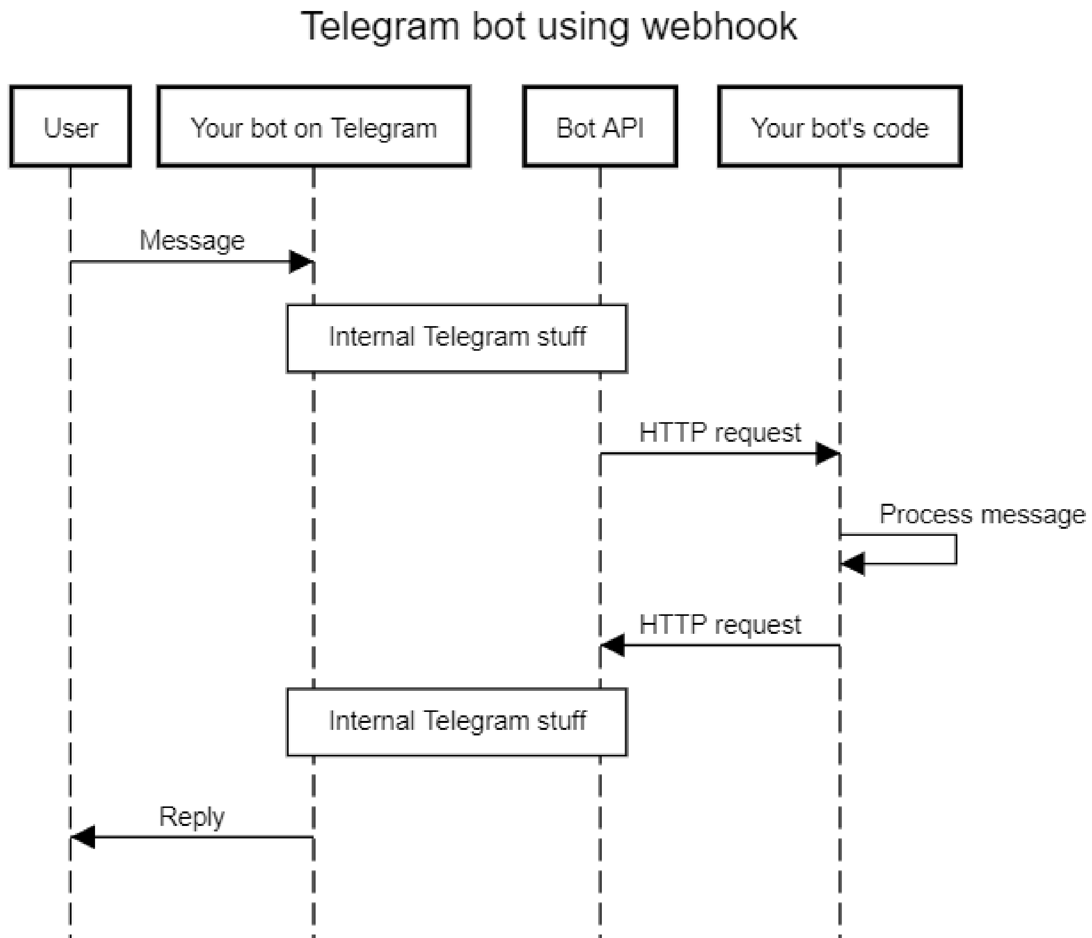


Рисунок 3.7 – Схема алгоритму роботи додатку з Telegram

На рис. 3.7 бачимо алгоритм роботи з ботом для додатку telegram. Дана платформа одна з легких для розробки. Процес починається з того, що потрібно викликати спеціального бота BotFather та натискаємо кнопку start, вибираємо команду \newbot, після чого, послідовно відповідаючи на питання, вказуємо ім'я бота і його користувача ім'я. У фіналі Telegram повідомить, що бот успішно створений, надасть його ключ і запропонує вказати додаткове опис або ввести назви команд.

Технічно э два способи, щоб обробити повідомлення і команди бота. Перший полягає, в тому, що ваш сервіс періодично опитує сервер Telegram на наявність змін.

Другий спосіб вимагає налаштування webhook, тобто сервісу, розміщеного за https адресою, який буде обробляти зміни. Цей спосіб менше навантажує сервера, але вимагає наявності сертифіката. Втім, проблема з сертифікатом легко

вирішується завдяки сервісу Ngrok (https://ngrok.com/), який може здійснювати тунелювання запитів з https-адреси на адресу сервісу на вашій машині.

Щоб скористатися Ngrok, необхідно:

1. зареєструватися на сайті та отримати персональний ключ;
2. встановити ngrok.exe і в командному рядку ввести ngrok authtoken ключ Ngrok;
3. виконати команду ngrok http -host-header="localhost:порт" порт;

Висновок до розділу

В розділі було зроблено аналіз монолітної, сервісно-орієнтованої архітектури та мікросервісної, описано сильні та слабкі сторони кожної з них та обрано саме підходящу архітектуру для додатку.

Було розглянуто багато різних додатків для обміну повідомленнями та вибрано два сервіси, які відповідають всім затребуваним характеристикам. Вибір зупинився на Slack та Telegram. Кожний з даних додатків має свої переваги тому саме вибір був спрямованих на них, та основною перевагою є – те що є в наявності відкритий API сервіс для розробки.

Також було зроблено опис всіх інструментів та допоміжних пакетів, які допомагали розроблювати даний сервіс.

Наступний розділ буде в собі містити опис використаних алгоритмів для балансування навантаження на сервер разом з деякими частинами коду та описом. Кожний алгоритм має певну реалізацію та буде описано, яким чином вдалось їх реалізувати.

4 ОПИС ВИКОРИСТАНИХ АЛГОРИТМІВ

У розділі було детально досліджено різні популярні та оптимальні алгоритми, які використовуються для балансування навантаження на сервер, а також вибрані декілька та реалізовані для даної магістерської роботи.

4.1 Алгоритм Round Robin

Round Robin, або алгоритм кругового обслуговування, являє собою перебір по круговому циклу: перший запит передається одного сервера, потім наступний запит передається іншому і так до досягнення останнього сервера, а потім все починається спочатку [14].

Суть алгоритму є в тому, що в системі є N об'єктів, здатних виконати задану дію, і M завдань, які повинні бути виконані цими об'єктами. Мається на увазі, що об'єкти n рівні за своїми властивостями між собою, завдання m мають рівний пріоритет. Тоді перше завдання (m = 1) призначається для виконання першого об'єкту (n = 1), друга - другого і т. Д., До досягнення останнього об'єкта (m = N). Тоді наступне завдання (m = N + 1) буде призначена знову першому об'єкту і т. П. Простіше кажучи, відбувається перебір виконують завдання об'єктів по циклу, або по колу (round), і після досягнення останнього об'єкта наступне завдання буде також призначена першому об'єкту . Рішення задач може бути додатково розбите на кванти часу, причому для продовження рішення в часі нумерація об'єктів (і, відповідно, призначені завдання) зсувається по колу на 1, тобто завдання першого об'єкта віддається другому, другого - третього, і т. Д., а перший об'єкт отримує завдання останнього, або звільняється для прийому нового завдання. Таким чином, алгоритм Round-robin стає алгоритмом розподілу часу або балансування навантаження [15].

Найпоширенішою імплементацією цього алгоритму є, звичайно ж, метод балансування Round Robin DNS. Як відомо, будь-який DNS-сервер зберігає пару «ім'я хоста - IP-адреса» для кожної машини в певному домені.

DNS-сервер проходить по всіх записів таблиці і віддає на кожен новий запит наступний IP-адреса: наприклад, на перший запит - xxx.xxx.xxx.2, на

другий - xxx.xxx.xxx.3, і так далі. В результаті всі сервери в кластері отримують однакову кількість запитів [14].

Переваги:

1. не залежить від протоколу;
2. не вимагає зв'язку між серверами;
3. низька вартість.

Недоліки:

1. сервера повинні бути однаковими по ресурсам;
2. немає обліку завантаження конкретного сервера;
3. не враховується кількість активних підключень;
4. кешування DNS у клієнта.

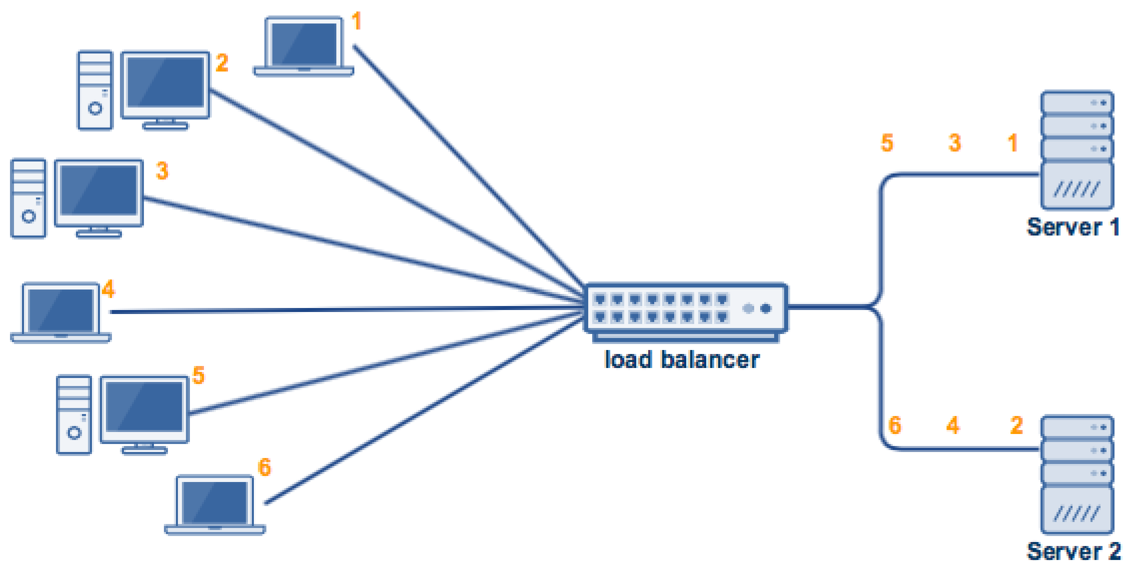


Рисунок 4.1 – Графічне зображення алгоритму «Round Robin»

Наприклад, з рис. 4.1 бачимо, що є 2 сервери, які чекають запитів за вашим балансувальником навантаження. Після того, як перший запит надійде, балансувальник відправить цей запит на 1-й сервер. Коли надходить другий запит (імовірно, від іншого клієнта), цей запит буде переадресовано на другий сервер.

Оскільки другий сервер є останнім у цьому кластері, наступний запит (наприклад, 3-й) буде перенесено на перший сервер, четвертий запит на другий сервер і так далі, циклічно.

У силу описаних вище обставин сфера застосування алгоритму Round Robin вельми обмежена. Реалізацію даного алгоритму буде приведено код у додатках.

4.2 Алгоритм Weighted Round Robin

Це вдосконалена версія алгоритму Round Robin. Суть вдосконалення полягає в наступному: кожному серверу присвоюється ваговий коефіцієнт згідно із його продуктивністю і потужністю. Це допомагає розподіляти навантаження більш гнучка: сервери з великою вагою обробляють більше запитів [14].

Так, наприклад, на рис. 4.2 бачимо, якщо пропускна здатність сервера 1 становить 5x більше, ніж сервер 2, то ви можете призначити йому вагу 5, а сервер 2 - вагою 1.

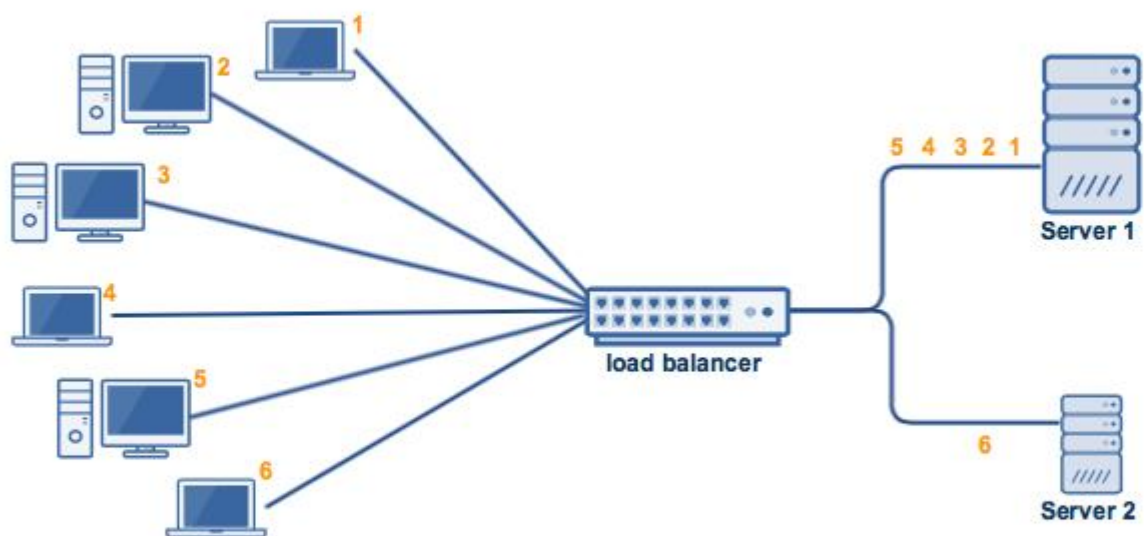


Рисунок 4.2 – Графічне зображення алгоритму «Weighted Round Robin»

Тому, коли клієнти починають входити, перші 5 будуть призначатися для вузла 1 та 6-го для вузла 2. Якщо з'явиться більше клієнтів, буде дотримуватися така ж послідовність. Тобто, 7, 8, 9, 10 і 11 всі йдуть на Server1, а 12-й на сервер 2 і так далі.

Перевагами даного алгоритму є вдосконалення алгоритму Round Robin, по більшим критерієм параметрів для сервера та різним його властивостям. Реалізацію даного алгоритму буде приведено код у додатках.

4.3 Задача комівояжера

Даний алгоритм комівояжера допомагає вирішити дуже критичну задачу, а саме знаходження найкоротшого шляху задачі від одного сервера до іншого за допомогою алгоритму Літла, який опишемо в даному розділі.

Алгоритм Літтла застосовують для пошуку рішення задачі комівояжера у вигляді гамільтонового контуру. Даний алгоритм використовується для пошуку оптимального гамільтонова контуру в графі, що має N вершин, причому кожна вершина i пов'язана з будь-якою іншою вершиною j двобічної дугою. Кожній дузі приписана вага Ci, j, причому ваги дуг строго позитивні (Ci, j≥0). Ваги дуг утворюють матрицю вартості. Всі елементи по діагоналі матриці прирівнюють до нескінченності (Cj, j = ∞) [16].

У випадку, коли пара вершин i i j не пов'язана між собою (граф не повно зв'язний), то відповідному елементу матриці приписуємо вага, рівний довжині мінімального шляху між вершинами i та j. Якщо в результаті дуга (i, j) увійде в результуючий контур, то її необхідно замінити відповідним їй шляхом. Матрицю оптимальних шляхів між усіма вершинами графа можна отримати застосувавши алгоритм Данцига або Флойда [16].

Алгоритм Літала є окремим випадком застосування методу "гілок та меж" для конкретного завдання. Загальна ідея тривіальна: потрібно розділити величезне число та перебираються варіанти на класи для того, щоб отримати оцінки (знизу - в задачі мінімізації, зверху - в задачі максимізації) для цих класів, щоб мати можливість відкидати варіанти не по одному, а цілими класами. Складність полягає в тому, щоб знайти такий поділ на класи (гілки) і такі оцінки (межі), щоб процедура була ефективною [16].

Зробимо опис даного алгоритму по етапам:

1. у кожному рядку матриці потрібно знайти мінімальний елемент і відняти його з усіх елементів рядка. Зробити це потрібно і для стовпців, що не містять нуля. Отримаємо матрицю, кожен рядок і кожен стовпець якої містять хоча б один нульовий елемент;
2. для кожного нульового елемента матриці сij розрахуємо коефіцієнт Gi, j, який дорівнює сумі найменшого елемента i рядки (виключаючи елемент Ci, j = 0) і найменшого елемента j стовпця. З усіх коефіцієнтів Gi, j виберемо такий, який є максимальним GK, l = max {Гі, j}. У гамільтонів контур вноситься відповідна дуга (k, l);
3. потрібно видалити k рядок і стовпець l, поміняємо на нескінченність значення елемента Cl, k (оскільки дуга (k, l) включена в контур, то зворотний шлях з l в k неприпустимий);
4. повторюємо алгоритм кроку 1, поки порядок матриці не стане рівним двом;
5. потім в поточний орієнтований граф вносимо дві відсутні дуги, що визначаються однозначно матрицею прядка 2. Отримуємо гамільтоновий контур [16].

Даний алгоритм бере участь при пошуку та аналізу проходження задачі через сервера в системі, котрій працює для того, щоб знайти найменший шлях до якого серверу задачі потрібно подолати та виконати.

Висновки до розділу

В даному розділі було зроблено опис тих алгоритмів, які були реалізовані в системі та використовуються для вирішення задачі балансування навантаження на сервер. Було оглянуто два основні алгоритми, які використовуються в системі, Round Robin та більш оптимізований алгоритм Weighted Round Robin — це вдосконалена версія алгоритму Round Robin, який вдосконалений тим, що кожному серверу в системі за вдяки великій кількості параметрів проставляється ваги та індекс роботи, далі при розподіленні задач по різним серверам або кластерам дані параметри беруться до уваги та аналізується.

Наступний розділ присвячений опису алгоритмам існуючих нейронних мереж, опису процесу навчання та аналізу завдяки багатьом параметрам. Також в розділі описано вибраний алгоритм та представлено основні можливості та переваги даного алгоритму – багатошарового перцептрона.

Також в розділі зроблений детальний опис реалізації алгоритму, тому що система в основному базується на аналізі параметрів та в подальшому навчанні нейронної мережі, щоб без участі різних користувачів або розробників вона стабільно працювала та не потребувала змін.

В наступному розділі описуються методи тестування мережі, а також метод розробки, який був використаний, а саме TDD, що дало змогу на кожній ітерації розробки контролювати процес роботи мережі та не робити помилки в системі та контролювати якість коду, його покриття.

5 ОПИС НЕЙРОННОЇ МЕРЕЖІ

5.1 Існуючі нейронні мережі

На сьогоднішній день — дуже поширена практика використання нейронної мережі при написані певних сервісів та додатків. Саме дана мережа дає можливість вирішувати певні задачі, які б людина вирішувала тривалий проміжок часу та на це пішло багато сил та ресурсів.

Нейронні мережі бувають різними за характеристиками та параметрами, а також за своє реалізацією. Існує певна кількість алгоритмів, які будують мережу за певними правилами та алгоритмами.

У цьому розділі зробимо короткий опис існуючих алгоритмів для побудови мережі, більш детально зробимо опис вибраного алгоритму побудови та наведемо прикладу коду за допомогою, якого була зроблена реалізація даного алгоритму.

Взагалі навчання нейронної мережі — це дуже затратний по часу процес, в якому параметри нейронної мережі налаштовуються за допомогою моделювання середовища, в яку ця мережа працює та повинна виконувати свої задачі. Тип навчання визначається способом підстроювання параметрів. Бувають різні типи навчання, а саме: навчання з вчителем та без вчителя [17].

Для того, щоб мережа могла виконати певні задачі її потрібно навчити і тому на рис. 5.1 показано схематична ілюстрація процесу навчання.

Процес навчання з вчителем є надання мережі вибірки навчальних прикладів, тобто користувач або інша система в нашому випадку — надає мережі данні, за якими вона може навчатися, аналізувати їх та розвиватись. Кожен зразок подається на вхід мережі, потім проходить обробку, яка займає певний час всередині структури, обчислюється вихідний сигнал мережі, який порівнюється з відповідним значенням цільового вектору, що представляє собою необхідний вихід мережі.

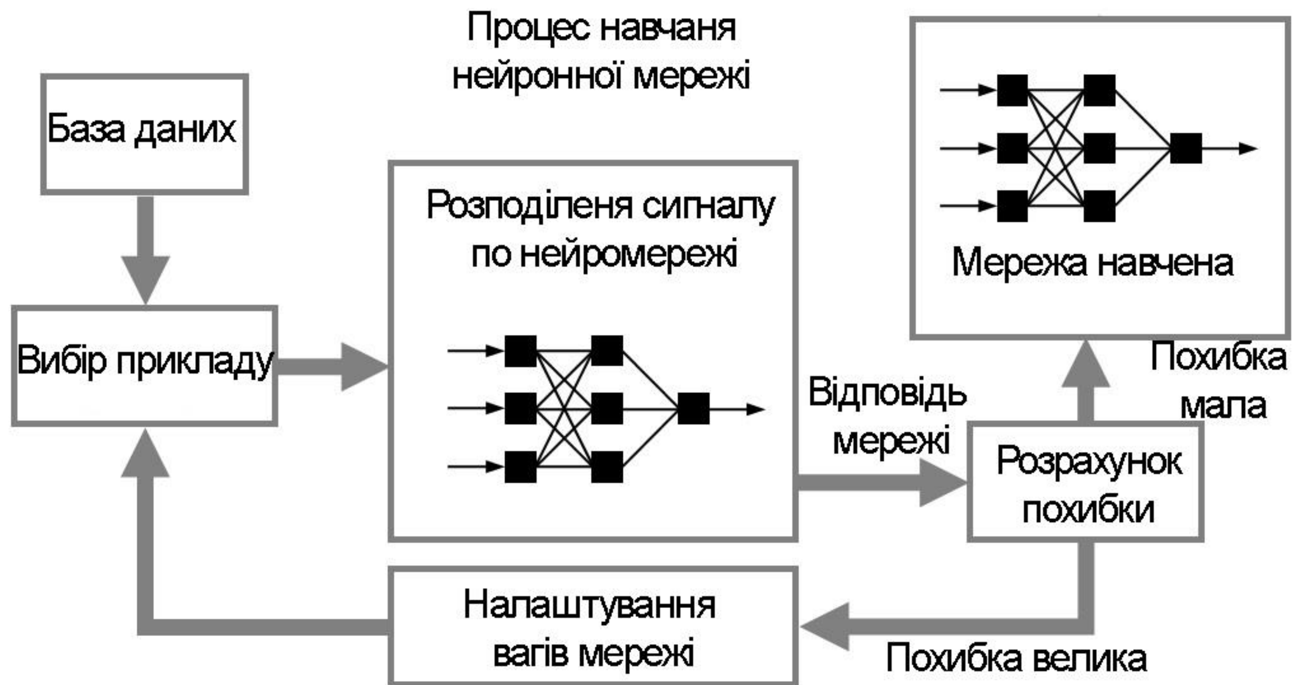


Рисунок 5.1 – Схематична ілюстрація процесу навчання

При навчанні без вчителя мережу множина з якою виконується навчання складається лише з вхідних векторів. Навчальний алгоритм підлаштовує ваги мережі так, щоб виходили узгоджені вихідні вектори, тобто щоб пред'явлення досить близьких вхідних векторів давало однакові виходи. Процес навчання, отже, виділяє статистичні властивості навчальної множини і групує подібні вектори в класи. Пред'явлення на вхід вектору з даного класу дасть певний вихідний вектор, але до навчання неможливо передбачити, який вихід буде проводитися даним класом вхідних векторів. Отже, виходи подібної мережі повинні трансформуватися в деяку зрозумілу форму, зумовлену процесом навчання. Це не є серйозною проблемою. Зазвичай не складно ідентифікувати зв'язок між входом і виходом, встановлену мережею [17].

Для навчання нейронних мереж без вчителя застосовуються сигнальні метод навчання Хебба і Ойа. Перерахуємо алгоритми, які частіше за все використовуються при розробці мережі:

1. алгоритми локальної оптимізації з обчисленням похідних першого порядку:

- градієнтний алгоритм (метод найшвидшого спуску);
- методу сполучених градієнтів;
- методи, що враховують напрямок антиградієнта на кілька кроків алгоритму;

2. алгоритми локальної оптимізації з обчисленням похідних першого і другого порядку:
   - метод Ньютона;
   - методи оптимізації з розрідженими матрицями Гессе;
   - квазінютоновскі методи;
   - метод Гаусса-Ньютона;
   - метод Левенберга-Марквардта;
3. стохастичні алгоритми оптимізації:
   - метод Монте-Карло
4. алгоритми глобальної оптимізації (задачі глобальної оптимізації вирішуються за допомогою перебору значень змінних, від яких залежить цільової функції) [17].

Градієнтний алгоритм або спуск

Загальна проблема, з якою ми маємо справу — це пошук параметрів для виконання деяких цільових функцій — не є специфічним для машинного навчання. Насправді це дуже загальна проблема, знайдена в математичній оптимізації, відома нам протягом тривалого часу і зустрічається в набагато більших сценаріях, ніж просто нейронні мережі. Сьогодні багато проблем у багатомірній оптимізації функцій, включаючи навчання нейронних мереж, зазвичай покладаються на дуже ефективний алгоритм, який називається градієнтним спуском для пошуку набагато швидшого, ніж випадкові здогадки, і більш потужним, ніж лінійна регресія.

Інтуїтивно, як працює градієнтний спуск, це схоже на аналогію з гірським альпіністом. По-перше, ми починаємо з випадкової догадки за параметрами і починаємо там. Потім ми з'ясовуємо, в якому напрямку функція втрат

найчастіше падає вниз (щодо зміни параметрів), і трохи кроком в цьому напрямку. Іншими словами, ми визначаємо суми, щоб налаштувати всі параметри таким чином, щоб функція втрат зменшилась на найбільшу суму. Ми повторюємо цей процес знову і знову, поки ми не задоволені, ми знайшли найнижчу точку [17].

Щоб з'ясувати, в якому напрямку спуск найчастіше нахиляється вниз, необхідно розрахувати градієнт функції втрат по відношенню до всіх параметрів. Градієнт — це багатовимірне узагальнення похідної; це вектор, що містить кожну з приватних похідних функції за кожною змінною. Іншими словами, це вектор, який містить нахил функції втрат уздовж кожної осі.

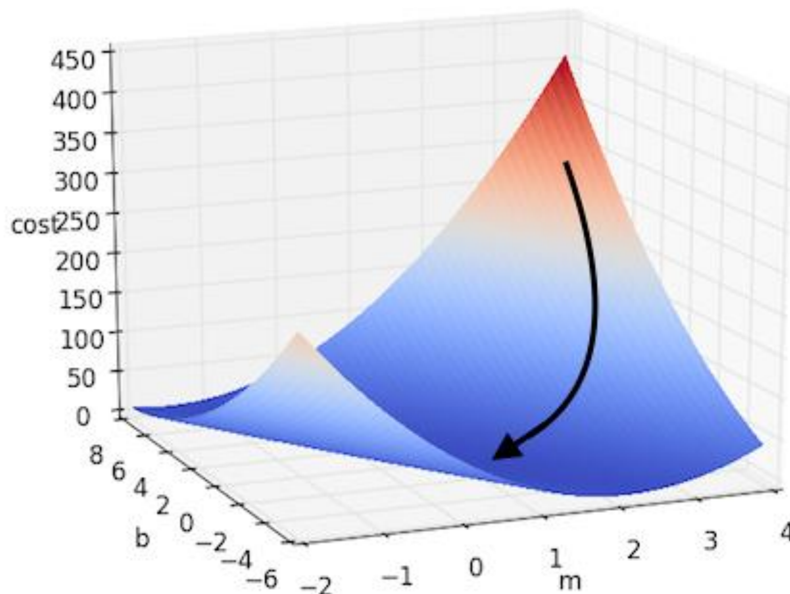


Рисунок 5.2 – Схематична ілюстрація процесу навчання

На рис. 5.2 показано приклад градієнтного спуску для лінійної регресії з двома параметрами. Ми приймаємо випадкові здогадки за параметрами і повторно оновлюємо нашу позицію, взявши невеликий крок у напрямку градієнта, поки ми не опинимося внизу функції спуску.

Імпульс

Імпульс відноситься до сімейства варіантів градієнтного спуску, де оновлення ваг має інерцію. Іншими словами, оновленні ваги вже не є функцією

просто градієнта на поточному етапі часу, але поступово коригується з швидкістю попереднього оновлення.

При стандартному градієнтному спуску ми обчислюємо градієнт і використовуємо наступну формулу оновлення параметрів із швидкістю навчання $W_t := W_t - \alpha \nabla J(W_t)$.

Зауважмо, що ми додали t індекс, щоб позначити поточний момент часу, який раніше був пропущений. На відміну від цього, загальна формула для градієнтного спуску з імпульсом полягає в наступному:

$z_t := \beta z_{t-1} + \nabla J(W_{t-1})$

$W_t := W_{t-1} - \alpha z_t$

Також дуже популярний алгоритм, який поєднує в собі два попередніх, а саме алгоритм стохастичного градієнта з імпульсом. Даний алгоритм являє собою вдосконалений алгоритм стохастичного градієнта [18].

Момент або алгоритм стохастичного градієнта з імпульсом — це метод, який допомагає прискорити вектори градієнтів у правильному напрямку, тим самим призводячи до швидшого зближення. Це один з найпопулярніших алгоритмів оптимізації, і багато хто з найсучасніших моделей навчаються за допомогою цього. Перш ніж перейти до рівнянь оновлення алгоритму, проаналізуємо математику, яка лежить в основі роботи імпульсу [19].

Експоненціальна зважений середній показник

Експоненціальні зважені середні значення мають послідовність чисел. Припустимо, у нас є деяка послідовність S, яка шумно. Для цього прикладу побудована косинус функцію і додана деякі гауссові шуми. Це виглядає так на рис 5.2:

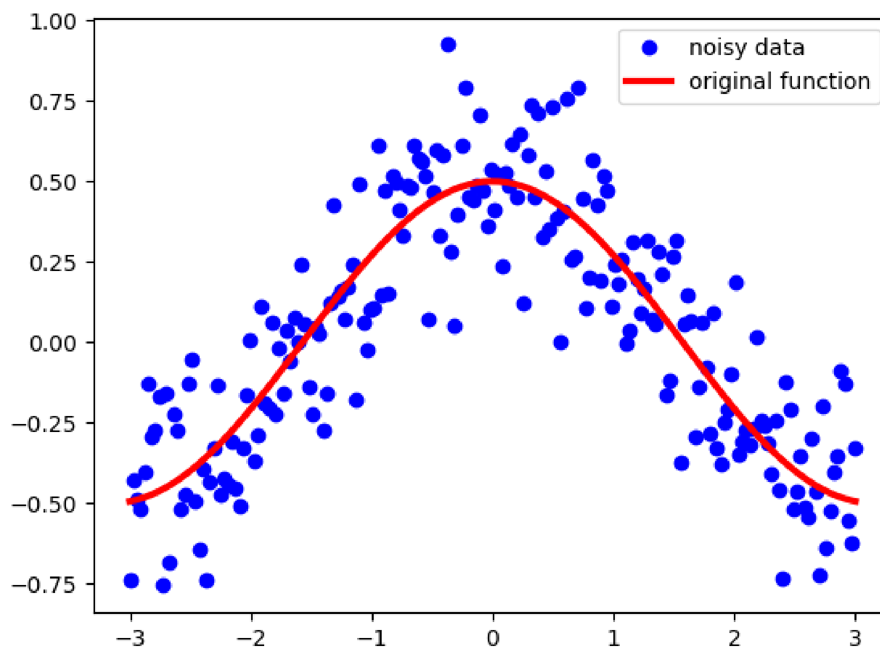


Рисунок 5.2 – Вихідна послідовність

Зауважмо, що навіть якщо ці точки виглядають дуже близько до кожного, ніхто з них не має координат x . Це унікальний номер для кожної точки. Ось це число визначає індекс кожної точки в нашій послідовності S.

Експоненціально зважені середні показники можуть дати нам зображення, яке виглядає так на рис. 5.3:

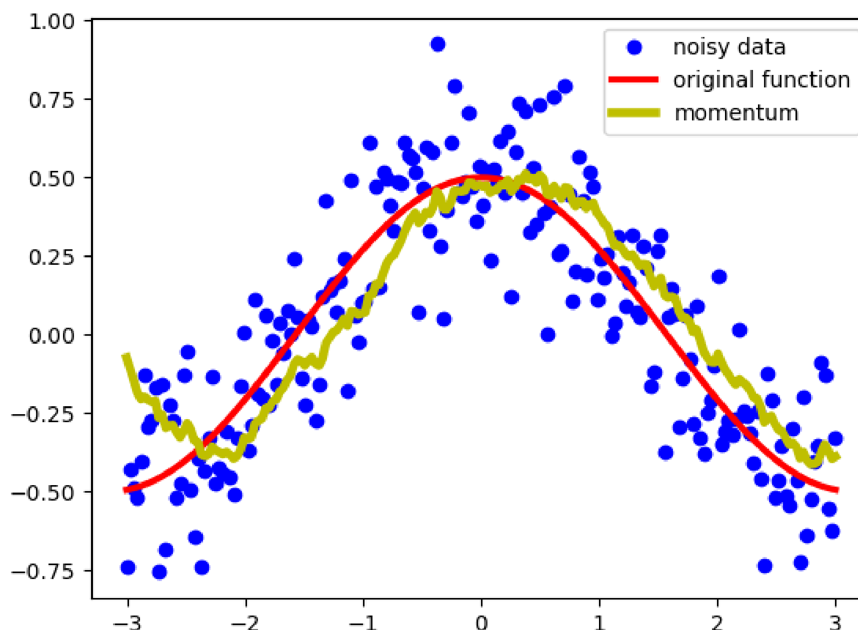


Рисунок 5.3 – імпульс – дані експоненціально зважених середніх

Замість того, щоб мати дані з великою кількістю шумів, отримали набагато більш гладку лінію, яка ближче до початкової функції, ніж дані, які ми мали. Отже, експоненціально зважені середні показники можуть дати нам кращу оцінку, яка ближче до фактичного похідного, ніж наші шумні розрахунки. Це одна з причин, чому імпульс може працювати краще, ніж класичний алгоритм стохастичного градієнта [19].

Але для реалізації було обрано інший алгоритм.

5.2 Багатошаровий перцептрон

Для навчання нейронної мережі було обрано алгоритм нейронної мережі з використанням багатошарового перцептрона.

Багатошарові перцептрони ефективні при вирішенні тих же самих завдань, що і одношарові перцептрони, але мають значно більшу обчислювальної здатністю в порівнянні з одношаровими перцептроном. Завдяки цій своїй здатності вони можуть набагато точніше описувати багатовимірні залежності з великим ступенем не лінійності і високим рівнем перехресного і групового впливу вхідних змінних на вихідні [20].

Приклад двошарового перцептрону з M входами і K виходами був раніше наведено на рис. 5.4. При необхідності використання мереж більш складної структури додаються нові приховані шари або нарощується кількість нейронів в прихованих шарах [20].

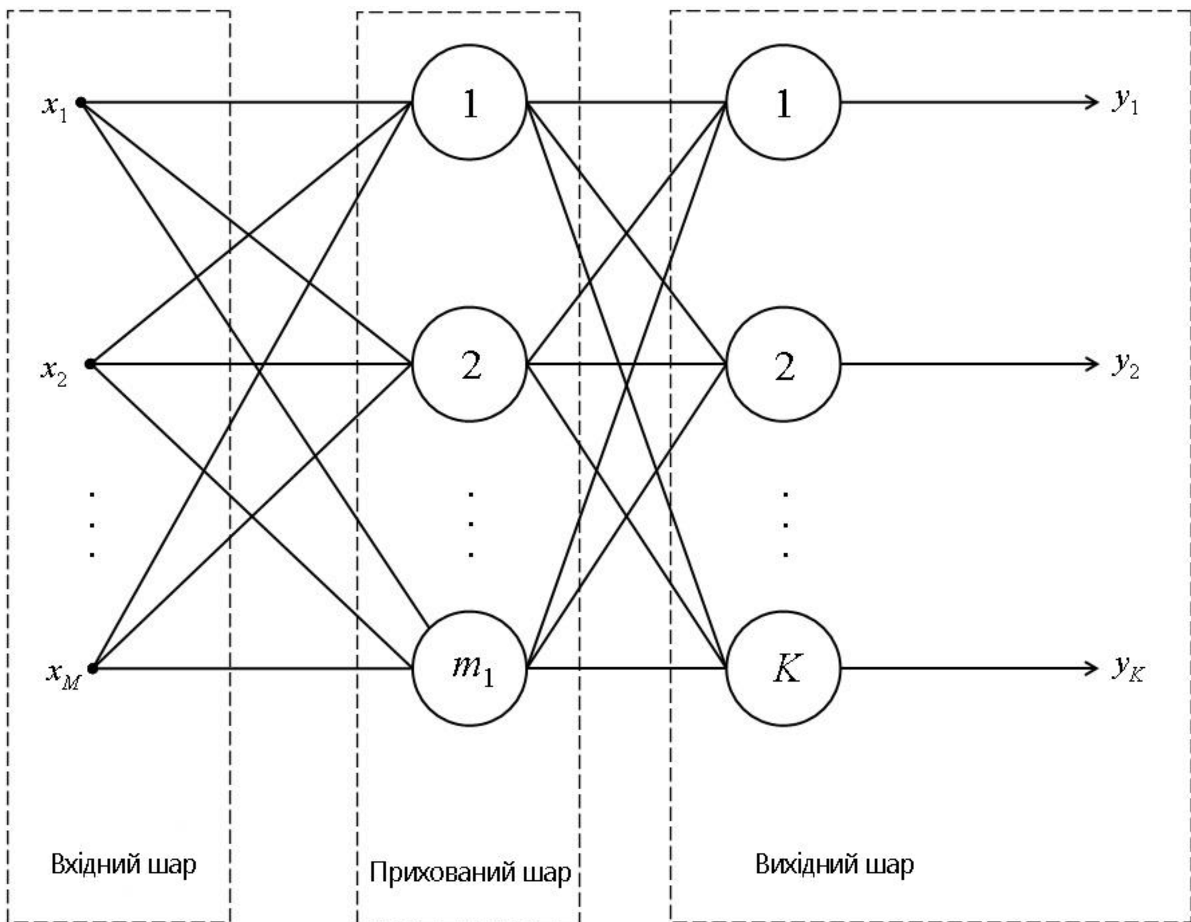


Рисунок 5.4 – Схема двошарової штучної нейронної мережі

Кількість вагових коефіцієнтів, що настроюються в процесі навчання багатошарового перцептрона з L прихованими шарами по ml нейронів в кожному, розраховується наступним чином:

$$N_w = (M+1)m_1 + \sum_{l=2}^{L}(m_{l-1}+1)m_l + (m_L+1)K$$

Для кожного нейрона мережі крім синаптичних зв'язків з елементами вхідного вектора налаштовується зв'язок з фіктивним одиничним входом (коефіцієнт зміщення).

Для більшості завдань, що вирішуються за допомогою багатошарових перцептронів, вибір структури нейронної мережі повинен здійснюватися на основі наступного правила («Правила 2-5»): кількість параметрів в процесі навчання вагових коефіцієнтів має бути в 2-5 разів менше, ніж кількість

прикладів навчальної вибірки . Якщо це співвідношення менше 2, мережа втрачає здатність до узагальнення навчальної інформації, а при досягненні 1 і менше просто запам'ятовує відповіді для кожного навчального прикладу. Якщо ж кількість навчальних прикладів занадто велике для обраної структури мережі, нейромережева модель у багатьох випадках просто усереднює вихідні значення для різних комбінацій вхідних векторів, втрачаючи здатність до коректного відгуку в окремих приватних випадках і підвищуючи величину максимальної вибіркової помилки [20].

Крім того, при виборі структури багатошарового перцептрона слід задавати кількість нейронів в прихованому шарі, що передує вихідному прошарку, не меншим, ніж кількість самих виходів.

Завдання навчання багатошарових перцептронів може бути сформульована як оптимізаційна, в якій цільовою функцією (критерієм оптимізації) є загальна помилка, розрахована за навчальною вибіркою. Відповідно, і вирішувати це завдання можна як будь-яке завдання багатовимірної оптимізації з використанням методів детермінованого або стохастичного пошуку [20].

Найбільш поширений метод навчання багатошарового перцептрона — метод зворотного поширення помилки. Суть даного методу полягає в тому, що сигнал помилки кожного вихідного значення, розрахований на поточному такті навчання, поширюється по шарам в зворотному напрямку (від вихідного до першого) з урахуванням тих же вагових коефіцієнтів, які використовувалися при прямому проходженні вхідних сигналів по нейронної мережі і розрахунку вихідних значень [20].

Алгоритм методу зворотного поширення помилки включає наступні етапи:

1. вагові коефіцієнти багатошарового перцептрона обраної структури ініціалізуються невеликими за абсолютною величиною (не більше M-1 (L + 1) -1) випадковими значеннями, де L - кількість прихованих шарів;

2. на входи нейронної мережі подається вхідний вектор одного з прикладів навчальної вибірки. Проводиться пряме поширення сигналів по мережі з розрахунком значень вихідних змінних;
3. для кожного розрахованого значення вихідної змінної по співвідношенню обчислюється похибка в порівнянні зі значеннями елементів вихідного вектору;
4. за отриманими погрішностей розраховуються нейронів вихідного шару з урахуванням похідних відповідних активаційних функцій:

$$\delta_j = \Delta_j \cdot f'(s_j)$$

5. у зворотній послідовності (від останнього прихованого шару до першого) розраховуються нейронів інших шарів з урахуванням зв'язують шари синаптичних зв'язків:

$$\delta_i = \left(\sum_{j=1}^{m_i} w_{ij} \delta_j \right) \cdot f'(s_i)$$

6. для всіх шарів нейронів, крім першого, перераховуються значення вагових коефіцієнтів по наступних співвідношеннях:

$$w_{0j}^{(q+1)} = w_{0j}^{(q)} + v\delta_j$$

$$w_{ij}^{(q+1)} = w_{ij}^{(q)} + v\tilde{y}_i^p \delta_j$$

7. цикл повторюється з кроку 2 до виконання одного або декількох умов закінчення:
   - вичерпано заданий гранична кількість епох навчання;
   - досягнутий задовільний рівень помилки по всій навчальній вибірці;
   - не відбувається зменшення помилки навчальної вибірки протягом заданого граничного кількості епох навчання;
   - вичерпано заданий граничний фізичний час навчання.

Протягом однієї доби навчання послідовно або у випадковому порядку пред'являються всі приклади навчальної вибірки, причому кожен приклад повинен бути пред'явлений одноразово [21].

Коефіцієнт швидкості навчання задається позитивної константою або змінною величиною ($0 < v <= 1$), що поступово зменшується в процесі навчання нейронної мережі [21].

Якщо порівнювати архітектури багатошарових та одношарових перцептронів, можна виділити кілька її відмінних рис:

- для одного і того ж складу вхідних і вихідних змінних в якості нейромережевої моделі можуть бути використані багатошарові перцептрони різної структури (з різною кількістю прихованих шарів і нейронів у них), тоді як при використанні одношарових перцептронів можливий лише один варіант структури мережі;
- багатошарові перцептрони дозволяють отримати більш коректне математичний опис багато зв’язних функціональних залежностей з яскраво вираженою нелінійністю;
- для навчання багатошарових перцептронів потрібно вибір складнішого алгоритму, ніж для навчання одношарових перцептронів;
- навчання багатошарових перцептронів займає більше часу і вимагає більший обсяг обчислювальних ресурсів комп'ютера [20].

Для оновлення вагів багатошарового перцептрона будемо використовувати стандартний метод «метод зворотного поширення помилки». Для нашого прикладу ми використовували тільки один шар всередині нейронної мережі між входами та виходами, але в нашій програмній реалізації для більшої точності обрахунків використано більше шарів, щоб досягти більшої відмінності в функціональності нейронної мережі [21].

Звичайно, завжди можливо створити одну складну функцію, яка відображатиме композицію по всьому шарові мережі. Наприклад, якщо робити шар 1: $3.x$ для створення прихованого виводу z, а шар 2: z^2 для генерації кінцевого виводу, буде сформована мережа $(3.x)^2 = 9.x^2$. Проте в більшості випадків компонування функцій дуже важке. Плюс для кожного композиції

треба обчислити виділену похідну композиції (яка взагалі не масштабована і дуже схильна до помилок) [21].

У нас є вихідна точка помилок, яка є функцією втрат, і ми знаємо, як її вивести, і якщо ми знаємо, як вивести кожну функцію зі складу, ми можемо поширювати помилку від кінця до початку.

На даному рис. 5.5 показується процес зворотного поширення помилок, що слідують за такими схемами:

— вхід;
— переадресація викликів;
— функція втрат;
— похідна;
— зворотне поширення помилок.

На кожному етапі ми отримуємо дельти на вагах цього етапу.

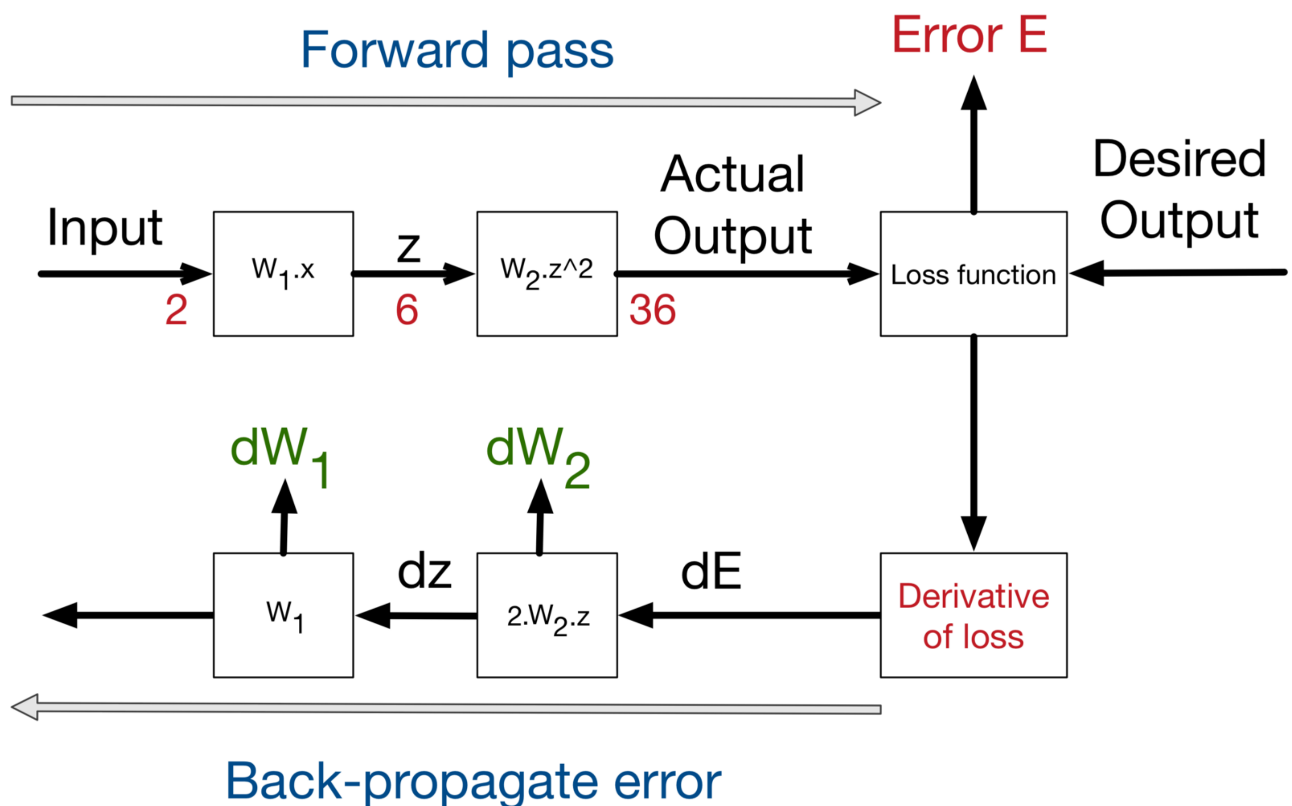


Рисунок 5.5 – Діаграма прямих та зворотних шляхів

Як ми писалось раніше, похідна — це лише швидкість, з якої помилка змінюється відносно змін ваги. У представленому раніше кількісному прикладі цей показник становить 60х. Це означає, що 1 одиниця зміни ваги призводить до 60 одиниць зміни помилки [21].

І оскільки ми знаємо, що помилка в даний час становить 30 одиниць, шляхом екстраполяції курсу, для того, щоб зменшити помилку до 0, нам потрібно зменшити вага на 0,5 одиниці.

Проте для реальних проблем ми не повинні оновлювати ваги з такими великими кроками. Оскільки є багато нелінійностей, будь-які великі зміни ваг призведуть до хаотичної поведінки. Не слід забувати, що похідна є локальною лише у точці, де ми обчислюємо похідну.

Отже, загальним правилом оновлення ваги є правило дельти:

Нова вага = стара вага - похідний курс * рівень навчання

Рівень навчання вводиться як постійна (зазвичай дуже мала), щоб змусити вагу оновлюватися дуже гладко і повільно (щоб уникнути великих кроків та хаотичної поведінки) [21].

По даному рівнянню можна визначити такі спостереження:

— якщо похідна величина позитивна, це означає, що збільшення ваги збільшить помилку, таким чином, нова вага має бути меншою;
— якщо похідна величина негативна, це означає, що збільшення ваги зменшить помилку, тому нам потрібно збільшити ваги;
— якщо похідна є 0, це означає, що ми знаходимося в стабільному мінімумі. Отже, не потрібно оновлювати ваги — ми досягли стабільного стану.

Зараз існує кілька методів оновлення ваги. Ці методи часто називають оптимізаторами. Правило дельти є найпростішим і інтуїтивно зрозумілим [21].

5.3 Реалізація алгоритму багатошарового перцептрона

Після теоретичного опису алгоритмів нейронної мережі, перейдемо то реалізації у програмного коді. Потрібно абстрагуватися від системних понять та переходити до абстракцій – про, шо каже практики розробки SOLID, а саме остання буква – принцип інверсії залежності, що модулі верхніх рівнів не повинні залежати від модулів нижчих рівнів, всі рівні модулів потрібно, щоб залежали від абстракцій. Абстракції не повинні залежати від деталей. Деталі повинні залежати від абстракцій.

Саме по даному принципу було розглянуто та розложено алгоритм мережі. Є кілька важливих об'єктів, про які нам потрібно звернути увагу. Це нейрони, з'єднання, шар та функції. У цьому рішенні окремий клас буде реалізовувати кожен з цих сутностей, перше за все клас слідуючи принципам розробки виділяємо інтерфейси, бо багато спеціалізованих краще, аніж один великий інтерфейс. Потім, об'єднавши все це разом і додаючи алгоритм зворотного відтворення на вершині цього, ми будемо мати нашу реалізацію нейронної мережі.

Зробимо опис основних моделей, які реалізовані за допомогою інтерфейсів.

Клас Neuron — реалізація інтерфейсу INeuron, в якому містяться такі поля:

- List<ISynapse> Inputs – наша мережа складається з великої кількості нейронів, тому у нейрона є список вхідних шарів;
- List<ISynapse> Outputs – вихідний шар для нейронів;
- Guid Id – унікальний ідентифікатор типу Guid для кожного нейрону;
- double PreviousPartialDerivate – обчислена часткова похідна в попередній ітерації навчального процесу мережі;

У табл. 5.1 показано методи, які має нейрон для роботи.

Таблиця 5.1 – Опис методів класу Neuron

Назва методу	Вхідний параметр	Вихідний параметр	Опис
AddInputNeuron	INeuron	–	Метод для додавання вхідних нейронів до списку
AddOutputNeuron	INeuron	–	Метод для додавання вихідних нейронів до списку
CalculateOutput	–	–	Обчислення нейронів за допомогою функції активації
AddInputSynapse	double	–	Додавання синапсису для поєднання нейронів
PushValueOnInput	double	–	Метод для встановлення нових значення на вхідних з'єднаннях.

Кожен нейрон має свій унікальний ідентифікатор – Id. Ця властивість використовується в алгоритмі зворотного розповсюдження. Інша властивість, яка додається для цілей зворотного відтворення, є PreviousPartialDerivate, але це детальніше розглянемо пізніше. Нейрон має два списки, один для вхідних з'єднань – Inputs, а інший для вихідних з'єднань – Outputs. Також має два поля, по одному для кожної з функцій. Вони ініціалізуються через конструктор. Таким чином, можна створити нейрони з різними функціями введення та активації. Функції активації та введення також буде описано нижче в даному розділі.

Цей клас також має деякі цікаві методи. AddInputNeuron та AddOutputNeuron вони використовуються для створення зв'язку між нейронами. Перший додає вхідний зв'язок з деяким нейроном, а другий додає вихідне з'єднання з деяким нейроном. AddInputSynapse додає InputSynapse до нейрона, який є спеціальним типом з'єднання. Це спеціальні з'єднання, які використовуються лише для вхідного шару нейрона, тобто вони використовуються лише для додавання вхідних даних до всієї системи.

Метод CalculateOutput використовується для активації ланцюгової реакції обчислення виходу. Даний метод викликає метод введення, яка запитує значення з усіх вхідних з'єднань. У свою чергу, ці зв'язки будуть запитувати вихідні значення з вхідних нейронів цих з'єднань, тобто вихідних значень нейронів з

попереднього шару. Цей процес буде виконано, доки не буде досягнутий рівень вхідного сигналу, а вхідні значення будуть поширюватися через систему.

Клас Synapse— реалізація інтерфейсу ISynapse, який абстрагує поняття з'єднання нейронів, в даному класі містяться такі поля:

— double Weight – вага з'єднання;

— double PreviousWeight – вага з'єднання на попередніх ітераціях обчислення;

— INeuron _toNeuron – поле нейрону у синапсису, для з'єднання;

— INeuron _fromNeuron – поле нейрону у синапсису, для з'єднання;

У табл. 5.2 показано методи, які має синапсис для роботи.

Таблиця 5.2 – Опис методів класу Neuron

Назва методу	Вхідний параметр	Вихідний параметр	Опис
GetOutput	–	double	Метод для отримання вихідного значення з'єднання
IsFromNeuron	Guid	bool	Метод перевіряє, чи має нейрон певне число у якості вхідного нейрону
UpdateWeight	double, double	–	Метод для оновлення вагів

Кожне з'єднання має свою вагу, представлену через властність з тим самим ім'ям. Додаткова властивість PreviousWeight додана і використовується при зворотному поширенні помилки через систему. Оновлення поточної ваги та збереження попереднього виконується в допоміжній функції UpdateWeight. Є ще одна допоміжна функція – IsFromNeuron, яка визначає, чи певним нейроном є вхідний нейрон до з'єднання. Звичайно, є метод, який отримує вихідне значення з'єднання – GetOutput.

Окрім цієї реалізації зв'язку, є ще одна реалізація. Це InputSynapse і використовується як вхід до системи. Вага цих з'єднань завжди 1, і вона не оновлюється під час тренувального процесу. Дана реалізація коду буде представлена у додатках до магістерської дисертації.

Реалізація нейронного шару досить проста. Весь код також буде міститись у додатках.

Він містить список нейронів, що використовуються в цьому шарі, і метод ConnectLayers, який використовується для склеювання двох шарів.

Функція введення, як згадувалося раніше, найважливіша частина нейрона є функція введення та функція активації. Давайте розглянемо функцію введення. По-перше, було створено інтерфейс для цієї функції, тому його можна легко змінити в реалізації нейрону пізніше.

Дана функція має тільки один метод — CalculateInput, який отримує список з'єднань, які описані в інтерфейсі ISynapse. Метод CalculateInput повинен повернути якесь значення на основі даних, що містяться в списку з'єднань. Потім було здійснено конкретну реалізацію функції вводу — зваженої функції суми.

Використовуючи такий же підхід, як і при реалізації функції введення, спочатку реалізується інтерфейс для функцій активації, його код також буде наведено в додатках.

Після цього можна виконати конкретні реалії. Метод CalculateOutput повинен повернути вихідне значення нейрона на основі вхідного значення, отриманого від функції введення.

5.4 Тестування нейронної мережі

Для покращення написання якості коді та у подальшому його підтримці потрібно було обрати певний спосіб тестування нової функціональності, знаходження різних помилок. Було обрано саме написання юніт тестів для тестування нейронної мережі, тому що вони мають ряд переваг, а саме:

- — розробники пишуть оптимізований та тестований код для покриття його тестами в майбутньому;
- — документація тестів дає змогу швидше розібратись в коді та зрозуміти його основну логіку та роль кожного класу в системі;
- — простіше точно встановити проблему, бо існує велика кількість написаний малих тестів, які тестують певний функціонал;

Основна частина розробки проводилось через такий метод, як TDD, а саме написання коде через написання тестів. З початку було написано всі тести, які

потрібні були для роботи сутностей, але вони не проходили, бо основного функціоналу не було. Після чого проводилась розробка основного функціоналу та проходження всіх написаних тестів. Основним етапом було рефакторинг написаного коду. На рис. 5.6 можемо побачити вікно проходження тестів і всі тести пройдено так як зелено стрічка саме це символізує. Як бачимо з даного рисунку було всього розроблено 25 тестів.

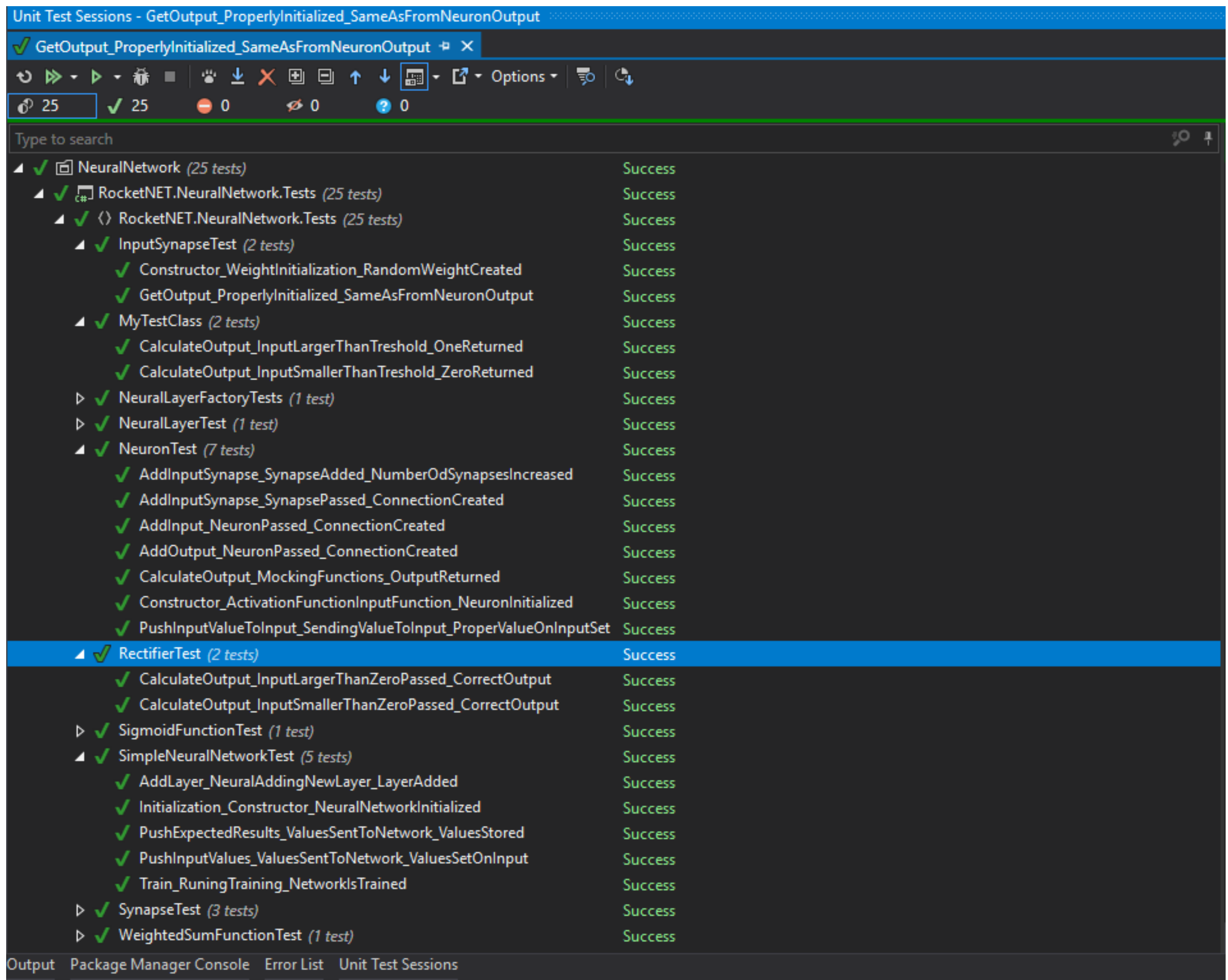


Рисунок 5.6 – Вікно тестів в VS 2017 з плагіном Resharper

Після тестування системи, було проведено процес – визначення проценту покриття коду. Таким чином можна дізнатись, кількість коду, які повністю покриті тестами в процентному відношенні. За допомогою вбудованої системи Code coverage в спеціальний плагін Resharper було виявлено, що 98 відсотків коду нейронної мережі було покрито тестами, а це каже про дуже значний процент покриття коду, майже максимальний. Це можна побачити на рис. 5.7.

Coverage Tree | Hot Spots

Type to search

Symbol	Coverage (%)	Uncovered/Total Stmts.
Total	56%	336/756
Parsing	0%	3/3
Model	0%	18/18
DbInitializers	0%	68/68
Notification	0%	89/89
Logging	0%	148/148
Tests	78%	2/9
NeuralNetwork	98%	8/420
RocketNET.NeuralNetwork	98%	6/244
RocketNET.NeuralNetwork.Synapses	89%	6/55
RocketNET.NeuralNetwork.InputFunctions	100%	0/4
RocketNET.NeuralNetwork.ActivationFunctions	100%	0/17
RocketNET.NeuralNetwork.Neuron	100%	0/36
RocketNET.NeuralNetwork	100%	0/132
RocketNET.NeuralNetwork.Tests	99%	2/176
RocketNET.Core	100%	0/1

Output Coverage

Рисунок 5.7 – Статистика покриття коду

Як показано на рисунку, що не всі частини системи було написано та покрито кодом, бо це дуже кропітка робота та займає великий об’єм часу. У подальшому розвитку додатку — всі частини системи будуть покрити тестами за для того, щоб система працювала стабільно та коректно.

Висновок до розділу

В даному розділі було описано детально алгоритми, які частіше за все використовується розробниками для написання, навчання та налагодження нейронної мережі для роботи. Також був зроблений детальний опис вибраного алгоритму, а саме багатошаровий перцептрон.

Було описано реалізацію мережі у програмному коді та взаємодію між класами даними, алгоритм роботи нейронної мережі.

Також було детально описано всі підходи тестування мережі, які було виконані в даній системі, було наведена кількість тесті, які було написано, а саме 25 та виокремлено недолік системи в тому, шо не повністю всі модулі та сервіси вдалось покрити тестами, бо це великий об’єм часу та ресурсу розробника.

Наступний розділ має на меті зробити опис всіх розроблених систем в додатку та зробити детальний опис коду та пояснення. Як системи взаємодіють між собою та налагодженість роботи.

6 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

6.1 Реалізація роботи з ботами для сповіщення

Для роботи з ботами було обрано використовувати два види провайдерів, які дають доступ по відправці повідомлень slack, telegram. В даному розділі опишемо реалізацію за допомогою telegram бота.

Було додано шаблон проектування – будівник, що дає можливість будувати складний об’єкт та не витрачати багато ресурсів та розробити тестований код. Будівельник дозволяє відокремити процес створення складного об'єкта від його реалізації. При цьому, результатом одних і тих же операцій можуть бути різні об'єкти. Даний шаблон використовується в разі, якщо:

- процес створення об'єкта можна розділити на частини (кроки);
- алгоритм цього процесу повинен бути незалежним від того, з яких частин складається об'єкт;
- конструювання повинно забезпечувати можливість створювати різні об'єкти.

Краще зрозуміти роботу будівельника можна використовуючи таке порівняння: багато породжують шаблони, використовуючи конкретні вихідні дані, видають узагальнений результат (інтерфейс об'єкта). Будівельник ж навпаки, використовуючи узагальнений набір даних, створює відому клієнту конкретну реалізацію.

Шаблон будівельник включає двох учасників процесу:

- будівельник (Builder) – надає методи для складання частин об'єкта, при необхідності перетворює вихідні дані в потрібний вид, створює і видає об'єкт. В нашому випадку дану роль відіграє клас – NotificationBuilderAbstract, що виступає абстрактним класом, який в собі містить методи для побудови повідомлення, а саме BuildTopic та BuildInformation;
- розпорядник (Director) – визначає стратегію збирання: збирає дані і визначає порядок викликів методів будівельника, в даному випадку це клас

Notification – він у будівника визиває метод – GetNotification – для побудови повідомлення та отримання об’єкту.

Після побудови повідомлення в дії вступає бот, який отримує повідомлення, його опрацьовує та відправляє відповідь.

Щоб використовувати telegram бота потрібно мати сертифікат для сервісу, але це можливо зробити за кошти або скористуватися інструментом ngrok. Даний інструмент робить захищений тунель до localhost або певного серверу – клієнтської частини.

Nngrok створює тунель із загальнодоступного інтернету до порту на локальній машині. Ви можете надати цю URL-адресу будь-кому, щоб дозволити їм випробувати веб-сайт, який ви розробляєте, без будь-якого розгортання. За замовчуванням ngrok використовуватиме ngrok.com як сторонній ретранслятор. Ця послуга надається безкоштовно, саме тому була обрана. Також можна налаштувати та використовувати власний сервер.

Для того що сервер почав працювати потрібно виконати команду – ngrok http 54146 -host-header="localhost:54146". Тоді стартує сервер та починає працювати за певною адресою.

```
ngrok by @inconshreveable

Session Status                online
Account                       Romashchenko Pavel (Plan: Free)
Version                       2.2.8
Region                        United States (us)
Web Interface                 http://127.0.0.1:4040
Forwarding                    http://519e664f.ngrok.io -> localhost:54146
Forwarding                    https://519e664f.ngrok.io -> localhost:54146

Connections                   ttl     opn     rt1     rt5     p50     p90
                              0       0       0.00    0.00    0.00    0.00
```

Рисунок 6.1 – Старт роботи серверу ngrok

Після чого потрібно додати до бота строку серверу де він працює. Даний код робить це – client.SetWebhookAsync("https://519e664f.ngrok.io/bot").Wait();.

Далі потрібно запустити додаток і він буде працювати.

ngrok online Inspect Status Documentation

All Requests Clear

POST /bot	200 OK	40.95ms
GET /	404 Not Found	13.97ms

a few seconds ago Duration 40.95ms IP 149.154.167.209

POST /bot

Summary Headers Raw Binary Replay

Headers

Accept-Encoding	gzip, deflate
Content-Length	329
Content-Type	application/json
User-Agent	Go-http-client/1.1
X-Forwarded-For	149.154.167.209
X-Forwarded-Proto	https
X-Original-Host	519e664f.ngrok.io

200 OK

Summary Headers Raw Binary

Рисунок 6.2 – Серверна сторінка ngrok

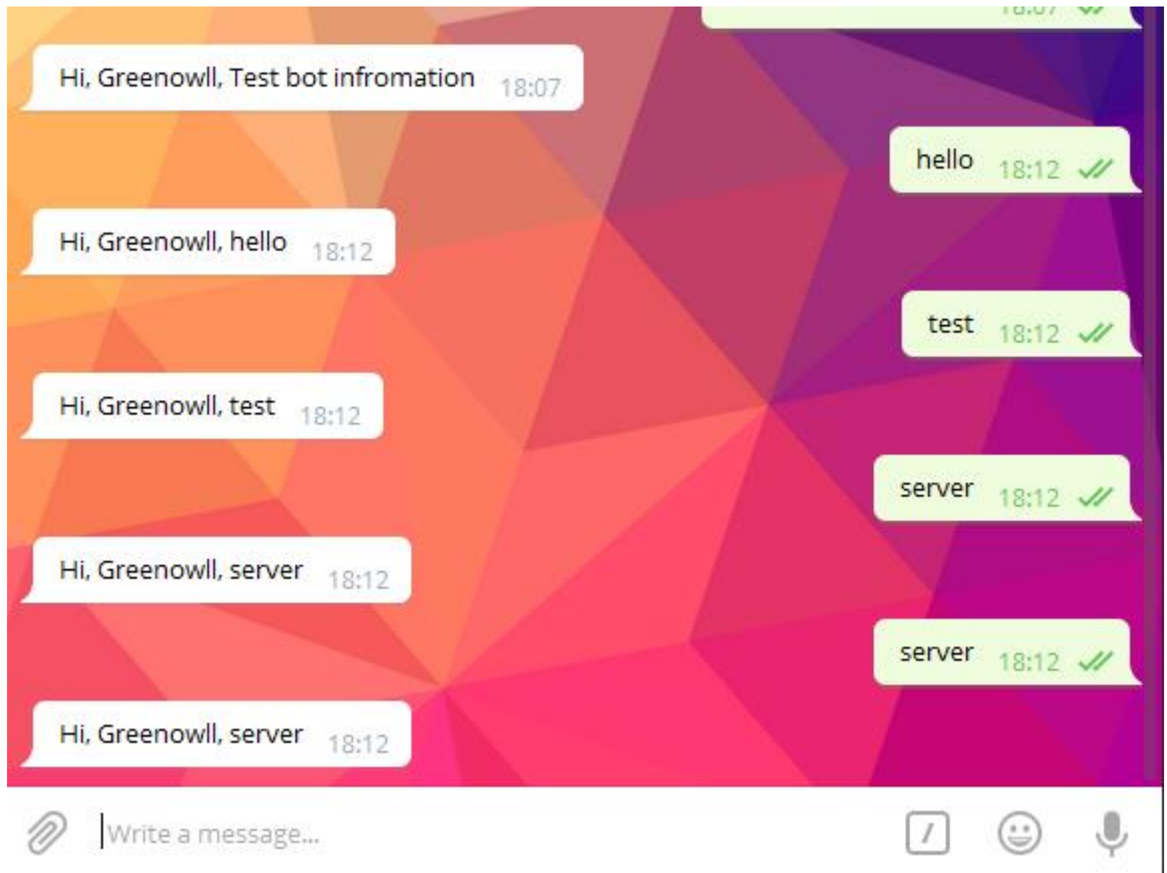


Рисунок 6.3 – Повідомлення з чату telegram

6.2 Реалізація логування

Для логування подій в системі не було обрано певний інструмент, це зроблено для того, що реалізувати власну систему запису логів до будь якого сховища зберігання та зробити його універсальним з можливістю використання певних плагінів або додатків. Також дана можливість дає незалежність проекту від сторонніх бібліотек, що дає можливість в подальшому не мати проблем при збірці проекту та розгортування на серверах.

Було реалізовано два види інтерфейсу логування, а саме синхронний та асинхронний. Перед початком опису, клас має велику кількість перечислень, зробимо їх опис.

LogLevel – за допомогою даної властивості вирізняється рівень повідомлень в системі, існують такі рівні:

- trace – даний параметр дає повну інформацію про все, що відбувається в коді;
- debug – інформація при налаштуванні, відкладці програми;
- info –спеціальний параметр, для отримання інформації з коду;
- warn – параметр попередження про виникнення певних ситуацій, на які потрібно зробити наголо та в подальшому їх виправити;
- error – параметр попередження про виникнення помилкових ситуацій, які потрібно негайно виправляти, бо може статися помилка найвищого рівня, після якої система буде просто перезавантажена;
- fatal – параметр попередження про виникнення фатальної помилки або ситуації;

LogParameter – за допомогою даної властивості вирізняється параметр по якому проводиться логування – дана властивість створення, для того, щоб при записі інформації роботи нейронної мережі, було розуміння по якому параметру проводиться запис. Існують такі параметри:

- Default – звичайний тип параметру запису, записується тільки інформація з коду розробника;

— Memory – параметр запису пам'яті на сервері в даний час логування;
— Time – параметр часу логування;
— CPU – параметр запису процент завантаженості процесорів в сервері;

LogType – за допомогою даної властивості розуміється, з яким ресурсом працює система запису інформації, це може бути файл або база даних. Існують такі параметри:

— Console – записування відбувається до консолі;
— DataBase – записування відбувається до бази даних;
— File – записування відбувається до файлу;
— Json – записування відбувається до файлу формату json;
— Combine – відбувається комбіноване записування до різних файлів або також до бази даних.

Опишемо інтерфейс ILog, він має такі поля:

— LogLevel LogLevel – рівень запису в системі;
— LogType LogType – тип повідомлення;
— LogParameter LogParameter – який параметр запису буде відбуватися;
— LogNotification LogNotification – параметр запису в якій системі.

Зробимо опис важливих методів в табл. 6.1 для класу Log. В даному класі також використаний один із принципів гарного коду. Це принцип перевантаження методів. За таким принципом пишеться один метод з однією кількість параметрів, а потім даний метод перевантажується іншими за різною кількістю параметрів. Таких підхід дає можливість написати реалізація коду тільки для одного методу, який містить саме більше кількість параметрів, а потім його використовувати в інших методах.

Таблиця 6.1 – Опис методів класу Log

<table>
<tr><th>Назва методу</th><th>Вхідний параметр</th><th>Вихідний параметр</th><th>Опис</th></tr>
<tr><td>Log</td><td>message</td><td>bool</td><td rowspan="5">Метод для запису подій в системі</td></tr>
<tr><td>Log</td><td>message, exception</td><td>bool</td></tr>
<tr><td>Log</td><td>message, exception, logLevel</td><td>bool</td></tr>
<tr><td>Log</td><td>message, exception, logLevel, logNotification</td><td>bool</td></tr>
<tr><td>Log</td><td>message, exception, logLevel, logNotification, logParameter</td><td>bool</td></tr>
</table>

Висновки до розділу

В даному розділі було описано реалізація коду відправки повідомлення за допомогою бота та описано підхід зі сервером, який був використаний. Також було описана взаємодія з класами та описано детально реалізацію логування завдяки відбувається запис в системі про різні події та помилки будь якого рівня, на які потрібно зважати.

В наступному розділі буде описано систему з точки зору стартап проекту та в цілях і можливостях реалізації з економічної точки зори, проблеми, які можуть стати на заваді цьому та фірми конкуренти, які будуть боротися з нашою системою називатися кращою. Було описано зміст ідей проекту та цінність його.

Проведено технологічний аудит проекту для того, щоб визначити, які були задіяні технології для його реалізації, можливі терміни та ресурси, які потрібно для розробки. Також було проаналізовано ринок та можливі перспективи магістерської дисертації, які цілого стартап проекту.

7 СТАРТАП ПРОЕКТ

7.1 Опис ідеї проекту

Універсальна система навчання та аналізу навантаження планувальника завдань має великий потенціал у компаніях різного масштабу та роботи. Дана система виконує перед собою ряд поставлених задач, які допомагають різним компаніям відчувати стабільність в інформаційному полі. Дана система розроблена таким чином, щоб користувачі могли легко нею керувати та вносити зміни або монітори стан. У табл. 7.1 представлено напрямки застосування, ідеї та вигідні позиції для користувачів або компаній.

Таблиця 7.1 – Зміст ідей стартап проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Можливість обрання свого алгоритму розподілення навантаження серед реалізованих	Системи управління трафіком та роботи в кластерах серверів для розподілення навантаження	Можливість довершувати та вдосконалювати процес розподілення навантаження на сервер
Можливість налаштування графічного інтерфейсу під свої бажання	Системи відображення графічного інтерфейсу користувача	Можливість працювати з зручним графічним інтерфейсом налагодженим під кожного користувача
Надання можливості користувачеві бачити інформації про стан системи в зручному вигляді	Системи відображення стану цілої системи та сервісів окремо	Можливість працювати з зручним графічним інтерфейсом та отримувати інформацію про стан системи
Застосування алгоритму навчання нейронної мережі	Система навчання та аналізу для подальшої роботи	Можливість використовувати та налаштовувати алгоритм нейронної мережі
Надання можливості користувачу отримувати сповіщення через зручний канал повідомлень	Системи відображення стану цілої системи та сервісів окремо через зручні канали повідомлень	Можливість зручно та швидко отримувати повідомлення та стан системи на використовувані системи повідомлень користувача

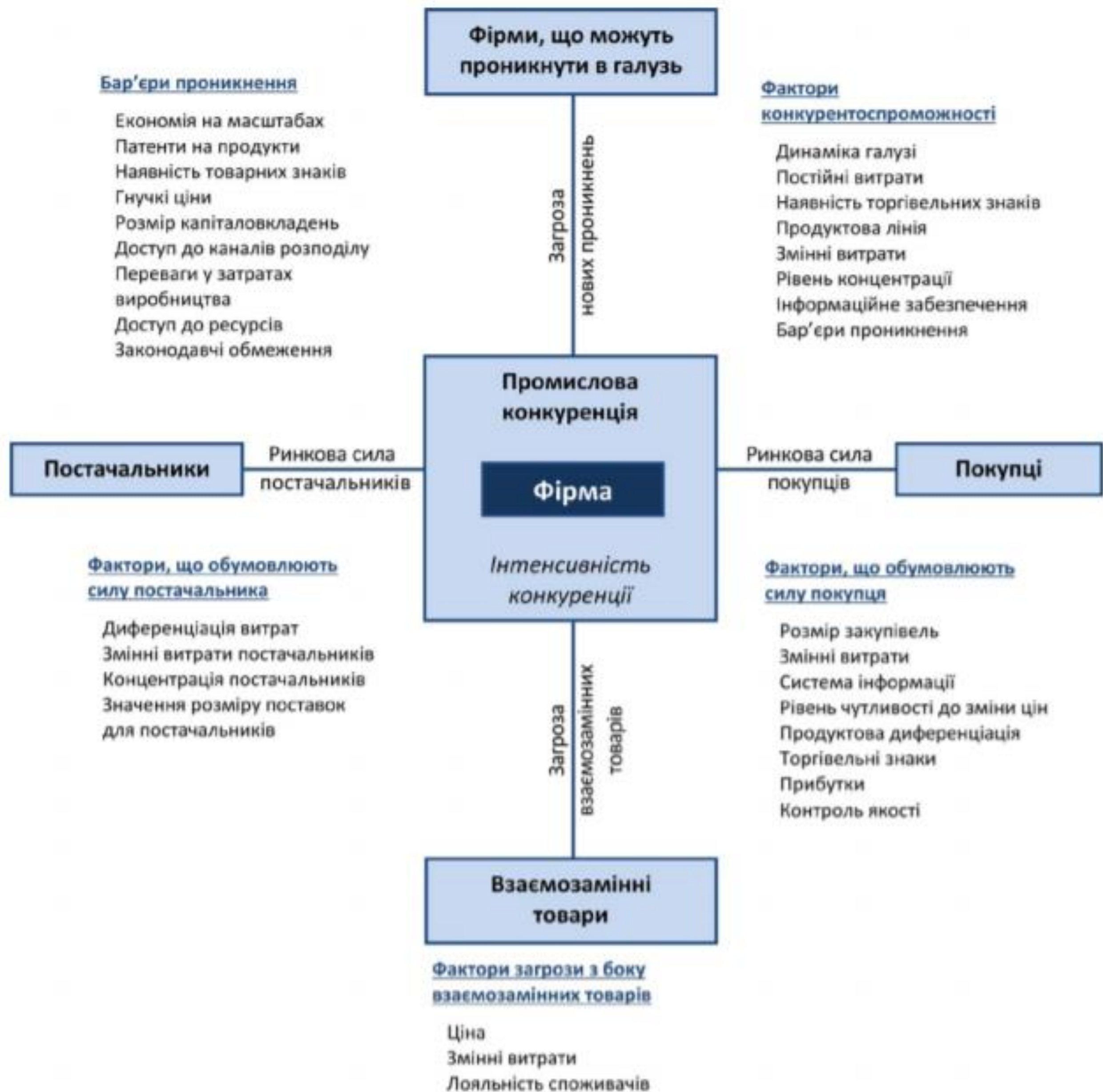


Рисунок 7.1 – Модель 5 сил М. Портера

За моделлю п'яти сил Портера — зробимо аналіз своєї продукції використовуючи основні пункти, які зазначені на рис. 7.1.

З техніко-економічної точки зору продукт та сервіс в цілому має великий ряд переваг. З точки зору бар'єрів потрапляння на ринок, основними факторами продукту можна вважати те, що він є відкритим продуктом та не потребує покупки ліцензії з боку користувача системи. Також використовуючи продукт покупець не витрачати власні кошти на отримання продукту, а з огляду на технічні властивості продукту, провести велику оптимізацію обчислювальних

ресурсів в компанії. Також, продукція має перевагу у затратах на виробництво, а це мінімальні затрати та не потребує великих вкладень в продукт для розвитку. З боку доступу до ресурсів — не має обмежень, бо використання всесвітньої мережі доступно всім, та не має обмежень на законодавчій області.

Галузь, в якій буде встановлюватись продукт є дуже динамічна та розвивається з кожним днем. Тому зазначимо також ще декілька характеристик продукту, а саме:

- розвиток та розширення лінії продукції – даної перевага зумовлена тим, що продукт новий та розширюється та розростається кожен день і тому лінія продукції буде розширюватись та доповнюватися – тим самим приваблювати нових користувачів та клієнтів;
- розробка та наявність торгівельного знаку для продукції – торгівельний знак продукції на ринку буде виокремлювати даний сервіс серед інших, а також буде гарно впізнаний в даному сегменті на ринку та буде приваблювати клієнтів для співпраці з даним продуктом;
- відсутність постійних та змінних витрат – дана перевага має в собі інвестиційний зміст в тому, що не потрібно робити великі вклади в даний продукт, його потрібно просувати на ринку та розширювати продукцію. Можливо для того, щоб сервіс та система в цілому стала популярною потрібно зробити та втілити попередні два пункти, а саме створити більше лінійку продукції для захоплення більшої аудиторії клієнтів, а також створення торгівельного знаку для пізнаваності та визнання, а після цього створення агресивної маркетингової політики, для того щоб продукт мав рекламу та розвиток;
- унікальність продукту на ринку – в даний час на ринку не має такого ж продукту за функціональністю та можливістю гнучкого налаштування продукту використовуючи різні налаштування. Також на сьогодні не має такої зручної системи інтеграції з системами сповіщення, а також зручного для користувача інтерфейсу;

- орієнтованість на клієнта – головною перевагою системи є орієнтованість на клієнта в плані оптимізації та розширення функціональності. Це є основною та ключовою ланку при розробці продукції, бо таким способом клієнти будуть користуватись даною системою.

Зважаючи на всі ці фактори – та беручи до уваги, що на даний момент на ринку не має аналогічних систем, а тільки спроби створити, щось аналогічне та в цілях навчальної практики, то даний продукт повинен гарно себе зарекомендувати на ринку. Бар'єр проникнення інших систем на ринок, це тільки способом зацікавлення користувачів, але також ця ланка вже обдумана та є способи для заохочення клієнтів обирати наш продукт.

З огляду на конкурентні продукти, запропонований продукт має ряд переваг у важливих техніко-економічних характеристиках та не має негативних характеристик.

7.2 Технологічний аудит ідеї проекту

Проведемо аудит технології, за допомогою якої можна реалізувати ідею продукту. Для цього визначимо технології створення продукту, існування таких технологій, а також стадію їх розробки і доступність. Ці показники зазначено в табл. 7.2. Даний аудит потрібен для того, щоб дати розуміння, які можливості має проект та технології реалізації, для розуміння, яким чином та які інструменти потрібні для того, щоб здійснити проект в реальність, так які ресурси і технології затратити.

Таблиця 7.2 – Технологічна здійсненність ідеї проекту

Ідея проекту	Технології реалізації	Наявність технології	Доступність технології
Можливість обрання свого алгоритму розподілення навантаження серед реалізованих	На даний момент існує велика кількість алгоритмів розподілення навантаження на сервер	Наявна технологія	Алгоритми наявні у відкритому доступі, але реалізацію потрібно розроблювати самостійно
Можливість налаштування графічного інтерфейсу під свої бажання	Стек для розробки веб проектів – html, css, js	Інструменти та технології розробки наявні	Технологія доступна у відкритому вигляді
Надання можливості користувачеві бачити інформації про стан системи в зручному вигляді	Стек для розробки веб проектів – html, css, js	Інструменти та технології розробки наявні	Технологія доступна у відкритому вигляді
Застосування алгоритму навчання нейронної мережі	Реалізація технології з використанням алгоритму багатошарового перцептрона	Наявність алгоритмів але технології ні	Доступні алгоритми для навчання та матеріали та приклади реалізації інших алгоритмів нейронних мереж
Надання можливості користувачу отримувати сповіщення через зручний канал повідомлень	Реалізація технології завдяки відкритим бібліотекам та прикладам коду для роботи	Наявність відкритого доступу до API та бібліотеки для використання	Технології доступні

При розробці продукту було обрано вже наявні інструменти та технології. Також було обрано реалізацію тих модулів продукту, які наявні тільки алгоритми розробки, а також приклад, тільки іншим систем. Таким чином продукт в своїй мірі являється унікальним в розробці алгоритму нейронної мережі з використанням багатошарового перцептрона.

Найкращий технологічний шлях розробки проекту — це максимально наявних технологій для швидкості та реалізації, а також вибір алгоритму

реалізації, який буде виділяти систему серед всіх інших. Таким чином продукт гарантує попит та успіх на ринку серед користувачів.

7.3 Аналіз ринкових можливостей запуску стартап-проекту

Зробимо визначення ринкових можливостей, які можна використати під час ринкового впровадження даної системи в роботу, та ринкові загрози, які можуть перешкодити реалізації та розробки системи в цілому або окремих модулей проекту. Зазначений аналіз дозволяє спланувати напрями розвитку проекту із урахуванням стану ринкового середовища, потреб потенційних клієнтів та великих компаній, а також попит та пропозицій. Результати аналізу попиту на продукт, що розробляється, зазначено в табл. 7.3.

Таблиця 7.3 – Аналіз попиту на продукт

Показники стану ринку	Характеристика
Кількість головних гравців, од	2-4
Загальний обсяг продаж грн/ум.од	30-50 тис./місяц
Динаміка ринку	Зростає
Наявність обмежень для входу	Приваблювання клієнтів для роботи з проектом та системою
Специфічні умови для стандартизації та сертифікації	Відсутні
Середня норма рентабельності в галузі	70-80%

Середня норма рентабельності в галузі значно вища за банківські відсотки вкладення, бо при старті проекту великих вкладень не потрібно робити, тільки в рекламу або інші можливості популяризації проекту, отже проект є привабливим для інвестицій за попереднім оцінюванням. В наступній таблиці визначимо потенційні групи клієнтів, їх характеристики та сформуємо орієнтовний перелік вимог до товари для кожної (табл. 7.4).

Таблиця 7.4 – Потенційні групи клієнтів

Потреба, що формує ринок	Цільова аудиторія	Відмінності у поведінці різних цільових груп	Вимоги споживачів до товару
Потреба отримання статус системи через графічний інтерфейс	Користувачі даної системи; спеціалісти, які налаштовують системи для роботи	Кожна з груп має свої зауваження до використання системи, користувачі прагнуть зручності, спеціалісти функціональності	Зручність використання інтерфейсу та можливість налаштування під власні потреби
Потреба отримувати сповіщення через повідомлення	Користувачі даної системи; спеціалісти, які налаштовують системи для роботи або менеджери	Кожна з груп має свої зауваження до використання системи, користувачі прагнуть зручності отриманні повідомлень, спеціалісти функціональності	Зручність використання повідомлення та можливість налаштування під власні потреби
Потреба оптимізації роботи серверів	спеціалісти, які налаштовують та впроваджують дану систему для роботи або менеджери	Кожна з груп має свої сценарії для використання даної системи та вимоги до оптимізації роботи та коректності	Зручність конфігурації алгоритму для оптимізації роботи сервера
Потреба використання різних алгоритмів для вирішення проблеми оптимізації роботи серверів	спеціалісти, які налаштовують та впроваджують дану систему для роботи або розробники, які дописують функціональність до системи	Кожна з груп має свої сценарії для використання даної системи та вимоги до оптимізації роботи та коректності	Зручність вибору та налаштування алгоритму під власні потреби

Зробимо аналіз ринкового середовища: визначмо групи факторів, що сприяють ринковому впровадженню проекту, та факторів, що йому перешкоджають (табл. 7.5, табл. 7.6).

Таблиця 7.5 – Фактори загроз

Фактор	Зміст загрози	Можлива реакція компанії
Зміни у податках	Зміни сплати податків, що значно впливають на фінансову модель	Застосування іншої фінансової моделі, технології реклами для продукту
Збій роботи апаратного забезпечення	Непередбачені помилки в роботі апаратного забезпечення, що впливають на збій виконання системи в цілому та перевантаження	Закупівля якісного обладнання для розробки та схема роботи при такій проблемі
Збій роботи алгоритмічного забезпечення	Непередбачені помилки та нюанси в роботі алгоритмічного забезпечення	Покращення алгоритмів роботи
Піратство	Злами ПЗ, що впливають на розповсюдження продукту та фінансову модель	Покращення ліцензування та захищеності системи

Таблиця 7.6 – Фактори можливостей

Фактор	Зміст можливості	Можлива реакція компанії
Розвиток ринку open-source ПЗ	Розвиток ринку open-source ПЗ та поява нових бібліотек, які можна застосувати для покращення розробок	Застосування open-source бібліотек у продуктів
Розвиток нейронних мереж та створення нових алгоритмів	Розвиток нейронних мереж дасть можливість створювати новій алгоритми, які будуть краще опрацьовувати вхідні данні та краще проводити аналіз та швидше навчатись	Реалізувати алгоритми в системі, які більш оптимізовані да мають більше можливостей для використання
Розвиток додатків отримання повідомлень	Розвиток нових сервісів для отримання повідомлень, які будуть зручно інтегруватись в системи та швидше працювати та зручніше для користувачів	Створення команди розробників для додавання нових сервісів передачі повідомлень для зручності користувачам
Розвиток графічних інтерфейсів	Розвиток нових сервісів для графічних інтерфейсів, які будуть зручніше для користувачів	Створення команди розробників для додавання нових можливостей в графічні інтерфейси або переробка інтерфейсу на новий лад для зручності та швидкості роботи та відклику

Використовуючи вище зазначені фактори ризику та можливостей можливо провести детальний аналіз пропозиції та визначити, описати загальні риси конкуренції на ринку та описати можливість дії компанії при таких ситуаціях. Дане порівняння в табл. 7.7.

Таблиця 7.7 – Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
Олігополія	У кількості постачальників ПЗ до користувачів та якості додатку	Велика конкуренція, розробка рішення, що має новизну і потрібно приймати рішення для того, щоб бути конкурентоспроможною
Цінова конкурентна перевага	Переваги в галузі досягаються не лише за рахунок кращого результату використання продуктів, але і за рахунок ціни	Продукт розробляється як відкритий, що надає максимальну цінову перевагу при виборі рішення клієнтами
Національний рівень конкурентної боротьби	Розробка продукту не прив'язана до конкретного регіону або конкретних людей	Особливість середовища не впливає на діяльність підприємства та розробки додатку в цілому та систем окремо
Товарно-видова конкуренція	Конкуренція спостерігається між різними видами товарів (продуктів)	Необхідно розробити продукт, що якісно відрізняється від інших видів, або ж схожих видів
Внутрішньогалузева конкуренція	Конкуренція між продуктами спостерігається всередині галузі	Розробити сервіс, що має високі конкурентоспроможні показники в галузі

На основі аналізу конкуренції, а також із урахуванням характеристик ідеї проекту, вимог споживачів до товару та факторів маркетингового середовища було визначено та обґрунтовано перелік факторів конкурентоспроможності в табл. 7.8.

Таблиця 7.8 – Обґрунтування факторів конкурентоспроможності

Фактор	Обґрунтування
Ціна	Продукт позиціонується як відкритий, що надає переваги перед конкурентами з точки зору цінового фактору вибору та не потребує його купівлі, що є привабливим для клієнтів та користувачів системи
Зручність управління та зміна параметрів в системі	Технічна перевага продукту так як конкуренти мають складні механізми, але зручний інтерфейс та можливість гнучка змінювати параметри в системі вирішують дане питання
Зручність отримання інформації про стан системи	Технічна перевага продукту в тому, що система налаштована сповіщати системних адміністраторів або персонал про різні стани система або інформаційні повідомлення про зручний канал зв'язку
Зручність роботи в графічному інтерфейсі	Перевага продукту для клієнтів enterprise сегменту, оскільки продукти конкурентів не адаптовані для enterprise ринку взагалі

Всі перечисленні фактори дозволяють продукту та сервісу в цілому бути конкурентоспроможним на ринку та заявляти про себе на перших місцях рейтингу, але кожен з описаних факторів вище – має різний вплив на успіх ринку – тому в табл. 7.9 було зроблено порівняння систем, які схожі до розроблюваної за певними параметрами та характеристиками. Скорочені назви конкурентів:

— Job Scheduling Cloud Computing Using Neural Networks – J;
— Analysis of Job Scheduling Algorithms in Cloud Computing – A;
— Radboud Universiteit Nijmegen – R.

Таблиця 7.9 – Порівняльний аналіз сильних та слабких сторін «Системи навчання та нализу навантаження планувальника завдань»

Фактор	Балл	-3	-2	-1	0	1	2	3
Ціна	20			R	J, A			
Зручність керування	18		R		J	A		
Зручність налаштування	20		A	R		J		
Інформаційне забезпечення	15	J, A, R						
Сповіщення користувачів	20	J, A, R						
Можливість кастомізації	10	J	R		A			

Таблиця 7.10 – SWOT-аналіз стартап проекту

Сильні сторони: Ціна, Зручність керування, зручність налагодження, інформаційне забезпечення, зручність інформування через повідомлення, зручність контролю за станом системи та кожного сервісу окремо	**Слабкі сторони:** Можливість індивідуалізація системи
Можливості: Залучення користувачів до створення бази знань, що значно покращить інформаційне забезпечення. Створення окремих рішень з налаштування системи, що зробить її більш гнучкою. Створення команд розробників для розроблення нових сервісів системи, для додавання нових алгоритмів нейронних мереж або розподілення навантаження на сервери.	**Загрози:** Поява широкої потреби в індивідуалізація рішення. Поява інших конкурентів, які зможуть велику кількість користувачів переманити на іншу систему завдяки більшої функціональності системи або різними додатковими кращими якостями.

На табл. 7.10 можемо побачити SWOT аналіз продукту, що розробляється та надає змогу розглянути сильні і слабкі сторони сервісу в цілому, а також можливості та загрози, що існують в процесі розробки стартап проекту.

7.4 Розроблення ринкової стратегії проекту

Основою ринкової стратегії проекту є опис цільових груп потенційних користувачів продукту, який зазначено в табл. 7.11.

Таблиця 7.11 – Цільові групи користувачів

Опис профілю цільової групи	Готовність сприйняти продукт	Орієнтовний попит	Інтенсивність конкуренції	Простота входу
маленькі компанії	Висока	Висока	Середня	Низька
середні компанії	Висока	Висока	Середня	Низька
великі компанії	Висока	Висока	Середня	Низька
корпорації	Висока	Високий	Середня	Низька

Як бачимо з табл. 7.11 продукт розроблено універсально та таким чином, щоб компанія будь якого рівня та фінансовими можливостями могла дозволити використовувати нашу продукцію. Таким чином велика кількість ринку буде захоплена нашим продуктом, що дасть великі досягнення та прибутки в подальшому при розвитку. Розвиток не стоїть на одному сегменті — продукт намагається всіма силами завоювати всі сегменти для простішого розвитку у майбутньому та забезпечити себе від інших конкурентів. Базова стратегія конкурентної поведінки включає такі положення:

- система представляє себе, як унікальним продукт та позиціонує на ринку, як перший;
- проект буде шукати та приваблювати інших клієнтів, також буде використано агресивний маркетинг для того, щоб завоювати більше кількість користувачів та ринок;
- в якості стратегії буде обрана стратегія зайняття конкурентної ніші та розвиток.

7.5 Розроблення маркетингової програми стартап-проекту

Зробимо огляд основних позицій та різні вигоди, що пропонує стартап проект для вирішення потреб користувачів та переваги рішення перед конкурентами в табл. 7.12.

Таблиця 7.12 – Ключові переваги проекту

Потреба	Вигода	Переваги над конкурентами
Можливість обрання свого алгоритму розподілення навантаження серед реалізованих	Вигода в системі управління трафіком та роботи в кластерах серверів для розподілення навантаження	Можливість довершувати та вдосконалювати процес розподілення навантаження на сервер
Можливість налаштування графічного інтерфейсу під свої бажання	Вигода в системі відображення графічного інтерфейсу користувача	Можливість працювати з зручним графічним інтерфейсом налагодженим під кожного користувача
Застосування алгоритму навчання нейронної мережі	Вигода в системі навчання та аналізу для подальшої роботи	Можливість використовувати та налаштовувати алгоритм нейронної мережі

Збут продукту виконується завдяки відділу продажів та збуту. Завдяки проведення агресивної політики маркетингу, було залучено велика кількість компаній та користувачів. Також, як раніше описувалось проект в цілому безкоштовний та цін на нього немає. Концепт комунікація проходить через основних менеджерів в компанії. За кожен регіон закріплений свій менеджер, перед яким стоїть основна задача — це донести та розповісти користувачу про систему та про роботу її в цілому використовуючи демонстраційні матеріали та презентації.

Висновки до розділу

В результаті проведення аналізу можливості створення комерціалізації та подальшого розвитку стартап проекту, розроблено стратегії конкурентоспроможності проекту, проведено технологічний аудит стартап

проекту, розроблено основні ринкові стратегії запуску проекту, а також проведено SWOT аналіз.

З огляду на потенційні групи клієнтів – основним моментом залишається провести агресивну маркетингову політику, яка дасть завоювати велику кількість користувачів та ринок компаній різного рівня. Проект має перспективи у різних галузях, бо в кожній галузі використовуються сервери для обміну даними.

Також система має недолік, а саме індивідуалізація системи. Проблема полягає в тому, що під кожного замовника потрібно налаштовувати певні модулі або сервіси окремо, що обмежую систему в роботі. Дана проблема вирішується набором та навчання спеціалістів високого рівня для розроблення системи зручної індивідуалізації під кожного замовника та налаштування специфічних параметрів.

ВИСНОВКИ

У даному дослідженні проведено огляд та аналіз існуючих систем по навчанню та аналізу навантаження планувальника завдань.

1. Сформульовано недоліки з порівняльних систем та виокремлені основні пункти для покращення розроблюваної системи. Виокремлено основні задачі, які були поставлені перед магістерською дисертацією та всі вони були успішно виконані.
2. Виконано порівняльний аналіз існуючих систем, а саме декількох авторів наукових статей, котрі прагнули створити схожі системи для роботи та оптимізації виконання задач хмарних серверів. Дані статті були детально вивчені та проаналізовані для того, щоб зрозуміти специфіку даних систем та виокремити сильні та слабкі сторони, що дають змогу вдосконалити їх у магістерській роботі.
3. Проведено порівняння різних архітектур проектування та способи розробки великих мікросервісних додатків. Було обрано три види архітектури, а саме: монолітна, сервісно-орієнтована та мікросервісна та описано сильні та слабкі сторони кожної з них. Прийнято рішення використовувати мікросервісну архітектуру, бо вона має ряд переваг.
4. Зроблено аналіз порівняння методів балансування навантаження на сервер та обраний алгоритм – Weighted Round Robin, дуже гнучкий для використання та може бути видозмінений. Даний алгоритм відрізняється від інших легкістю реалізації та можливістю бути вдосконаленим завдяки великої кількості параметрів, що оброблюються.
5. Проведено аналіз даних, які можна отримати при роботі алгоритму балансування навантаження при роботі на сервері разом з нейронною мережею в парі. На основі отриманих даних було прийнято рішення реалізації способу, який поєднав роботу алгоритмів в цілому для отримання результатів в магістерській дисертації.

Розроблений алгоритм нейронної мережі – багатошаровий перцептрон, дав можливість системі аналізувати та опрацьовувати дані без втручання інших користувачів в систему. Даний алгоритм був обраний через легкість реалізації та інтеграції з методом балансування навантаження задач. Також було додано алгоритм комівояжера, який вирішував задачу найшвидшого пошуку серверу, якому потрібно дати задачу на виконання. Робота всіх алгоритмів здійснюється за таким принципом, що після запуску роботи системи, нейронна мережа збирає та аналізує дані з мережі та з роботи алгоритму балансування навантаження. Таким чином проходить навчання перцпетрону, яке триває досить великий проміжок часу. Після навчання мережа використовуючи пошук найкоротшого серверу, параметри серверів та мережі самостійно розподіляє навантаження планувальника завдань. Також було реалізована система сповіщення користувачів про стан системи.

Створена система має ряд переваг над іншими системами:

— легкість у використанні різних алгоритмів балансування навантаження;
— отримання сповіщення про роботу системи;
— універсальний та вдалий вибір алгоритму навчання та аналізу параметрів;
— універсальність роботи не залежно від ОС.

Для тестування роботи системи та зручної підтримки програмного коду, було написано велика кількість тестів, яка покривала велику функціональність додатку та використано метод розробки програмного забезпечення через тестування, а саме TDD.

Результатом виконання магістерської дисертації є універсальна комплексна система, яка містить в собі алгоритми здатні повністю бути автономні та коректно, правильно роботи аналіз і проводити балансування навантаження планувальника завдань. Магістерська дисертація повністю відповідає технічному завданню та вирішує всі поставлені перед нею задачі.

ПЕРЕЛІК ПОСИЛАНЬ

1. Job Scheduling for Cloud Computing Using Neural Networks. [Електронний ресурс] – Режим доступу:http://file.scirp.org/pdf/CN_2014082810201017.pdf
2. Analysis of Job Scheduling Algorithms in Cloud Computing. [Електронний ресурс] – Режим доступу: https://pdfs.semanticscholar.org/0da5/b58196c4363313d26c9e216462c96d5cf9d2.pdf
3. Implementing a Process Scheduler Using Neural Network Technology. [Електронний ресурс] – Режим доступу: https://theses.ubn.ru.nl/bitstream/handle/123456789/168/Bex,%20P.MScThesis.pdf?sequence=1
4. Pattern: Microservice Architecture. [Електронний ресурс] – Режим доступу: https://microservices.io/patterns/microservices.html
5. Microservices architecture style. [Електронний ресурс] – Режим доступу: https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices
6. Микросервисы (Microservices). [Електронний ресурс] – Режим доступу: https://habrahabr.ru/post/249183/
7. Microservices. [Електронний ресурс] – Режим доступу: https://aws.amazon.com/ru/microservices/
8. Microservices vs. SOA 2. [Електронний ресурс] – Режим доступу: https://dzone.com/articles/microservices-vs-soa-2
9. Триярусна архітектура. [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/%D0%A2%D1%80%D0%B8%D1%8F%D1%80%D1%83%D1%81%D0%BD%D0%B0_%D0%B0%D1%80%D1%85%D1%96%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0
10. Difference between Microservices Architecture and SOA. [Електронний ресурс] – Режим доступу: http://stackoverflow.com/questions/25501098/difference-between-microservices-architecture-and-soa

11. Microservices a definition of this new architectural term. [Електронний ресурс] – Режим доступу: https://www.martinfowler.com/articles/microservices.html
12. 10 интересных вещей о платформе DotNet Core. [Електронний ресурс] – Режим доступу: https://proglib.io/p/10-things-about-dotnet-core/
13. Что такое Docker и технология контейнеров Linux. [Електронний ресурс] – Режим доступу: https://vps.ua/blog/docker-and-linux-containers/
14. Балансування навантаження: основні алгоритми і методи. [Електронний ресурс] – Режим доступу: https://habr.com/company/selectel/blog/250201/
15. Round-robin (алгоритм). [Електронний ресурс] – Режим доступу: https://ru.wikipedia.org/wiki/Round-robin_(%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82)
16. Алгоритм Літтла - метод вирішення задачі комівояжера. [Електронний ресурс] – Режим доступу: https://intellect.ml/algoritm-littla-metod-resheniya-zadachi-kommivoyazhera-7734
17. Обучение нейронной сети. [Електронний ресурс] – Режим доступу: https://neuronus.com/theory/nn/238-obucheniya-nejronnoi seti.html#sel=3:2,3:2
18. How neural networks are trained. [Електронний ресурс] – Режим доступу: https://ml4a.github.io/ml4a/how_neural_networks_are_trained/
19. Stochastic Gradient Descent with momentum. [Електронний ресурс] – Режим доступу: https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d
20. Многослойные перцептроны. [Електронний ресурс] – Режим доступу: https://neuronus.com/theory/nn/953-mnogoslojnye-pertseptrony.html#sel=46:1,46:7;2:2,2:2
21. Neural networks and back-propagation explained in a simple way. [Електронний ресурс] – Режим доступу: https://medium.com/datathings/neural-networks-and-backpropagation-explained-in-a-simple-way-f540a3611f5e

ДОДАТКИ

ДОДАТОК А

Текст програмного коду

```
public class SimpleNeuralNetwork
    {
        private NeuralLayerFactory _layerFactory;

        public List<NeuralLayer> _layers;
        public Dictionary<int, double[]> _neuronErrors;
        public double _learningRate;
        public double[][] _expectedResult;

        /// <summary>
        /// Constructor of the Neural Network.
        /// Note:
        /// Initialy input layer with defined number of inputs will be created.
        /// </summary>
        /// <param name="numberOfInputNeurons">
        /// Number of neurons in input layer.
        /// </param>
        public SimpleNeuralNetwork(int numberOfInputNeurons)
        {
            _layers = new List<NeuralLayer>();
            _neuronErrors = new Dictionary<int, double[]>();
            _layerFactory = new NeuralLayerFactory();

            // Create input layer that will collect inputs.
            CreateInputLayer(numberOfInputNeurons);

            _learningRate = 2.95;
        }

        /// <summary>
        /// Add layer to the neural network.
        /// Layer will automatically be added as the output layer to the last layer in the
neural network.
        /// </summary>
        public void AddLayer(NeuralLayer newLayer)
        {
            if (_layers.Any())
            {
                var lastLayer = _layers.Last();
```

```
                newLayer.ConnectLayers(lastLayer);
            }

            _layers.Add(newLayer);
            _neuronErrors.Add(_layers.Count - 1, new double[newLayer.Neurons.Count]);
        }

        /// <summary>
        /// Push input values to the neural network.
        /// </summary>
        public void PushInputValues(double[] inputs)
        {
            _layers.First().Neurons.ForEach(x =>
x.PushValueOnInput(inputs[_layers.First().Neurons.IndexOf(x)]));
        }

        /// <summary>
        /// Set expected values for the outputs.
        /// </summary>
        public void PushExpectedValues(double[][] expectedOutputs)
        {
            _expectedResult = expectedOutputs;
        }

        /// <summary>
        /// Calculate output of the neural network.
        /// </summary>
        /// <returns></returns>
        public List<double> GetOutput()
        {
            var returnValue = new List<double>();

            _layers.Last().Neurons.ForEach(neuron =>
            {
                returnValue.Add(neuron.CalculateOutput());
            });

            return returnValue;
        }

        /// <summary>
        /// Train neural network.
        /// </summary>
```

```
        /// <param name="inputs">Input values.</param>
        /// <param name="numberOfEpochs">Number of epochs.</param>
        public void Train(double[][] inputs, int numberOfEpochs)
        {
            double totalError = 0;

            for (int i = 0; i < numberOfEpochs; i++)
            {
                for (int j = 0; j < inputs.GetLength(0); j++)
                {
                    PushInputValues(inputs[j]);

                    var outputs = new List<double>();

                    // Get outputs.
                    _layers.Last().Neurons.ForEach(x =>
                    {
                        outputs.Add(x.CalculateOutput());
                    });

                    // Calculate error by summing errors on all output neurons.
                    totalError = CalculateTotalError(outputs, j);
                    HandleOutputLayer(j);
                    HandleHiddenLayers();
                }
            }
        }

        /// <summary>
        /// Hellper function that creates input layer of the neural network.
        /// </summary>
        private void CreateInputLayer(int numberOfInputNeurons)
        {
            var inputLayer = _layerFactory.CreateNeuralLayer(numberOfInputNeurons, new
RectifiedActivationFuncion(), new WeightedSumFunction());
            inputLayer.Neurons.ForEach(x => x.AddInputSynapse(0));
            this.AddLayer(inputLayer);
        }

        /// <summary>
        /// Hellper function that calculates total error of the neural network.
        /// </summary>
        private double CalculateTotalError(List<double> outputs, int row)
```

```
        {
            double totalError = 0;

            outputs.ForEach(output =>
            {
                var error = Math.Pow(output - _expectedResult[row][outputs.IndexOf(output)],
2);
                totalError += error;
            });

            return totalError;
        }

        /// <summary>
        /// Hellper function that runs backpropagation algorithm on the output layer of the
network.
        /// </summary>
        /// <param name="row">
        /// Input/Expected output row.
        /// </param>
        private void HandleOutputLayer(int row)
        {
            _layers.Last().Neurons.ForEach(neuron =>
            {
                neuron.Inputs.ForEach(connection =>
                {
                    var output = neuron.CalculateOutput();
                    var netInput = connection.GetOutput();

                    var expectedOutput =
_expectedResult[row][_layers.Last().Neurons.IndexOf(neuron)];

                    var nodeDelta = (expectedOutput - output) * output * (1 - output);
                    var delta = -1 * netInput * nodeDelta;

                    connection.UpdateWeight(_learningRate, delta);

                    neuron.PreviousPartialDerivate = nodeDelta;
                });
            });
        }

        /// <summary>
```

```
        /// Hellper function that runs backpropagation algorithm on the hidden layer of the
network.
        /// </summary>
        /// <param name="row">
        /// Input/Expected output row.
        /// </param>
        private void HandleHiddenLayers()
        {
            for (int k = _layers.Count - 2; k > 0; k--)
            {
                _layers[k].Neurons.ForEach(neuron =>
                {
                    neuron.Inputs.ForEach(connection =>
                    {
                        var output = neuron.CalculateOutput();
                        var netInput = connection.GetOutput();
                        double sumPartial = 0;

                        _layers[k + 1].Neurons
                        .ForEach(outputNeuron =>
                        {
                            outputNeuron.Inputs.Where(i => i.IsFromNeuron(neuron.Id))
                            .ToList()
                            .ForEach(outConnection =>
                            {
                                sumPartial += outConnection.PreviousWeight *
outputNeuron.PreviousPartialDerivate;
                            });
                        });

                        var delta = -1 * netInput * sumPartial * output * (1 - output);
                        connection.UpdateWeight(_learningRate, delta);
                    });
                });
            }
        }
    }

    public class NeuralLayerFactory
    {
        public NeuralLayer CreateNeuralLayer(
            int numberOfNeurons,
            IActivationFunction activationFunction,
            IInputFunction inputFunction)
```

```
        {
            var layer = new NeuralLayer();

            for (int i = 0; i < numberOfNeurons; i++)
            {
                var neuron = new Neuron.Neuron(activationFunction, inputFunction);
                layer.Neurons.Add(neuron);
            }

            return layer;
        }
    }
public class NeuralLayer
    {
        public List<INeuron> Neurons;

        public NeuralLayer()
        {
            Neurons = new List<INeuron>();
        }

        /// <summary>
        /// Connecting two layers.
        /// </summary>
        public void ConnectLayers(NeuralLayer inputLayer)
        {
            var combos = Neurons.SelectMany(neuron => inputLayer.Neurons, (neuron, input) =>
new { neuron, input });
            combos.ToList().ForEach(x => x.neuron.AddInputNeuron(x.input));
        }
    }
public class Synapse : ISynapse
    {
        internal INeuron _fromNeuron;
        internal INeuron _toNeuron;

        /// <summary>
        /// Weight of the connection.
        /// </summary>
        public double Weight { get; set; }

        /// <summary>
```

```
/// Weight that connection had in previous itteration.
/// Used in training process.
/// </summary>
public double PreviousWeight { get; set; }

public Synapse(INeuron fromNeuraon, INeuron toNeuron, double weight)
{
    _fromNeuron = fromNeuraon;
    _toNeuron = toNeuron;

    Weight = weight;
    PreviousWeight = 0;
}

public Synapse(INeuron fromNeuraon, INeuron toNeuron)
{
    _fromNeuron = fromNeuraon;
    _toNeuron = toNeuron;

    var tmpRandom = new Random();
    Weight = tmpRandom.NextDouble();
    PreviousWeight = 0;
}

/// <summary>
/// Get output value of the connection.
/// </summary>
/// <returns>
/// Output value of the connection.
/// </returns>
public double GetOutput()
{
    return _fromNeuron.CalculateOutput();
}

/// <summary>
/// Checks if Neuron has a certain number as an input neuron.
/// </summary>
/// <param name="fromNeuronId">Neuron Id.</param>
/// <returns>
/// True - if the neuron is the input of the connection.
/// False - if the neuron is not the input of the connection.
/// </returns>
```

```
        public bool IsFromNeuron(Guid fromNeuronId)
        {
            return _fromNeuron.Id.Equals(fromNeuronId);
        }

        /// <summary>
        /// Update weight.
        /// </summary>
        /// <param name="learningRate">Chossen learning rate.</param>
        /// <param name="delta">Calculated difference for which weight of the connection
needs to be modified.</param>
        public void UpdateWeight(double learningRate, double delta)
        {
            PreviousWeight = Weight;
            Weight += learningRate * delta;
        }
    }
public class Neuron : INeuron
    {
        private IActivationFunction _activationFunction;
        private IInputFunction _inputFunction;

        /// <summary>
        /// Input connections of the neuron.
        /// </summary>
        public List<ISynapse> Inputs { get; set; }

        /// <summary>
        /// Output connections of the neuron.
        /// </summary>
        public List<ISynapse> Outputs { get; set; }

        public Guid Id { get; private set; }

        /// <summary>
        /// Calculated partial derivate in previous iteration of training process.
        /// </summary>
        public double PreviousPartialDerivate { get; set; }

        public Neuron(IActivationFunction activationFunction, IInputFunction inputFunction)
        {
            Id = Guid.NewGuid();
            Inputs = new List<ISynapse>();
```

```
            Outputs = new List<ISynapse>();

            _activationFunction = activationFunction;
            _inputFunction = inputFunction;
        }

        /// <summary>
        /// Connect two neurons.
        /// This neuron is the output neuron of the connection.
        /// </summary>
        /// <param name="inputNeuron">Neuron that will be input neuron of the newly created
connection.</param>
        public void AddInputNeuron(INeuron inputNeuron)
        {
            var synapse = new Synapse(inputNeuron, this);
            Inputs.Add(synapse);
            inputNeuron.Outputs.Add(synapse);
        }

        /// <summary>
        /// Connect two neurons.
        /// This neuron is the input neuron of the connection.
        /// </summary>
        /// <param name="outputNeuron">Neuron that will be output neuron of the newly
created connection.</param>
        public void AddOutputNeuron(INeuron outputNeuron)
        {
            var synapse = new Synapse(this, outputNeuron);
            Outputs.Add(synapse);
            outputNeuron.Inputs.Add(synapse);
        }

        /// <summary>
        /// Calculate output value of the neuron.
        /// </summary>
        /// <returns>
        /// Output of the neuron.
        /// </returns>
        public double CalculateOutput()
        {
            return
_activationFunction.CalculateOutput(_inputFunction.CalculateInput(this.Inputs));
        }
```

```
    /// <summary>
    /// Input Layer neurons just receive input values.
    /// For this they need to have connections.
    /// This function adds this kind of connection to the neuron.
    /// </summary>
    /// <param name="inputValue">
    /// Initial value that will be "pushed" as an input to connection.
    /// </param>
    public void AddInputSynapse(double inputValue)
    {
        var inputSynapse = new InputSynapse(this, inputValue);
        Inputs.Add(inputSynapse);
    }

    /// <summary>
    /// Sets new value on the input connections.
    /// </summary>
    /// <param name="inputValue">
    /// New value that will be "pushed" as an input to connection.
    /// </param>
    public void PushValueOnInput(double inputValue)
    {
        ((InputSynapse)Inputs.First()).Output = inputValue;
    }
}
public class StepActivationFunction : IActivationFunction
{
    private double _treshold;

    public StepActivationFunction(double treshold)
    {
        _treshold = treshold;
    }

    public double CalculateOutput(double input)
    {
        return Convert.ToDouble(input > _treshold);
    }
}
public class SigmoidActivationFunction : IActivationFunction
{
    private double _coeficient;
```

```
        public SigmoidActivationFunction(double coeficient)
        {
            _coeficient = coeficient;
        }

        public double CalculateOutput(double input)
        {
            return (1 / (1 + Math.Exp(-input * _coeficient)));
        }
    }
    public class RectifiedActivationFuncion : IActivationFunction
    {
        public double CalculateOutput(double input)
        {
            return Math.Max(0, input);
        }
    }
    public interface IActivationFunction
    {
        double CalculateOutput(double input);
    }
public class RocketConsoleLogger : ILog, ILogAsync
    {
        public LogLevel LogLevel { get; set; }
        public LogType LogType { get; set; }
        public LogParameter LogParameter { get; set; }
        public LogNotification LogNotification { get; set; }

        #region ILog implementation
        public bool Log(string message)
        {
            throw new NotImplementedException();
        }

        public bool Log(string message, Exception exception)
        {
            throw new NotImplementedException();
        }

        public bool Log(string message, Exception exception, LogLevel logLevel)
        {
```

```
    throw new NotImplementedException();
}

public bool Log(string message,
    Exception exception,
    LogLevel logLevel,
    LogNotification logNotification)
{
    throw new NotImplementedException();
}

public bool Log(string message,
    Exception exception,
    LogLevel logLevel,
    LogNotification logNotification,
    LogParameter logParameter)
{
    throw new NotImplementedException();
}
#endregion

#region ILogAsync implementation
public bool LogAsync(string message)
{
    throw new NotImplementedException();
}

public bool LogAsync(string message, Exception exception)
{
    throw new NotImplementedException();
}

public bool LogAsync(string message, Exception exception, LogLevel logLevel)
{
    throw new NotImplementedException();
}

public bool LogAsync(string message,
    Exception exception,
    LogLevel logLevel,
    LogNotification logNotification)
{
    throw new NotImplementedException();
```

```
        }

        public bool LogAsync(string message,
            Exception exception,
            LogLevel logLevel,
            LogNotification logNotification,
            LogParameter logParameter)
        {
            throw new NotImplementedException();
        }
        #endregion
    }
public class RocketLogger : ILog, ILogAsync
    {
        public LogLevel LogLevel { get; set; }
        public LogType LogType { get; set; }
        public LogParameter LogParameter { get; set; }
        public LogNotification LogNotification { get; set; }

        #region ILog implementation
        public bool Log(string message)
        {
            throw new NotImplementedException();
        }

        public bool Log(string message, Exception exception)
        {
            throw new NotImplementedException();
        }

        public bool Log(string message, Exception exception, LogLevel logLevel)
        {
            throw new NotImplementedException();
        }

        public bool Log(string message,
            Exception exception,
            LogLevel logLevel,
            LogNotification logNotification)
        {
            throw new NotImplementedException();
        }
```

```
public bool Log(string message,
    Exception exception,
    LogLevel logLevel,
    LogNotification logNotification,
    LogParameter logParameter)
{
    throw new NotImplementedException();
}
#endregion

#region ILogAsync implementation
public bool LogAsync(string message)
{
    throw new NotImplementedException();
}

public bool LogAsync(string message, Exception exception)
{
    throw new NotImplementedException();
}

public bool LogAsync(string message, Exception exception, LogLevel logLevel)
{
    throw new NotImplementedException();
}

public bool LogAsync(string message,
    Exception exception,
    LogLevel logLevel,
    LogNotification logNotification)
{
    throw new NotImplementedException();
}

public bool LogAsync(string message,
    Exception exception,
    LogLevel logLevel,
    LogNotification logNotification,
    LogParameter logParameter)
{
    throw new NotImplementedException();
}
```

```
        #endregion
    }

/// <summary>
    /// Class like a builder
    /// Step by step initialize database environment for working with postgre data base
    /// </summary>
    public class Initializer : IInitializer
    {
        /// <summary>
        /// Path to folder where database setup
        /// </summary>
        private readonly string _mainInstallPath;

        /// <summary>
        /// Path to scripts folder
        /// </summary>
        private readonly string _sqlScriptsPath;



        /// <summary>
        /// Connection string
        /// </summary>
        private readonly string _connectionString;

        /// <summary>
        /// Constructor
        /// </summary>
        /// <param name="mainInstallPath">Path to folder where database environment will be
setup</param>
        /// <param name="sqlScriptsPath">Path to folder of database scripts
destinations</param>
        /// <param name="connectionString">Connection string to database</param>
        public Initializer(
            string mainInstallPath,
            string sqlScriptsPath,
            string connectionString)
        {
            _mainInstallPath = mainInstallPath;
            _sqlScriptsPath = sqlScriptsPath;
            _connectionString = connectionString;
        }

        /// <summary>
```

```
        /// Main method to Setup all database environment
        /// </summary>
        public void SetupEnvironment()
        {
            try
            {
                CreateDatabase();

                CreateTables();

                CreatePackages();

                CreateViews();
            }
            catch (Exception e)
            {
                Console.ForegroundColor = ConsoleColor.Red;
                Console.WriteLine(e);
                Console.ResetColor();

                throw;
            }
        }

        public void CreateDatabase()
        {
            string sqlScript = GetResourseScript("CreateDatabase");

            try
            {
                using (NpgsqlConnection connection = new
NpgsqlConnection(_connectionString))
                {
                    using (NpgsqlCommand command = new NpgsqlCommand(sqlScript))
                    {
                        connection.Open();
                        command.ExecuteNonQuery();
                        connection.Close();
                    }
                }
                // TODO: If everything OK need logging
            }
            catch (Exception e)
```

```
            {
                // TODO: Need logging
                throw e;
            }
        }

        public void CreateTables()
        {
            throw new NotImplementedException();
        }

        public void CreateSequences()
        {
            throw new NotImplementedException();
        }

        public void CreatePackages()
        {
            throw new NotImplementedException();
        }

        public void CreateViews()
        {
            throw new NotImplementedException();
        }


        /// <summary>
        /// Get resourse sql script to work
        /// </summary>
        /// <param name="scriptName"></param>
        /// <returns></returns>
        private string GetResourseScript(string scriptName)
        {
            string resourse = String.Empty;
            var assembly = Assembly.GetExecutingAssembly();

            try
            {
                using (Stream stream =
assembly.GetManifestResourceStream($"{scriptName}.sql"))
                {
                    using (StreamReader reader = new StreamReader(stream))
```

```
                {
                    resourse = reader.ReadToEnd();
                }
            }
        }
        catch (Exception e)
        {
            throw e;
        }

        return resourse;
    }
}
public class SimpleButtons
{
    public SimpleButtons()
    {

    }

    //public IEnumerable<KeyboardButton> GetSimpleButtons1()
    //{
    //    return new KeyboardButton[][]
    //    {
    //        new KeyboardButton[]
    //        {
    //            new KeyboardButton("item"),
    //            new KeyboardButton("item")
    //        },
    //        new KeyboardButton[]
    //        {
    //            new KeyboardButton("item")
    //        }
    //    };
    //}

    public ReplyKeyboardMarkup GetKeywordButtons()
    {
        return new ReplyKeyboardMarkup
        {
            Keyboard = new[]
            {
```

```
                new KeyboardButton[]
                {
                    new KeyboardButton("item"),
                    new KeyboardButton("item")
                },
                new KeyboardButton[]
                {
                    new KeyboardButton("item")
                }
            }
        };
    }

    public InlineKeyboardMarkup GetInlineButtons()
    {
        var keys = new[]
        {
            new InlineKeyboardButton[]
            {
                new InlineKeyboardButton()
                {
                    Text = "Firts button ",
                    CallbackData = "Test 1"
                },
                new InlineKeyboardButton()
                {
                    Text = "Second button ",
                    CallbackData = "Test 2"
                }
            },
            new InlineKeyboardButton[]
            {
                new InlineKeyboardButton()
                {
                    Text = "Third button ",
                    CallbackData = "Test 3"
                },
                new InlineKeyboardButton()
                {
                    Text = "Test button ",
                    CallbackData = "Test 3"
                }
            }
```

```
            };

            return new InlineKeyboardMarkup(keys);
        }
    }


/// <summary>
    /// Strat server - ngrok http 54146 -host-header="localhost:54146"
    /// </summary>
    [Produces("application/json")]
    [Route("bot")]
    public class BotController : Controller
    {
        private readonly TelegramBotClient _client =
            new TelegramBotClient("");

        [HttpGet]
        public IActionResult Get()
        {
            return Ok("Test");
        }

        [HttpPost]
        public async void Post([FromBody]Update update)
        {
            SimpleButtons buttons = new SimpleButtons();

            if (update == null)
                return;

            var message = update.Message;
            if (message?.Type == MessageType.Text)
            {

                if (!String.IsNullOrEmpty(message.Chat.Username))
                    await _client.SendTextMessageAsync(message.Chat.Id,
                        $"Hi, {message.Chat.Username}, " + message.Text, replyMarkup:
buttons.GetInlineButtons());
                else
                    await _client.SendTextMessageAsync(message.Chat.Id, message.Text,
                        replyMarkup: buttons.GetInlineButtons());
            }
```

```
        }
    }
public static class Program
    {
        public static void Main(string[] args)
        {
            // Endpoint must be configured with netsh:
            // netsh http add urlacl url=https://+:8443/ user=<username>
            // netsh http add sslcert ipport=0.0.0.0:8443 certhash=<cert thumbprint>
appid=<random guid>

            using (WebApp.Start<Startup>("https://+:8443"))
            {
                // Register WebHook
                // You should replace {YourHostname} with your Internet accessible hosname
                Bot.Api.SetWebhookAsync("https://{YourHostname}:8443/WebHook").Wait();

                Console.WriteLine("Server Started");

                // Stop Server after <Enter>
                Console.ReadLine();

                // Unregister WebHook
                Bot.Api.DeleteWebhookAsync().Wait();
            }
        }
    }

    public class Startup
    {
        public void Configuration(IAppBuilder app)
        {
            var configuration = new HttpConfiguration();

            configuration.Routes.MapHttpRoute("WebHook", "{controller}");

            app.UseWebApi(configuration);
        }
    }

    public class WebHookController : ApiController
    {
        public async Task<IHttpActionResult> Post(Update update)
```

```
        {
            var message = update.Message;

            Console.WriteLine("Received Message from {0}", message.Chat.Id);

            if (message.Type == MessageType.Text)
            {
                // Echo each Message
                await Bot.Api.SendTextMessageAsync(message.Chat.Id, message.Text);
            }
            else if (message.Type == MessageType.Photo)
            {
                // Download Photo
                var file = await
Bot.Api.GetFileAsync(message.Photo.LastOrDefault()?.FileId);

                var filename = file.FileId + "." + file.FilePath.Split('.').Last();

                using (var saveImageStream = File.Open(filename, FileMode.Create))
                {
                    await Bot.Api.DownloadFileAsync(file.FilePath, saveImageStream);
                }

                await Bot.Api.SendTextMessageAsync(message.Chat.Id, "Thx for the Pics");
            }

            return Ok();
        }
    }
public static class Program
    {
        private static readonly TelegramBotClient Bot = new TelegramBotClient("Your API
key");

        public static void Main(string[] args)
        {
            var me = Bot.GetMeAsync().Result;
            Console.Title = me.Username;

            Bot.OnMessage += BotOnMessageReceived;
            Bot.OnMessageEdited += BotOnMessageReceived;
            Bot.OnCallbackQuery += BotOnCallbackQueryReceived;
            Bot.OnInlineQuery += BotOnInlineQueryReceived;
```

```
            Bot.OnInlineResultChosen += BotOnChosenInlineResultReceived;
            Bot.OnReceiveError += BotOnReceiveError;

            Bot.StartReceiving(Array.Empty<UpdateType>());
            Console.WriteLine($"Start listening for @{me.Username}");
            Console.ReadLine();
            Bot.StopReceiving();
        }

        private static async void BotOnMessageReceived(object sender, MessageEventArgs
messageEventArgs)
        {
            var message = messageEventArgs.Message;

            if (message == null || message.Type != MessageType.Text) return;

            switch (message.Text.Split(' ').First())
            {
                // send inline keyboard
                case "/inline":
                    await Bot.SendChatActionAsync(message.Chat.Id, ChatAction.Typing);

                    await Task.Delay(500); // simulate longer running task

                    var inlineKeyboard = new InlineKeyboardMarkup(new[]
                    {
                        new [] // first row
                        {
                            InlineKeyboardButton.WithCallbackData("1.1"),
                            InlineKeyboardButton.WithCallbackData("1.2"),
                        },
                        new [] // second row
                        {
                            InlineKeyboardButton.WithCallbackData("2.1"),
                            InlineKeyboardButton.WithCallbackData("2.2"),
                        }
                    });

                    await Bot.SendTextMessageAsync(
                        message.Chat.Id,
                        "Choose",
                        replyMarkup: inlineKeyboard);
                    break;
```

```
                // send custom keyboard
                case "/keyboard":
                    ReplyKeyboardMarkup ReplyKeyboard = new[]
                    {
                        new[] { "1.1", "1.2" },
                        new[] { "2.1", "2.2" },
                    };

                    await Bot.SendTextMessageAsync(
                        message.Chat.Id,
                        "Choose",
                        replyMarkup: ReplyKeyboard);
                    break;

                // send a photo
                case "/photo":
                    await Bot.SendChatActionAsync(message.Chat.Id, ChatAction.UploadPhoto);

                    const string file = @"Files/tux.png";

                    var fileName = file.Split(Path.DirectorySeparatorChar).Last();

                    using (var fileStream = new FileStream(file, FileMode.Open,
FileAccess.Read, FileShare.Read))
                    {
                        await Bot.SendPhotoAsync(
                            message.Chat.Id,
                            fileStream,
                            "Nice Picture");
                    }
                    break;

                // request location or contact
                case "/request":
                    var RequestReplyKeyboard = new ReplyKeyboardMarkup(new[]
                    {
                        KeyboardButton.WithRequestLocation("Location"),
                        KeyboardButton.WithRequestContact("Contact"),
                    });

                    await Bot.SendTextMessageAsync(
                        message.Chat.Id,
```

```
                        "Who or Where are you?",
                        replyMarkup: RequestReplyKeyboard);
                    break;

                default:
                    const string usage = @"
Usage:
/inline   - send inline keyboard
/keyboard - send custom keyboard
/photo    - send a photo
/request  - request location or contact";

                    await Bot.SendTextMessageAsync(
                        message.Chat.Id,
                        usage,
                        replyMarkup: new ReplyKeyboardRemove());
                    break;
            }
        }

        private static async void BotOnCallbackQueryReceived(object sender,
CallbackQueryEventArgs callbackQueryEventArgs)
        {
            var callbackQuery = callbackQueryEventArgs.CallbackQuery;

            await Bot.AnswerCallbackQueryAsync(
                callbackQuery.Id,
                $"Received {callbackQuery.Data}");

            await Bot.SendTextMessageAsync(
                callbackQuery.Message.Chat.Id,
                $"Received {callbackQuery.Data}");
        }
```

ДОДАТОК Б

Текст програмного коду тестів

```
[TestClass]
    public class NeuronTest
    {
        [TestMethod]
        public void Constructor_ActivationFunctionInputFunction_NeuronInitialized()
        {
            var activationFunction = new Mock<IActivationFunction>();
            var inputFunction = new Mock<IInputFunction>();

            var neuron = new Neuron.Neuron(activationFunction.Object, inputFunction.Object);

            Assert.IsNotNull(neuron);
            Assert.AreNotEqual(Guid.Empty, neuron.Id);
            Assert.IsNotNull(neuron.Inputs);
            Assert.IsNotNull(neuron.Outputs);
        }

        [TestMethod]
        public void AddInput_NeuronPassed_ConnectionCreated()
        {
            var activationFunction = new Mock<IActivationFunction>();
            var inputFunction = new Mock<IInputFunction>();

            var neuron = new Neuron.Neuron(activationFunction.Object, inputFunction.Object);

            var inputNeuron = new Neuron.Neuron(activationFunction.Object,
inputFunction.Object);

            neuron.AddInputNeuron(inputNeuron);

            Assert.AreEqual(1, neuron.Inputs.Count);
        }

        [TestMethod]
        public void AddOutput_NeuronPassed_ConnectionCreated()
        {
            var activationFunction = new Mock<IActivationFunction>();
            var inputFunction = new Mock<IInputFunction>();
```

```
            var neuron = new Neuron.Neuron(activationFunction.Object, inputFunction.Object);

            var outputNeuron = new Neuron.Neuron(activationFunction.Object,
inputFunction.Object);

            neuron.AddOutputNeuron(outputNeuron);

            Assert.AreEqual(1, neuron.Outputs.Count);
        }

        [TestMethod]
        public void AddInputSynapse_SynapsePassed_ConnectionCreated()
        {
            var activationFunction = new Mock<IActivationFunction>();
            var inputFunction = new Mock<IInputFunction>();

            var neuron = new Neuron.Neuron(activationFunction.Object, inputFunction.Object);
            neuron.AddInputSynapse(0.11);

            Assert.AreEqual(1, neuron.Inputs.Count);
            Assert.AreEqual(1, neuron.Inputs.First().Weight);
            Assert.AreEqual(0.11, neuron.Inputs.First().GetOutput());
        }

        [TestMethod]
        public void CalculateOutput_MockingFunctions_OutputReturned()
        {
            var activationFunction = new Mock<IActivationFunction>();
            activationFunction.Setup(x =>
x.CalculateOutput(It.IsAny<double>())).Returns(111);

            var inputFunction = new Mock<IInputFunction>();
            inputFunction.Setup(x =>
x.CalculateInput(It.IsAny<List<ISynapse>>())).Returns(23);

            var neuron = new Neuron.Neuron(activationFunction.Object, inputFunction.Object);

            Assert.AreEqual(111, neuron.CalculateOutput());
        }

        [TestMethod]
        public void AddInputSynapse_SynapseAdded_NumberOdSynapsesIncreased()
        {
            var activationFunction = new Mock<IActivationFunction>();
```

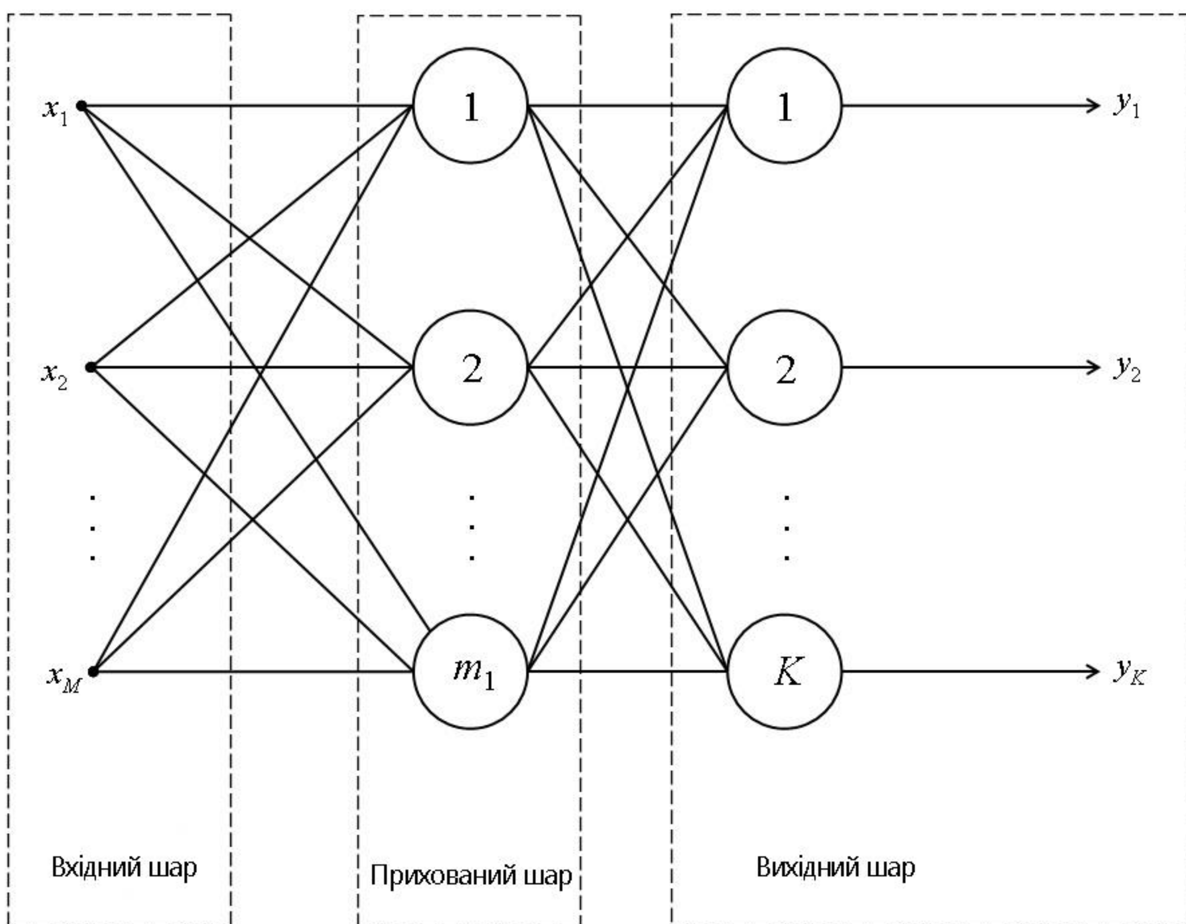
```
            activationFunction.Setup(x =>
x.CalculateOutput(It.IsAny<double>())).Returns(111);

            var inputFunction = new Mock<IInputFunction>();
            inputFunction.Setup(x =>
x.CalculateInput(It.IsAny<List<ISynapse>>())).Returns(23);

            var neuron = new Neuron.Neuron(activationFunction.Object, inputFunction.Object);

            neuron.AddInputSynapse(0);

            Assert.AreEqual(1, neuron.Inputs.Count);
        }

        [TestMethod]
        public void PushInputValueToInput_SendingValueToInput_ProperValueOnInputSet()
        {
            var activationFunction = new Mock<IActivationFunction>();
            activationFunction.Setup(x =>
x.CalculateOutput(It.IsAny<double>())).Returns(111);

            var inputFunction = new Mock<IInputFunction>();
            inputFunction.Setup(x =>
x.CalculateInput(It.IsAny<List<ISynapse>>())).Returns(23);

            var neuron = new Neuron.Neuron(activationFunction.Object, inputFunction.Object);
            neuron.AddInputSynapse(0);

            neuron.PushValueOnInput(1);
            Assert.AreEqual(1, neuron.Inputs.First().GetOutput());
        }
    }

[TestClass]
    public class SimpleNeuralNetworkTest
    {
        [TestMethod]
        public void Initialization_Constructor_NeuralNetworkInitialized()
        {
            var network = new SimpleNeuralNetwork(3);
            Assert.AreEqual(1, network._layers.Count);
            Assert.AreEqual(3, network._layers.First().Neurons.Count);

            Assert.AreEqual(2.95, network._learningRate);
        }
```

```
        [TestMethod]
        public void AddLayer_NeuralAddingNewLayer_LayerAdded()
        {
            var network = new SimpleNeuralNetwork(3);
            var layerFactory = new NeuralLayerFactory();
            network.AddLayer(layerFactory.CreateNeuralLayer(3, new
RectifiedActivationFuncion(), new WeightedSumFunction()));

            Assert.AreEqual(2, network._layers.Count);
        }

        [TestMethod]
        public void PushInputValues_ValuesSentToNetwork_ValuesSetOnInput()
        {
            var network = new SimpleNeuralNetwork(3);
            network.PushInputValues(new double[] { 3, 5, 7 });

            Assert.AreEqual(3,
network._layers.First().Neurons.First().Inputs.First().GetOutput());
        }

        [TestMethod]
        public void PushExpectedResults_ValuesSentToNetwork_ValuesStored()
        {
            var network = new SimpleNeuralNetwork(3);
            network.PushExpectedValues(new double[][] { new double[] { 3, 5, 7 } });

            Assert.AreEqual(3, network._expectedResult[0][0]);
            Assert.AreEqual(5, network._expectedResult[0][1]);
            Assert.AreEqual(7, network._expectedResult[0][2]);
        }

        [TestMethod]
        public void Train_RuningTraining_NetworkIsTrained()
        {
            var network = new SimpleNeuralNetwork(3);

            var layerFactory = new NeuralLayerFactory();
            network.AddLayer(layerFactory.CreateNeuralLayer(3, new
RectifiedActivationFuncion(), new WeightedSumFunction()));
            network.AddLayer(layerFactory.CreateNeuralLayer(1, new
SigmoidActivationFunction(0.7), new WeightedSumFunction()));
```

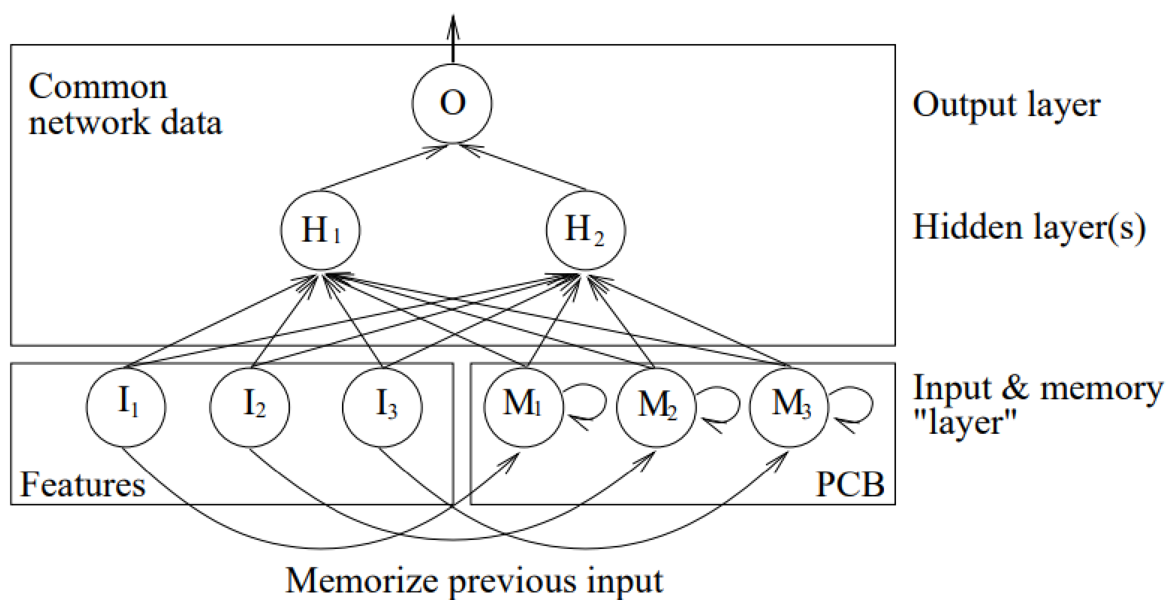
```
        network.PushExpectedValues(
            new double[][] {
                new double[] { 0 },
                new double[] { 1 },
                new double[] { 1 },
                new double[] { 0 },
                new double[] { 1 },
                new double[] { 0 },
                new double[] { 0 },
            });

        network.Train(
            new double[][] {
                new double[] { 150, 2, 0 },
                new double[] { 1002, 56, 1 },
                new double[] { 1060, 59, 1 },
                new double[] { 200, 3, 0 },
                new double[] { 300, 3, 1 },
                new double[] { 120, 1, 0 },
                new double[] { 80, 1, 0 },
            }, 10000);

        network.PushInputValues(new double[] { 1054, 54, 1 });
        var outputs = network.GetOutput();
    }
}
```

ДОДАТОК В

Графічне представлення нейронної мережі

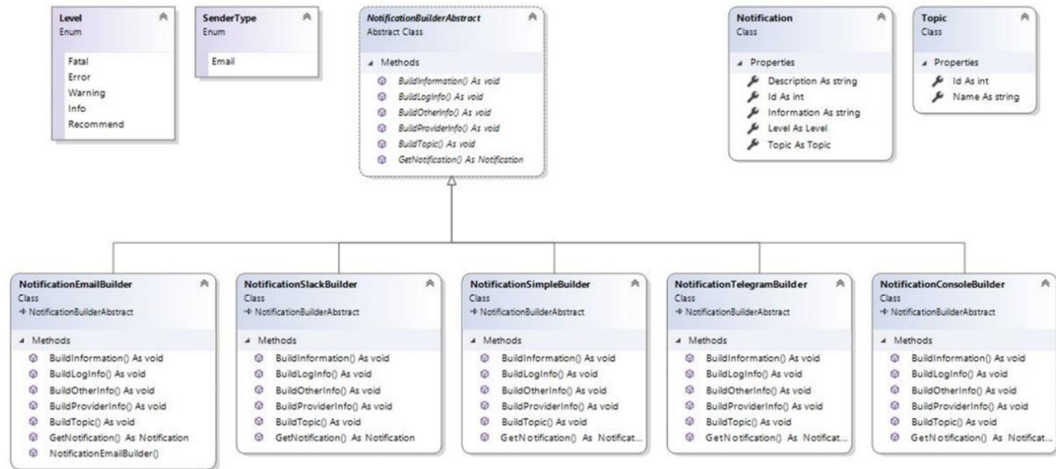


Графічне представлення багатошарового перцепторна



Архітектура мережі

Схема побудови повідомлень

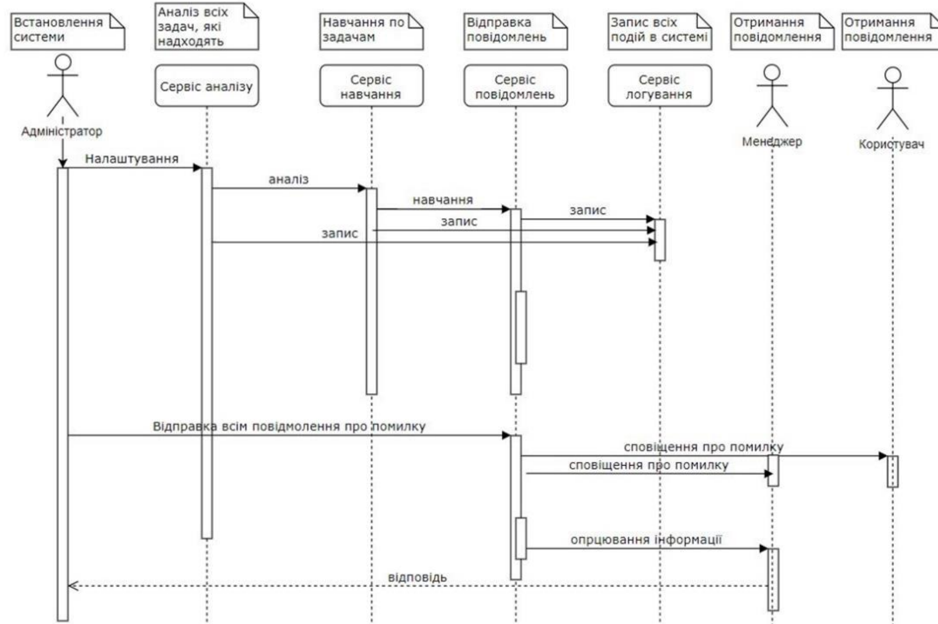


Демонстраційний плакат №1
до магістерської дисертації на тему
„Система навчання та аналізу навантаження планувальника завдань"

Розробив: Ромащенко П. С. Підпис ________

Прийняв: Пасько В. П. Підпис ________

UML–діаграма послідовності відправки повідомлень



Демонстраційний плакат №2
до магістерської дисертації на тему
„Система навчання та аналізу навантаження планувальника завдань"

Розробив: Ромащенко П. С. Підпис ________

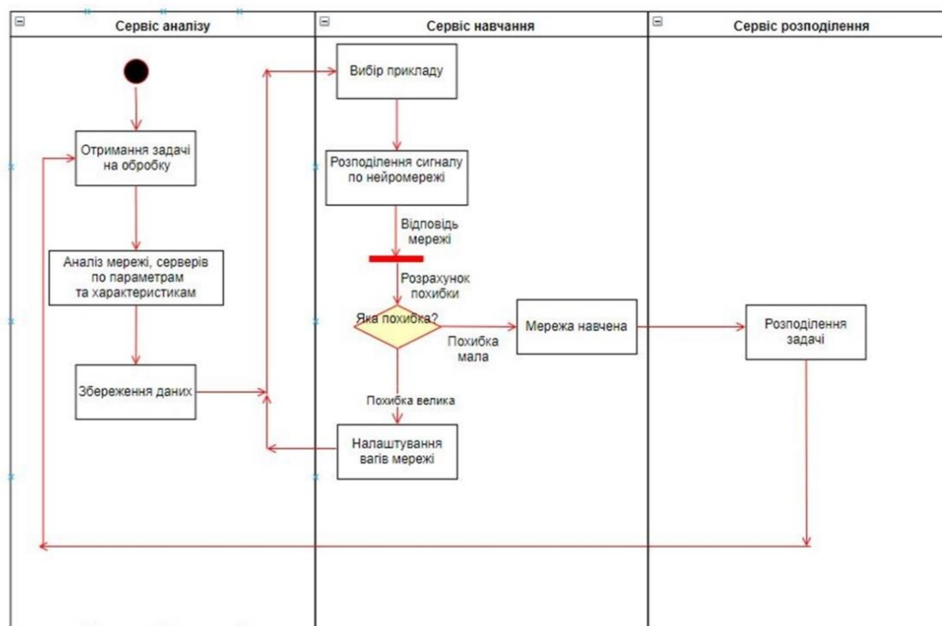
Прийняв: Пасько В. П. Підпис ________

Схема навчання нейронної мережі

Демонстраційний плакат №3
до магістерської дисертації на тему
„Система навчання та аналізу навантаження планувальника завдань"

Розробив: Ромащенко П. С. Підпис _________
Прийняв: Пасько В. П. Підпис _________

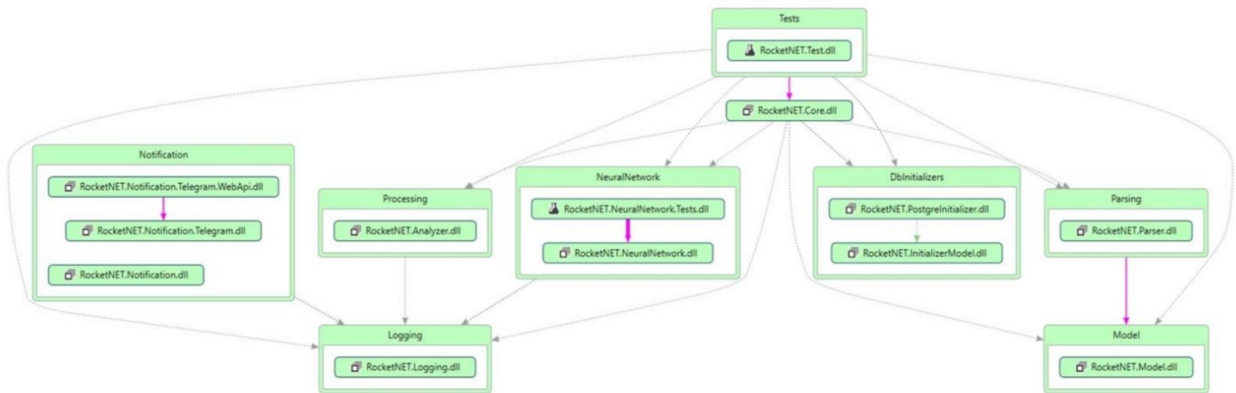
UML–діаграма діяльності роботи нейронної мережі



Демонстраційний плакат №4
до магістерської дисертації на тему
„Система навчання та аналізу навантаження планувальника завдань"

Розробив: Ромащенко П. С. Підпис _________
Прийняв: Пасько В. П. Підпис _________

Структурна схема моделі класів

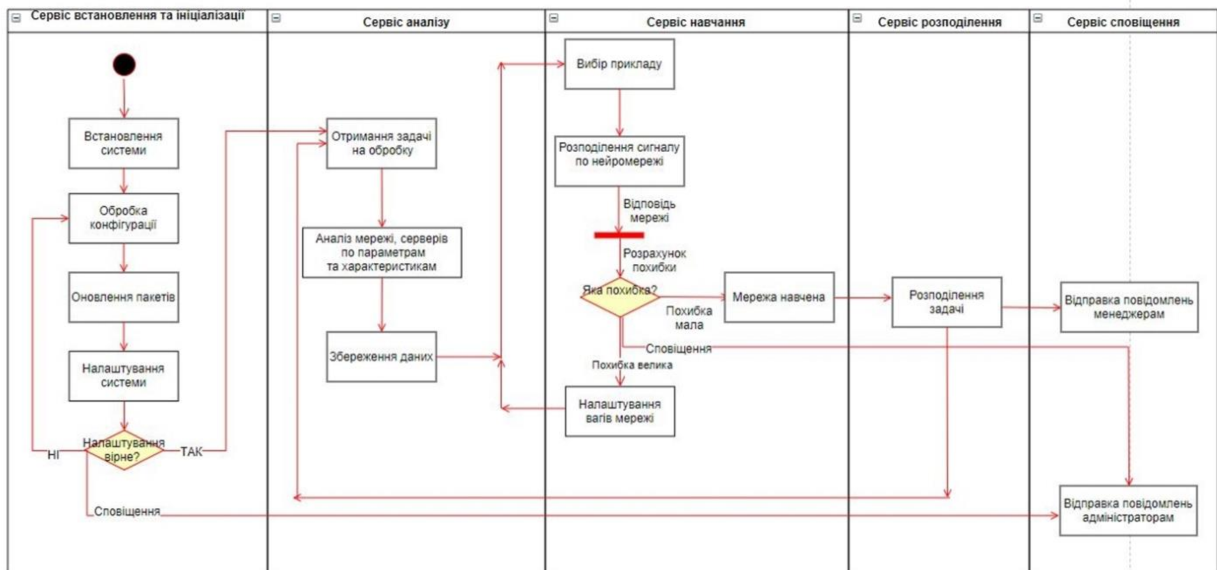


Демонстраційний плакат №5
до магістерської дисертації на тему
„Система навчання та аналізу навантаження планувальника завдань"

Розробив: Ромащенко П. С. Підпис ________

Прийняв: Пасько В. П. Підпис ________

UML–діаграма діяльності роботи в системі



Демонстраційний плакат №6
до магістерської дисертації на тему
„Система навчання та аналізу навантаження планувальника завдань"

Розробив: Ромащенко П. С. Підпис ________

Прийняв: Пасько В. П. Підпис ________