Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет _____Комп'ютерних наук_____
(повна назва)

Кафедра_____Програмної інженерії_____
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти - другий (магістерський)

Дослідження методів обробки даних для отримання температурних показників комплектуючих комп'ютера
(тема)

Виконав: студент 2 курсу, групи ІПЗм-18-2___
_____Білобров Є.О._____
(прізвище, ініціали)

спеціальності 121- Інженерія програмного забезпечення
(код і повна назва спеціальності)

Освітньо-наукової програми___
(тип програми)

Інженерія програмного забезпечення
(повна назва освітньої програми)

Керівник _____проф. Власенко Л.А_____
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри, проф. _____ З.В.Дудар

2020 р.

Харківський національний університет радіоелектроніки

Факультет_Комп'ютерних наук_____

Кафедра_Програмної інженерії_____

Рівень вищої освіти - другий (магістерський)

Спеціальність _121-Інженерія програмного забезпечення_____
(код і повна назва)

Тип програми освітньо-наукова програма_____

Освітня програма_Інженерія програмного забезпечення_____

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____»_____ 20 ___ р.

ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ

студентові ____Білоброву Євгену Олександровичу_____
(прізвище, ім'я, по батькові)

1. Тема роботи _Дослідження методів обробки даних для отримання температурних показників комплектуючих комп'ютера_____

затверджена наказом університету від "_27_" __03__2020 р № _473Ст_
заповнюється вручну після отримання наказу

2. Термін подання студентом роботи до екзаменаційної комісії
20 травня 2020 р.

3. Вихідні дані до роботи *електронні ресурси за обраною тематикою, вимоги до функціональності програми, методи теорії прийняття рішень, середовище розробки WebStorm*

4. Перелік питань, що потрібно опрацювати в роботі *мета роботи, аналіз проблемної галузі і постановка задачі, дослідження оптимізаційних моделей теорії прийняття рішень, методи згорток із теорії корисності, методи лінійного програмування із теорії дослідження операцій*

5 Консультанти розділів роботи

Найменування розділу	Консультант (посада, прізвище, ім’я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Спецчастина	доц. Мазурова О.О.		11.05.20

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1.	Аналіз предметної галузі	27.01.20 – 10.02.20	виконано
2.	Постановка задачі	11.02.20 – 13.02.20	виконано
3.	Дослідження існуючих методів та моделей	14.02.20 – 19.02.20	виконано
4.	Розробка математичних моделей та алгоритмів	20.02.20 – 3.03.20	виконано
5.	Розробка бази даних та архітектури	4.03.20 – 7.03.20	виконано
6.	Програмна реалізація	8.03.20 – 25.03.20	виконано
7.	Експериментальне дослідження розроблених алгоритмів	26.03.20 – 30.03.20	виконано
8.	Підготовка пояснювальної записки	1.04.20 – 29.04.20	виконано
9.	Підготовка презентації та доповіді	30.04.20 – 3.05.20	виконано
10.	Попередній захист	9.05.20	виконано
11.	Нормоконтроль, рецензування	10.05.20 – 15.05.20	виконано
12.	Занесення диплома в електронний архів	16.05.20	виконано
13.	Допуск до захисту у зав. кафедри	16.05.20	виконано

Дата видачі завдання _____ _____2020 р.

Студент _____ _____Білобров Є.О._____
(підпис)

Керівник роботи _____ _____проф. Власенко Л.А.____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Атестаційна робота магістра містить: 76 стор., 33 рис., 21 джерела.

КОНТРОЛЬ ТЕМПЕРАТУРИ, МЕТОДИ ОБРОБКИ ДАНИХ, IONIC4, C#, MONGODB, NEST.JS, ANGULAR.

Метою роботи є дослідження обробки даних для отримання температурних показників комплектуючих комп'ютера та розробка програми, що буде надавати інформацію з комплектуючих для користувача.

Методи проектування та розробки базуються на технології Nest.js, C#, IONIC 4, Angular, базі даних MongoDB.

У результаті роботи досліджені оптимальні методи обробки даних, на базі котрих була розроблина програма контролю температури комплектуючих.

TEMPERATURE CONTROL, DATA PROCESSING METHODS, IONIC4, C #, MONGODB, NEST.JS, ANGULAR.

The purpose of this work is to study data processing to obtain temperature indicators of computer components and to develop a software system that will provide information on components for the user.

Design and development methods are based on Nest.js, C #, IONIC 4, Angular, MongoDB database.

As a result, the optimal methods of data processing were investigated, on the basis of which a software system for temperature control of components was developed.

ЗМІСТ

ВСТУП

Актуальність даної теми дуже велика, тому що від працездатності охолоджуючих властивостей системи залежить в цілому працездатність всієї комп'ютерної системи – її продуктивність і довговічність.

Висока швидкодія сучасних комп'ютерів має свою ціну: вони споживають величезну потужність, яка розсіюється у вигляді тепла. Основні частини комп'ютера - центральний процесор, графічний процесор - вимагають власних систем охолодження; пройшли ті часи, коли ці мікросхеми задовольнялися маленьким радіатором. Новий системний блок обладнується кількома вентиляторами: як мінімум один в блоці живлення, один охолоджує процесор, серйозна відеокарта комплектується своїм вентилятором. Кілька вентиляторів встановлені в корпусі комп'ютера, зустрічаються навіть материнські плати з активним охолодженням мікросхем чіпсета. Деякі сучасні жорсткі диски також розігріваються до помітних температур.

Більшість комп'ютерів обладнується охолодженням за принципом мінімізації вартості: встановлюється один, два гучних корпусних вентилятори, процесор обладнується штатною системою охолодження. Охолодження виходить достатнім, дешевим, але дуже гучним.

Існує інший вихід – складні технічні рішення: рідинне (зазвичай водяне) охолодження, фреоновий охолодження, спеціальний алюмінієвий корпус комп'ютера, який розсіює тепло по всій своїй поверхні (по суті, працює як радіатор). Для деяких завдань такі рішення використовувати необхідно: наприклад, для студії звукозапису, де комп'ютер повинен бути повністю безшумний. Для звичайного домашнього і офісного застосування такі спеціалізовані системи надто дороги: їх ціни починаються від сотні доларів і вище. Подібні варіанти на сьогодні дуже екзотичні.

Враховуючи усе перераховане вище та за для надання розроблюваній системі актуальності необхідно забезпечити користувачеві можливість окрім стандартних функцій моніторингу температури комплектуючих, функції регулювання та оптимізації температур.

Метою роботи дослідження методів обробки даних для отримання температурних показників комплектуючих комп'ютера та впровадження програми контролю температури з використанням технології IONIC4, C#, бази даних MongoDB, локально – розгорнутого серверу Nest.js; відпрацювання основних принципів написання коду, основ рефакторінгу коду, а також загальних принципів розробки, проектування та підтримки програмного забезпечення; закріплення теоретичних знань та практичних навичок, що були набуті в ході вивчення ряду навчальних дисциплін.

В рамках роботи над серверною частиною для системи мають бути підтверджені навики що стосуються проектування, розробки. Також навики, що стосуються аналізу обраної предметної області та навики написання програмного забезпечення в цілому.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної галузі

Бездоганна робота персонального комп'ютера не може не радувати свого власника, але не завжди ЕОМ поводиться належним чином. Симптоми і причини поганої роботи можуть бути різними. Перегрів процесора і інших компонентів ПК - одна з найбільш частих проблем виходу з ладу комп'ютера.

До того ж, знову прийшло літо, а це значить, що температура повітря відчутно підскочила. І це відчувають не тільки люди, але і наші з Вами комп'ютери, яким і без того доводиться жарко, а тут ще й за вікном сонечко припікає. Коли нам жарко, що з нами відбувається? Правильно, в кращому випадку нам просто погано і дискомфортно, перестає нормально думатиметься, а в гіршому - ми ловимо сонячний удар.

Все те ж саме може статися і з комп'ютером, бо в жаркий період кулерам складніше утримувати прийнятну температуру і, як наслідок, Ваш залізний друг може почати вередувати і ризикує отримати сонячний удар у вигляді перезавантаження або виключення (або взагалі згоріти). Природно, постає питання - а як визначити перші ознаки перегріву і що робити, якщо вони виявлені?

У всесвітній павутині викладено безліч різних програм для моніторингу температури процесора і відеокарти. Серед широкого асортименту цього типу софта є як англомовні, так і русифіковані варіанти програм.

1.2 Аналіз існуючих аналогів

Оскільки задача контролю температури є досить поширеною, то існує багато утиліт – аналогів. З метою визначення недоліків серед основних конкурентів та доведення розроблюваної системи до вимог, що є найбільш актуальними для систем контролю температури, розглянемо декілька основних утиліт – аналогів.

1.2.1 Утиліта Game Assistant

Першим ми розглянемо утиліту Game Assistant. Утиліта Game Assistant дуже популярна серед геймерів, так як дозволяє моніторити температуру процесора і відеокарти прямо в ігровому додатку. Програма створена силами компанії IObit.. Відкривши програму, ми потрапимо на першу вкладку «Ігри»[2].

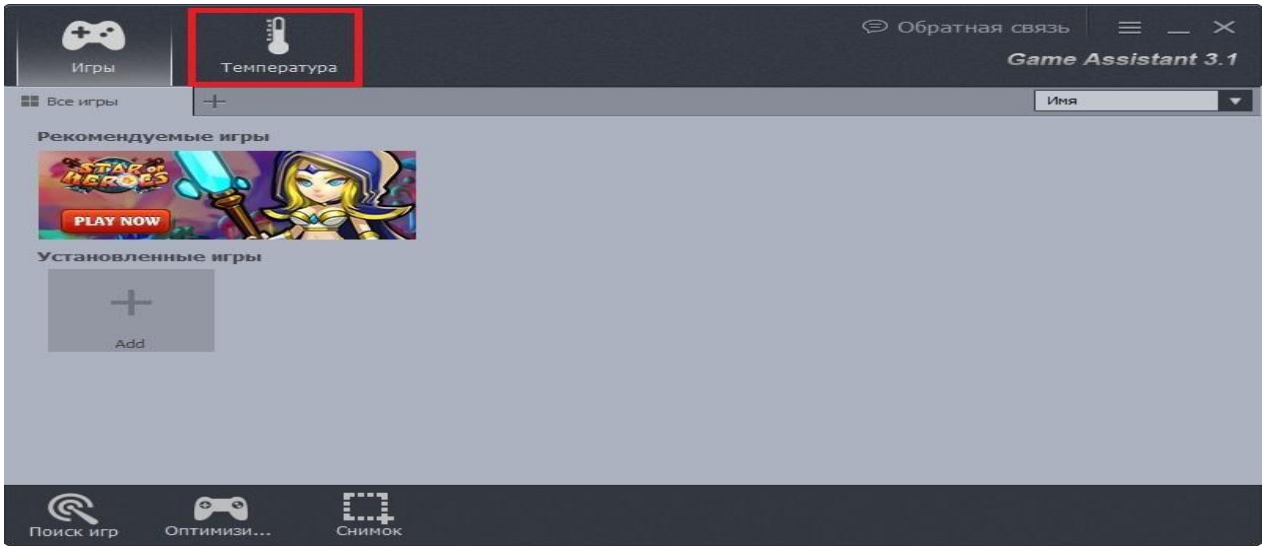


Рисунок 1.1- Вкладка «Ігри»

Нас же цікавить друга владка «Температура».

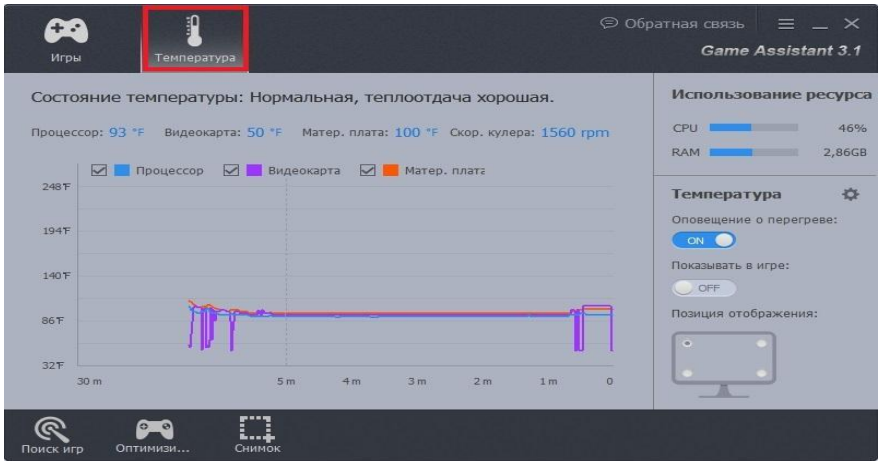


Рисунок 1.2- Вкладка «Температура»

На вкладці видно, що показники температури і графік відображаються в фаренгейтах. Щоб змінити температуру на звичну нам величину Цельсій, потрібно перейти в настройки програми і поставити соответствующею галочку, як показано на малюнку нижче.

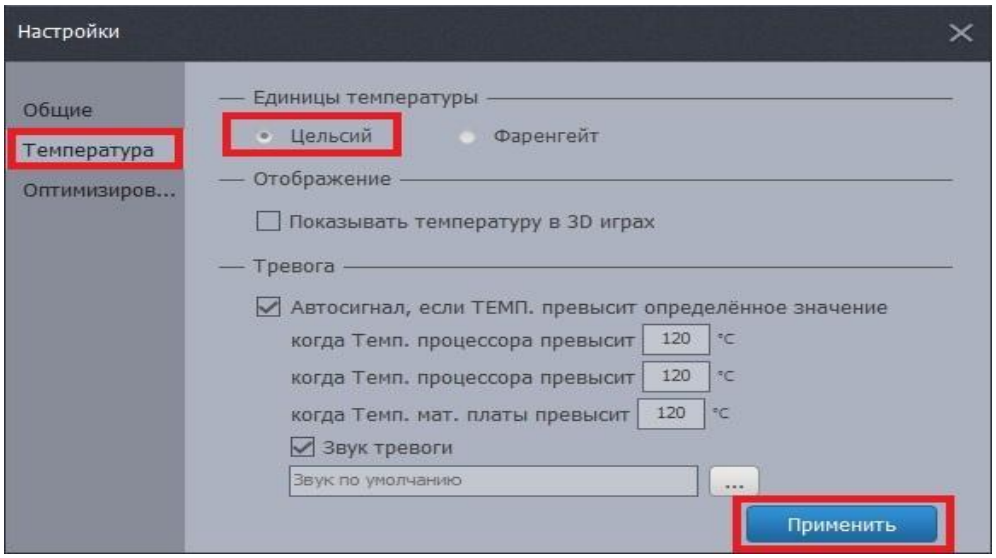


Рисунок 1.3- Вкладка «Настройки»

До переваг програми можна віднести красивий інтерфейс і функцію оповіщення при перегріві процесора. Наприклад, якщо ви граєте в важку гру, яка сильно вантажить процесор і відеокарту, то Game Assistant сповістить вас, коли вони перегріються.

З недоліків можна відзначити необхідність додавання ярликів ігор в ручному режимі. Крім того, для вступу в силу змін параметрів необхідно перезапускати додаток.

1.2.2 Утиліта AIDA64

Багатьом користувачам персональних комп'ютерів відома програма для тестування, діагностики та визначення характеристик комплектуючих EVEREST в ОС Windows. Програма EVEREST створена зусиллями розробників компанії Lavalys. З 2010 року Lavalys випустила наступницю утиліти EVEREST під назвою AIDA64. В утиліту AIDA64 включений весь функціонал EVEREST, а також запроваджено нові функції діагностики і тестування. Нам же від AIDA64 потрібен функціонал, який надає дані про теплові характеристики процесора і відеокарти[3].

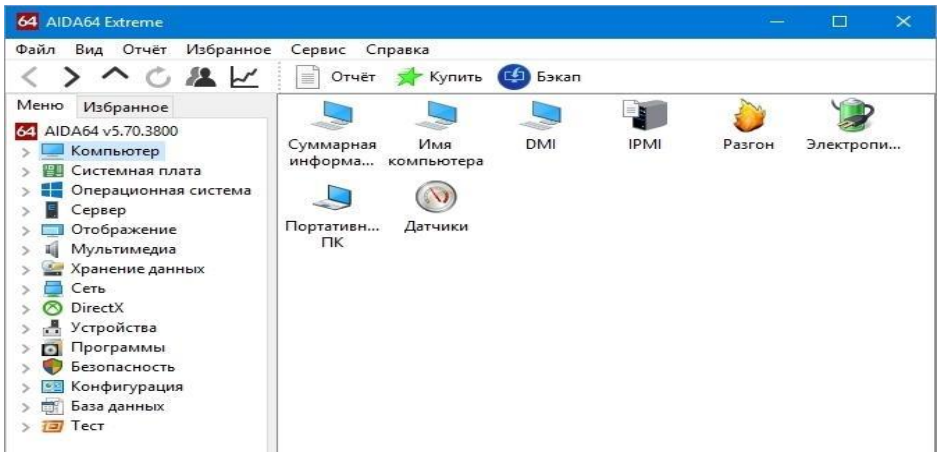


Рисунок 1.4- Утиліта AIDA64

У вікні, ми бачимо той багатий функціонал, присутній в EVEREST, який дозволяє перевірити систему. Нас же цікавить температура, тому перейдемо на вкладки «Комп'ютер / Датчики».

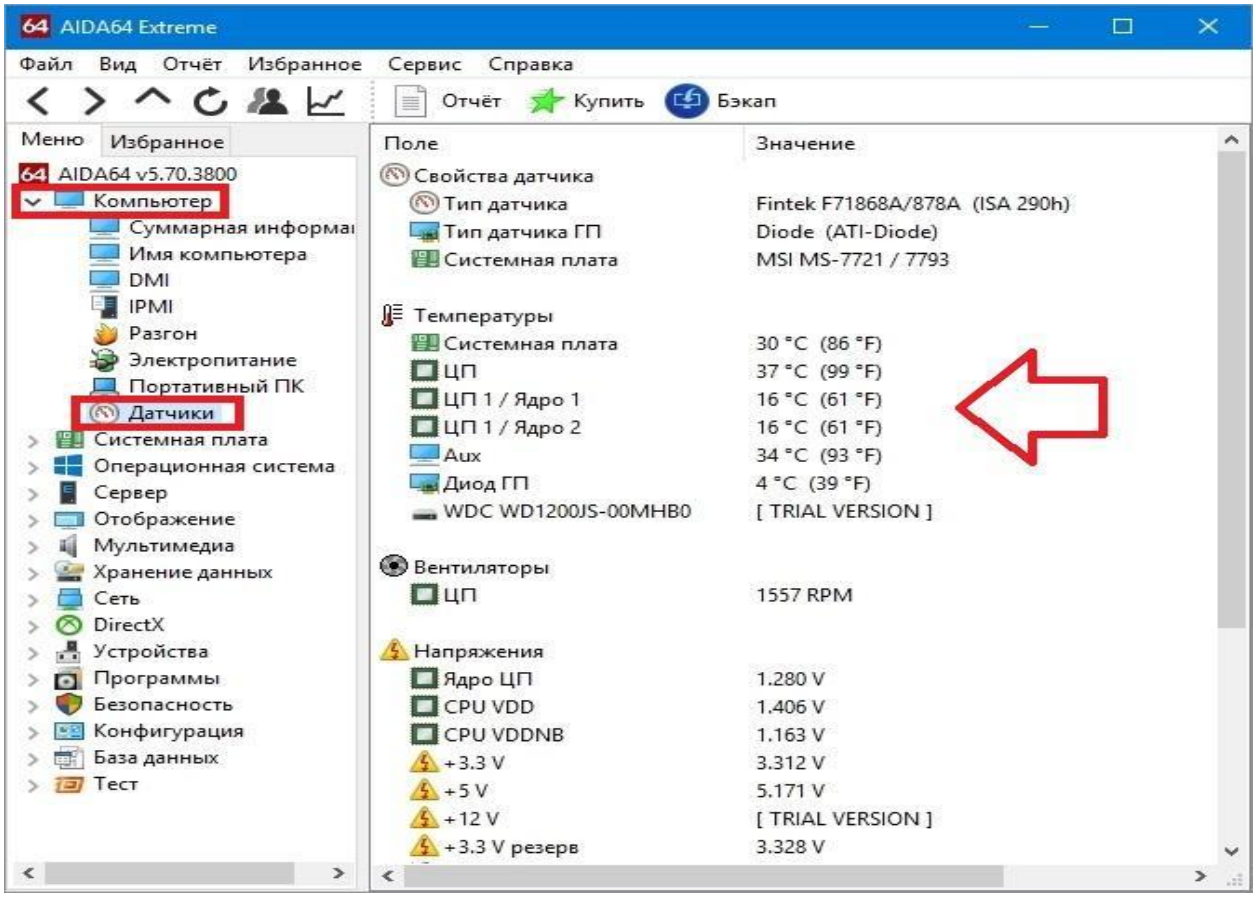


Рисунок 1.5- Показники температури

У вкладці, відображається вся інформація про датчиках, а також про температуру, яку ці датчики виводять. Крім температури процесора і відеокарти в цій вкладці можна ще дізнатися теплові характеристики HDD і материнської плати, а також дізнатися інформацію про напругу комплектуючих.

Якщо вам треба виявити проблему з процесором або відеокартою, то в цьому випадку вам на допомогу прийде «Тест стабільності системи». Запустити тест можна в меню «Сервис».

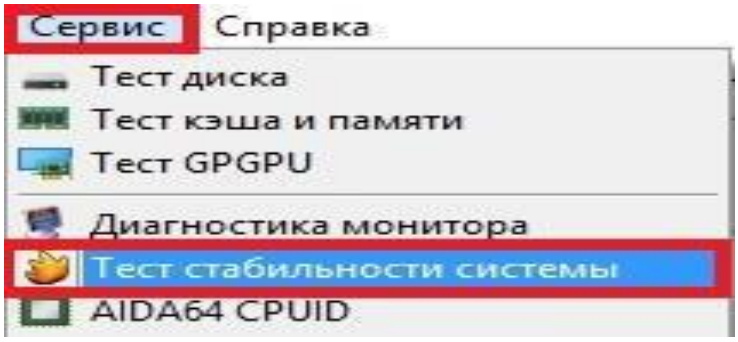


Рисунок 1.6- Тест стабільності системи

Серед недоліків варто відзначити те, що продукт має досить високу ціну та недостатньо інтуїтивне управління програмою.

1.2.3 Утиліта Wise System Monitor

З англійської назви утиліти Wise System Monitor можна зрозуміти, яке у неї призначення. Wise System Monitor поширюється безкоштовно. Після запуску

утиліти вона вбудується в області повідомлень і відобразить спливаюче вікно, в якому можна побачити температуру CPU[4].

Крім температури процесора і відеокарти в цій вкладці можна ще дізнатися теплові характеристики HDD і материнської плати, а також дізнатися інформацію про напругу комплектуючих.

Рисунок 1.7- Утиліта Wise System Monitor

Відкривши головне вікно програми, ми відразу ж потрапимо на вкладку «Процеси». Нас цікавить вкладка «Апаратне», яка неправильно переведена на російську мову.

В англійській версії вкладка має назву «Hardware Monitor», яку можна перевести як апаратний моніторинг.

Перейшовши на вкладку «Апаратне», ми потрапимо в загальний опис системи. Щоб подивитися інформацію про температуру процесора, потрібно клікнути по лівій вкладці «Процессор».

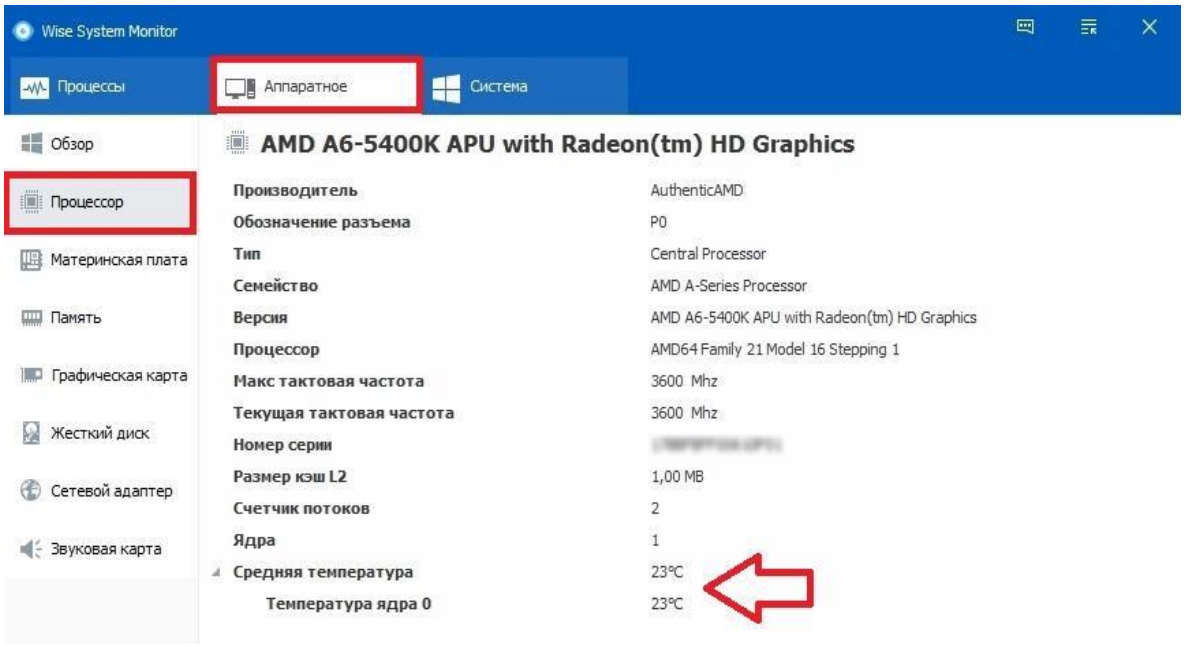


Рисунок 1.8- Вкладка «Процессор»

Клікнувши по вкладці «Графічна карта», ми дізнаємося всю інфу про відеокарту, а також її температуру.

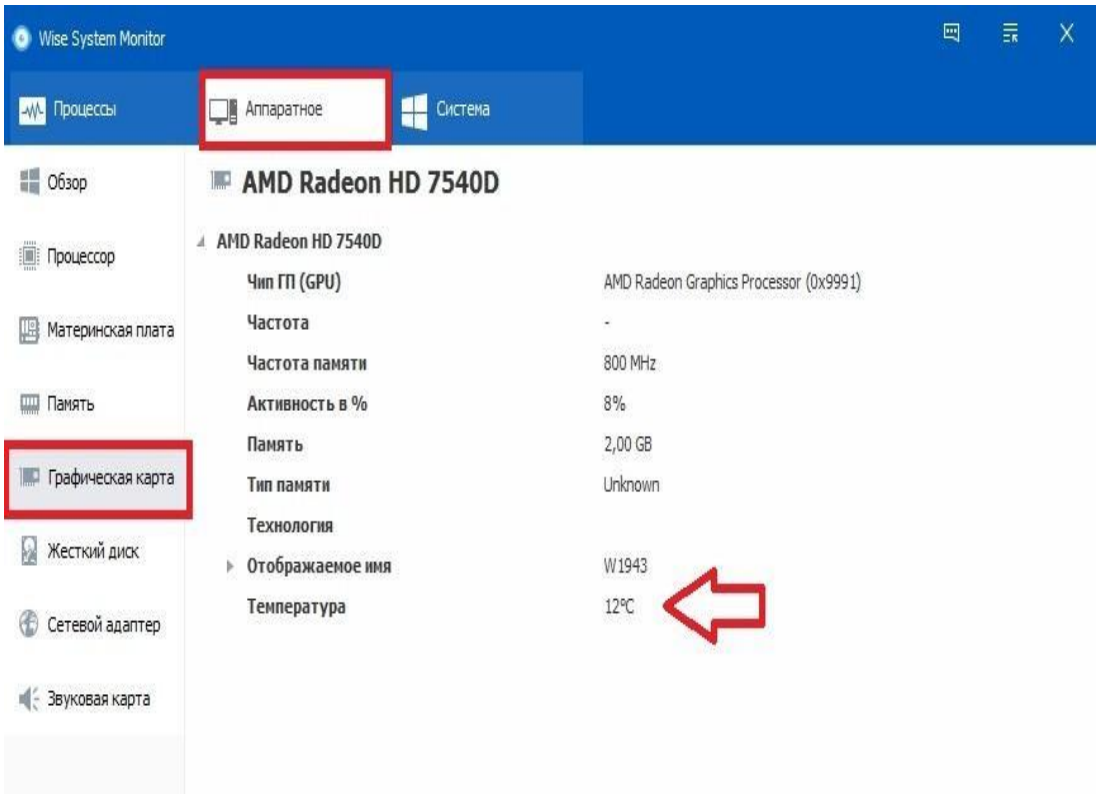


Рисунок 1.9- Вкладка «Графическая карта»

З прикладу видно, що утиліта має зручний і красивий інтерфейс і надає користувачеві ПК максимум інформації. Крім цього, в програму Wise System Monitor вбудований центр підтримки, де можна задати питання кусючою роботи утиліти і інформації, яку вона видає.

Серед недоліків же можна відзначити застарілий та відносно простий графічний інтерфейс, відсутність мобільного додатку та незначну стійкість до навантажень мережевої системи в цілому.

1.3 Постановка задачі

Проектування та створення системи контролю температури є досить складною та відповідальною задачею. Адже ряд функцій, які покладаються на такі системи, як правило, відповідають за своєчасне звернення уваги на перегрів та усунення проблем чи помилок.

Основною задачею, що постане перед даною утилітою, буде оптимізація функціоналу та робочого процесу, що існують, та додання необхідного функціоналу для зчитування й коректної обробки даних, що передається зчитувачем.

Серверна частина додатку має бути розроблена з використанням мови програмування C# та з використанням API – Nest.js, TypeORM, та база даних - MongoDB. Похідний код Windows-сервісу має бути розроблений з використанням мови програмування C# із використанням бібліотек платформи .NET. Установник служби має генеруватися динамічно через використання служби Devenv, що входить до комплекту Visual Studio 2015.

Web-частина має бути створена завдяки фреймворку IONIC4 [5], для гібридної розробки під мобільні платформи - Android, IOS. Надмножиною мови програмування JavaScript, TypeScript, Angular. З використанням зборщику пакетів Webpack 2, менеджеру пакетів мови NodeJS NPM, препроцессору SASS та із використанням фреймворка Bootstrap.

До основного набору функцій, котрі надаватиме розроблювана система, входять:

- передача інформації про ім'я комп'ютера і моделі - загальні дані;
- передача інформації про напруги;
- передача даних про процесор, його ядрах, температурі, частоті процесора;
- передача даних про електроживленні комп'ютера;
- передача інформації що до використання ресурсів;
- передача інформації про жорсткі диски;
- інформування про перевищення верхнього або нижнього порога температур звуковим сигналом, текстовим повідомленням, шляхом запуску зовнішньої програми;
- збереження результатів моніторингу відслідковуються параметрів в лог-файл;
- управління швідкістю обертання встановлених кулерів

Перш за все треба зрозуміти як обробляти дані, які ми отримуємо з комп’ютера, тому що методів обробки даних дуже багато. Планується що інформація про температуру комплектуючих буду конвертуватису у JSON-файл, також треба зробити графіки з цих даних[7].

Тому можно припустити що данні будуть великі, саме для цього навіть існує термін Big Data - це дані величезних обсягів, обробка та аналіз яких вимагає підходів, інструментів і методів, які істотно відрізняються від класичних.

2 ПЕРЕЛІК ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

2.1 Концепт програмної системи

Програмна система повина відповідати наступним вимогам щодо технологій:

– програма для персональних комп’ютерів під управлінням ОС Windows;

– передача даних з програми до API;

– WEB-сайт;

– мобільний додаток на Android;

– мобільний додаток на IOS.

Основна ідея у тому, щоб при відкриті програми та отриманні даних з зчитувачів, обробляти дані та відправляти їх до API(Nest.js). Звідки вже буде потрапляти до бази даних(MongoDB). Та для зручності користувача був розроблений веб додаток та мобільний, завдяки фраймворку IONIC4, куди будуть потрапляти дані з бази даних.

2.2 Функціональні вимоги

Функціональні вимоги описують сервіси, що надаються програмної системою, її поведінку в певних ситуаціях, реакцію на ті чи інші вхідні дані і дії, які система дозволить виконувати користувачам. Іноді сюди додаються відомості про те, чого система робити не повинна.

Під час опису вимог, розділимо їх на наступні категорії:

- система зобов’язана мати;
- повинна мати;
- могла б мати;
- бажано б мати.

Почнемо із вимог, що утиліта зобов’язана мати. Перш за все має бути відображені дані про комплектуючі, тобто температура. Мінімальна та максимальна температура для окремих деталій комп’ютера. Якщо температура перевищує показники допустимого, відправляти повідомлення користувачу. Роспізнавати різні моделі комплектуючих.

Наступним роздивимося, що система могла б мати. Запитувати користувача про дозвіл на розповсюдження повідомлень на усі його пристрої, тобто зробити мобільний додаток до цієї утиліти. Якщо користувач дав згоду на отримання повідомлень, система автоматично має зберегти його пристрій як пристрій на який буде приходити повідомлення.

Утиліта має дати можливість користувачеві отримувати дані у вигляді таблиць чи у вигляді списку для мобільного додатку.

Утиліта має дати можливість користувачеві регулювати допустимі норми максимальної температури та рекомендуємі для певного пристрою.

Зазначимо бажані функції системи.

Найбільш бажаним функціоналом є отримання звітності у вигляді діаграм та графіків. Особливо це стосується перегляду історій взаємодій для користувача та історії взаємодії для аудиторій.

Наступною бажаною звітністю є створення JSON таблиць із даними, що зберігаються в базі даних[7].

Веб-додаток має бути конфігурований через зборщик пакетів Webpack 2 та має на меті декларування усіх необхідних бібліотек та шляхів до директорій із результуючими файлами, що будуть направлені до браузеру.

Окрім Google авторизації, бажано мати інші соціальні мережі, насамперед це Facebook, Twitter, LinkedIn та GitHub.

2.3 Нефункціональні вимоги

Як для визначення функціональних, так і для визначення функціональних вимог використовуються робочі групи, члени яких визначають, перевіряють і стверджують вимоги. Для груп по визначенню не функціональних вимог особливо важливо залучити до цієї роботи не тільки аналітиків і користувачів, а й архітекторів і ключових розробників продукту або системи, а також групу тестування. Архітектор сприймає нефункціональні вимоги як вхідні дані для вибору і проектування архітектури додатку, а група тестування планує ті сценарії навантажувального тестування, які будуть використовуватися для перевірки виконання функціональних вимог (в основному це стосується атрибутів якості).

Всі атрибути якості з точки зору архітектури системи можна розділити на дві великі групи: перша група (runtime) - це атрибути, які відносяться до роботи програми або системи; друга група (design time) визначає ключові аспекти проектування програми або системи. Багато з цих атрибутів взаємозалежні[8].

До групи runtime відносяться наступні атрибути якості:

– доступність – серверна частина має працювати постійно, незалежно від завантаженості. Windows-сервіс починає працювати на локальній ЕОМ або сервері автоматично одразу після запуску. Мобільний додаток має працювати виключно при безпосередньому використанні;

– надійність – має бути дані що збережені у текстовому файлі для порівнювання роботи комплектуючих.

– вимоги – усі дані мають бути збережені у базі даних MongoDB та бути винесені до міграцій. Час зберігання необмежений. Дані, що зберігаються у пам’яті зчитувачів, мають бути видалені одразу ж після синхронізації із сервером. Дані, що зберігаються у NoSQL базі даних, мають бути знищені через одну годину після створення.

– масштабованість – система має підтримувати як горизонтальну масштабованість через створення нових кластерів із додатком, так і вертикальну, що має підсилити її продуктивність.

– зручність використання – інтерфейс користувача повинен бути збудований так, щоб він мав підтримку з боку сучасних браузерів, а саме: Internet Explorer 11, Edge, Mozilla Firefox, Google Chrome, Safari, Opera. Інтерфейс користувача має бути адаптивним, тобто бути зручним у використанні через мобільні пристрої, планшети та комп'ютери.

– вимоги до безпеки – дані, що передаються від контролерів до серверу та від серверу до контролерів повинні бути зашифровані через алгоритм шифрування AES. Під час обробки кожного запиту до серверу, запитувач має бути авторизований у системі через використання унікального токену, що має зберигатися у Redis. У разі відсутності токену у сховищі, користувач отримує помилку 401 Unauthorized. У випадках, коли користувач надсилає запит на дані, до яких він не має доступу, то користувач отримує помилку 403 Forbidden[9].

– вимоги до конфігурування – серверна частина повинна мати файл application.properties, де мають знаходитися рядки із конфігурацією доступу до бази даних, налаштування міграцій, сервісу OAuth 2.

Для Windows сервісу конфігурація повинна бути вбудована до виконуючого файлу та декларувати необхідні шляхи до з'єднань із зчитувачами.

Веб-додаток має бути конфігурований через зборщик пакетів Webpack 2 та має на меті декларування усіх необхідних бібліотек та шляхів до директорій із результуючими файлами, що будуть направлені до браузеру.

Середовище розгортання серверної частини має підтримувати мову програмування JS, та мати ОС Windows із встановленою програмою Visual Studio 2015. У системі має бути установлений IONIC4, сервер NodeJS(NestJS) бази даних MongoDB. В свою чергу, веб додаток не має підтримувати Internet Explorer до 11 версії [10].

Під час створення інсталятору сервісу усі інші запити мають чекати доки процесс не буде виконано.

До групи design time належать наступні атрибути якості:

– вимоги до повторного використання компонентів системи – повторне використання компонентів має бути використано у веб-додатку через створення шаблонів зі скриптами, що мають бути перевикористані через впровадження вхідних даних до них.

– вимоги до переносимості – серверна частина має однаково працювати на платформах Windows 7, Windows 8, Windows 8.1 та Windows 10, Windows Server 2012 та Windows Server 2016. Не допускається розгортання серверу у Unix-подібному середовищі. Мобільний додаток має підтримувати усі версії Android, починаючи із версії 4.0.0.

– вимоги до можливості та простоти локлізації – система має підтримувати лише англійську мову для інтерфейсу користувача. Інформація, що зберігається у базі даних не обмежена у мові.

– вимоги до можливості тестування – інтерфейс користувача має бути протестований через впровадження системи автоматичного аналізу якості Selenium. Окрім цього він має бути перевірений методом «мануального» тесутвання.

3 ОПИС ДОСЛІДЖЕННЯ ТА ПРИЙНЯТИХ ПРОЕКТНИХ РІШЕНЬ

3.1 Дослідження метода обробки даних

Перш за все треба розглянути що таке Big Data - це серія підходів, інструментів і методів обробки структурованих, слабо структурованих і неструктурованих даних величезних обсягів і значного різноманіття для отримання сприймаються людиною результатів, ефективних в умовах безперервного приросту, розподілу по численних вузлів обчислювальної мережі, альтернативних традиційним системам управління базами даних і рішень класу Business Intelligence . В дану серію включають кошти масово-паралельної обробки невизначено структурованих даних, перш за все, рішеннями категорії NoSQL, алгоритмами MapReduce, програмними каркасами і бібліотеками проекту Hadoop[18].

В якості визначальних характеристик для великих даних відзначають «три V» - volume (обсяг), velocity (швидкість), variety (різноманіття):

– обсяг - в сенсі величини фізичного обсягу,

– швидкість - в сенсах як швидкості приросту, так і необхідності високошвидкісної обробки і отримання результатів,

– різноманіття - в сенсі можливості одночасної обробки різних типів структурованих і неструктурованих даних

Отже ми зрозуміли що таке великі дані та для чого вони потрібні. Тому що наша база даних працює саме так(MongoDB - NoSQL). Тепер розглянемо методи:

– Data Mining: навчання асоціативним правилами, класифікація (методи категоризації нових даних на основі принципів, раніше застосованих до вже наявним даними), кластерний аналіз, регресійний аналіз.

– краудсорсінг - категоризація та збагачення даних силами широкого, невизначеного кола осіб.

– змішання і інтеграція даних - набір технік, що дозволяють інтегрувати різнорідні дані з різноманітних джерел для можливості глибинного аналізу.

– машинне навчання, включаючи навчання з учителем і без учителя, а також використання моделей, побудованих на базі статистичного аналізу або машинного навчання для отримання комплексних прогнозів на основі базових моделей.

– штучні нейронні мережі, мережевий аналіз, оптимізація, в тому числі генетичні алгоритми.

– просторовий аналіз - використання топологічної, геометричній і географічної інформації в даних.

– статистичний аналіз: A / B-тестування і аналіз часових рядів.

– візуалізація аналітичних даних - подання інформації у вигляді малюнків, графіків, схем і діаграм з використанням інтерактивних можливостей та анімації як для результатів, так і для використання в якості вихідних даних для подальшого аналізу.

Наша база даних працює за принципом ключ значення, також після перерахування методів обробки даних в Big Data, зрозуміло що нам потрібен метод Data Mining, а само дерево рішень (decision trees). Тому дані що ми отримуємо з комп’ютера нам потрібні для аналізу, коли температура була підвищена та коли потрібно відіслати повідомлення для клієнта о підвищеній температурі будь якого з комплектуючих.

Також треба виділити проблеми роботи з великими даними, а саме складність обробки великих даних та візуалізація їх:

– обмежена доступність – існуючі альтернативні програмні забезпечення обмежують вхідні параметри запиту, тобто не дозволяють шукати по всім странам.

– обмежена швидкодія – великі обсяги даних, які необхідні для аналізу результатів, є причиною високих витрат по швидкодії.

– відсутність аналогів, які візуалізують отримані результати – так як результат роботи подається в табличному вигляді, є причиною того, що користувачеві важко працювати з цими даними в подальшому. Складність обробки потоку великих даних та візуалізації їх, у свою чергу, породжує наступний ряд проблем:

– відсутність бази даних – громіздка база даних та важкість у підтримуванні бази, є причиною того, що дані зберігаються у файлах, що призводить до складності формування та обробки запиту.

– недостатня швидкодія – розголудженність вхідних даних та відсутність логіки при формуванні запиту збільшують час обробки запиту.

До найбільш поширених підходів обробки даних відносяться такі інструменти:

– SQL – мова структурованих запитів, що дозволяє працювати з базами даних. За допомогою SQL можна створювати і модифікувати дані, а управлінням масиву даних займається відповідна система управління базами даних.

– NoSQL – термін розшифровується як Not Only SQL (не тільки SQL). Включає в себе ряд підходів, спрямованих на реалізацію бази даних, що мають відмінності від моделей, що використовуються в традиційних, реляційних СУБД. Їх зручно використовувати при постійно мінливій структурі даних. Наприклад, для збору і зберігання інформації в соціальних мережах.

– MapReduce – модель розподілу обчислень. Використовується для паралельних обчислень над дуже великими наборами даних (петабайт * і більше). У програмному інтерфейс не дані передаються на обробку програмі, а програма – даними. Таким чином запит є окремою програмою. Принцип роботи полягає в послідовній обробці даних двома методами Map і Reduce. Map вибирає попередні дані, Reduce агрегує їх.

– Hadoop – використовується для реалізації пошукових і тематичних механізмів високонавантажених сайтів – Facebook, eBay, Amazon і ін. Відмінною особливістю є те, що система захищена від виходу з ладу будь-якого з вузлів кластера, так як кожен блок має, як мінімум, одну копію даних на іншому вузлі.

– SAP HANA – високопродуктивна NewSQL платформа для зберігання і обробки даних. Забезпечує високу швидкість обробки запитів. Ще одним відмітним ознакою є те, що SAP HANA спрощує системний ландшафт, зменшуючи витрати на підтримку аналітичних систем

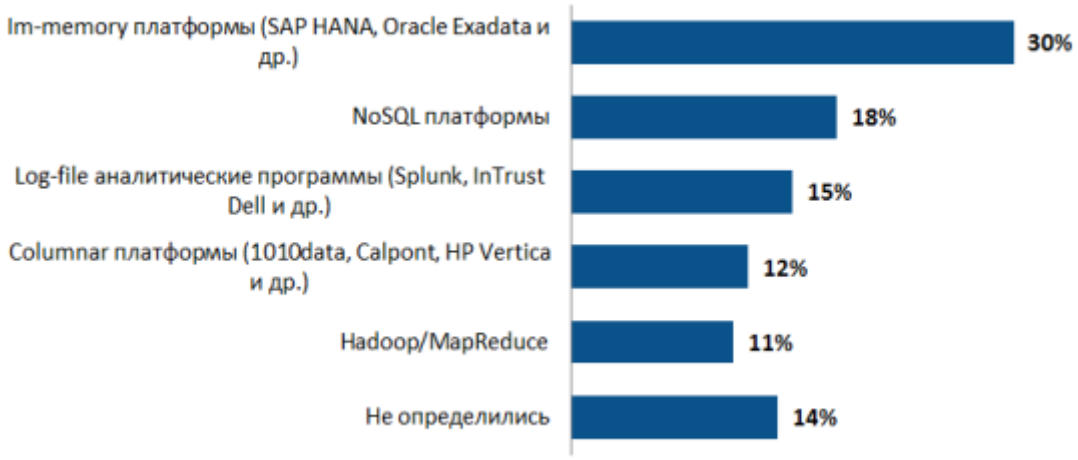


Рисунок 3.1 – Основні технології для роботи з BigData

Обробка великих даних завжди було непростою задачею, існує велика кількість підходів та засобів для опрацювання великих даних. Вибір остаточно способу має грунтуватися на кількості даних, кінцевої мети, та бюджету.

Існує безліч комбінацій програмного і апаратного забезпечення, які дозволяють створювати ефективні рішення Big Data для різних бізнес дисциплін: від соціальних медіа та мобільних додатків, до інтелектуального аналізу і візуалізації комерційних даних. Важливе значення Big Data – це сумісність нових інструментів з широко використовуваними в бізнесі базами даних, що особливо важливо при роботі з крос-дисциплінарними проектами.

Послідовність роботи з Big Data складається з збору даних, структурування отриманої інформації за допомогою звітів і аналітика (dashboard). Так як робота з Big Data має на увазі великі витрати на збір даних, результат обробки яких заздалегідь невідомий, основним завданням є чітке розуміння, для чого потрібні дані, а не те, як багато їх є в наявності. В цьому випадку збір даних перетворюється в процес отримання виключно потрібної для вирішення конкретних завдань інформації.

Big Data зазвичай зберігаються і організовуються в розподілених файлових системах. У загальних рисах, інформація зберігається на декількох (іноді тисячах) жорстких дисках, на стандартних комп'ютерах. Так звана «карта» (map) відстежує, де (на якому комп'ютері і / або диску) зберігається конкретна частина інформації. Для забезпечення відмовостійкості та надійності, кожну частину інформації зазвичай зберігають кілька разів, наприклад – тричі. За допомогою стандартного устаткування і відкритих програмних засобів для управління цією розподіленою файловою системою наприклад, Hadoop, порівняно легко можна реалізувати надійні сховища даних в масштабі петабайт.

Найпростіший спосіб запустити MapReduce-програму на Hadoop – скористатися streaming-інтерфейсом Hadoop. Streaming-інтерфейс передбачає, що map і reduce реалізовані у вигляді програм, які приймають дані з stdin і видають результат на stdout. MapReduce – це модель розподіленої обробки даних, запропонована компанією Google для обробки великих обсягів даних на комп'ютерних кластерах.

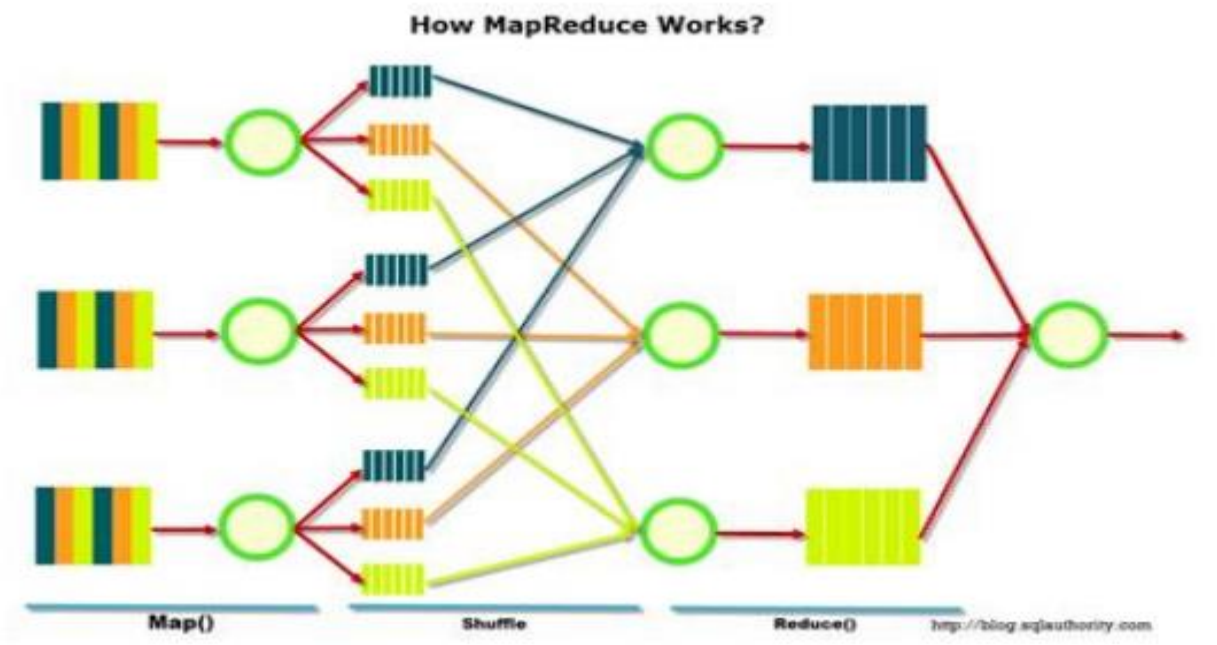


Рисунок 3.2 – Принцип роботи з MapReduce

Обробка даних відбувається в 3 стадії:

Перша стадія – стадія Map. На цій стадії дані оброблюються за допомогою функції map(), яку визначає користувач. Робота цієї стадії полягає в передобробці і фільтрації даних. Робота дуже схожа на операцію map в функціональних мовах програмування – призначена для користувача функція застосовується до кожної вхідного відрізку. Функція map () застосована до однієї вхідного відрізку і видає безліч пар ключ-значення.

Друга стадія – стадія Shuffle. Проходить непомітно для користувача. У цій стадії висновок функції map «розбирається по кошиках» – кожний кошик відповідає одному ключу виведення стадії map. Надалі ці кошики послужать входом для reduce.

Третя стадія – стадія Reduce. Кожний «кошик» із значеннями, сформований на стадії shuffle, потрапляє на вхід функції reduce (). Але не менш важливий вибір сервісу або бази даних для сбереження великої кількості даних. На сьогодні існує багато хмарних рішень для збереження та опрацювання великої кількості даних. Кожне рішення має свої сервіси систем керування базами даних. Так серед усіх існуючих варто виділити сервіси Amazon, оскільки вони є лідерами на ринку.

Підхід map-reduce можна використовувати в основних базах даних, які підтримують багатокластерність.

Вже було вирішено, що ми використовуємо метод обробки даних в Big Data – Data Mining, а саме дерево рішень (decision trees). Дерева рішень є одним з найбільш ефективних інструментів інтелектуального аналізу даних і самий корінь аналітики, які дозволяють вирішувати завдання класифікації і регресії.

Вони являють собою ієрархічні деревоподібні структури, що складаються з вирішальних правил виду «Якщо ..., то ...». Правила автоматично генеруються в процесі навчання на навчальній множині і, оскільки вони формуються практично на природній мові (наприклад, «Якщо температура відеокарти 70 градусів тоді – це погано»), дерева рішень як аналітичні моделі більш вербалізуемой і інтерпретованих, ніж, скажімо, нейронні сіті.

Оскільки правила в деревах рішень виходять шляхом узагальнення безлічі окремих спостережень (навчальних прикладів), що описують предметну область, то за аналогією з відповідним методом логічного висновку їх називають індуктивними правилами, а сам процес навчання - індукцією дерев рішень.

Власне, саме дерево рішень - це спосіб подачі вирішальних правил в ієрархічній структурі, що складається з елементів двох типів - вузлів (node) і листя (leaf). У вузлах знаходяться вирішальні правила і проводиться перевірка відповідності прикладів цього правила з якого-небудь атрибуту навчальної множини.

У найпростішому випадку, в результаті перевірки, безліч прикладів, які потрапили в вузол, розбивається на дві підмножини, в одне з яких потрапляють приклади, що задовольняють правилу, а в інше - що не задовольняють.

Потім до кожного подмножеству знову застосовується правило і процедура рекурсивно повторюється поки не буде досягнуто деяке умова зупинки алгоритму. В результаті в останньому вузлі перевірка і розбиття не проводиться і він оголошується листом. Лист визначає рішення для кожного потрапив в нього

приклад. Для дерева класифікації - це клас, що асоціюється з вузлом, а для дерева регресії - відповідний листу модальний інтервал цільової змінної.

Таким чином, на відміну від вузла, в листі міститься не правило, а підмножина об'єктів, які відповідають всім правилам гілки, яка закінчується даними листом.

Очевидно, щоб потрапити в лист, приклад повинен відповідати всім правилам, які лежать на шляху до цього листа. Оскільки шлях в дереві до кожного листу єдиний, то й кожен приклад може потрапити тільки в один лист, що забезпечує єдність розв'язку.

Основна сфера застосування дерев рішень - підтримка процесів прийняття управлінських рішень, використовувана в статистиці, аналізі даних і машинному навчанні. Завданнями, які розв'язуються за допомогою даного апарату, є:

– класифікація - віднесення об'єктів до одного з заздалегідь відомих класів. Цільова змінна повинна мати дискретні значення.

– регресія (чисельне пророкування) - прогноз числового значення незалежної змінної для заданого вхідного вектора.

– опис об'єктів - набір правил в дереві рішень дозволяє компактно описувати об'єкти. Тому замість складних структур, що описують об'єкти, можна зберігати дерева рішень.

Процес побудови дерев рішень полягає в послідовному, рекурсивном розбитті навчальної множини на підмножини з застосуванням вирішальних правил в вузлах. Процес розбиття триває до тих пір, поки всі вузли в кінці всіх гілок НЕ будуть оголошені листям. Оголошення вузла листом може статися природним чином (коли він буде містити єдиний об'єкт, або об'єкти тільки одного класу), або після досягнення деякого умови зупинки, що задається користувачем (наприклад, мінімально допустиму кількість прикладів в вузлі або максимальна глибина дерева).

Алгоритми побудови дерев рішень відносять до категорії так званих жодних алгоритмів. Жадібними називаються алгоритми, які допускають, що локально-оптимальні рішення на кожному кроці (розбиття в вузлах), призводять до

оптимального підсумкового рішення. У разі дерев рішень це означає, що якщо один раз був обраний атрибут, і по ньому було вироблено розбиття на підмножини, то алгоритм не може повернутися назад і вибрати інший атрибут, який дав би краще підсумкове розбиття. Тому на етапі побудови можна сказати забезпечить обраний атрибут, в кінцевому підсумку, оптимальне розбиття.

В основі більшості популярних алгоритмів навчання дерев рішень лежить принцип «розділяй і володарюй». Алгоритмічно цей принцип реалізується в такий спосіб. Нехай задано навчальну множину S, що містить n прикладів, для кожного з яких задана мітка класу $C_i(i = 1..k)$, и m атрибутов $A_j(j = 1..m)$

які, як передбачається, визначають приналежність об'єкта до того чи іншого класу.

Основні етапи побудови. В ході побудови дерева рішень потрібно вирішити кілька основних проблем, з кожної з яких пов'язаний відповідний крок процесу навчання:

– вибір атрибута, за яким буде проводитися розбиття в даному вузлі (атрибута розбиття).
– вибір критерію зупинки навчання.
– вибір методу відсікання гілок (спрощення).
– оцінка точності побудованого дерева.

3.2 Аспекти аналітичної обробки даних

Для оптимізації роботи з великими даними варто провести аналітичну роботу над тим, як можна оптимізувати дані для зменшення затратності їх опрацювання. Тому далі буде висвітлено основні проблеми, які доводиться вирішувати при створенні аналітичних систем на прикладі публічні закупівлі України, зокрема вилучення та обробка денормалізованих даних з існуючих сховищ. Також дослідження можливості аналізу та обробки даних. При цьому варто відслідкувати

основні проблеми побудови аналітичної системи тендерних закупівель та розглянути підходи до їх розв’язання, зокрема шляхи нормалізації текстових даних.

У MongoDB застосовується бессхемная організація неструктурованих або слабоструктурованих даних, тобто організація, яка не потребує опису схеми бази даних. Цей спосіб організації даних не призначений для того, щоб пов'язувати документи з даними однієї колекції з документами, що містять дані, інший колекції через ключові поля. У MongoDB немає власних коштів створення відносин між колекціями і документами.

Документоорієнтованих СУБД MongoDB призначена для зберігання даних в документах колекцій в форматі JSON (як звичайний текст), які не мають чіткої структури та призначені для зберігання будь-якого типу даних. Такий формат і організація зберігання даних в СУБД MongoDB забезпечує оперативну обробку даних, їх запис і виведення результатів. Таким чином, MongoDB відмінно підходить для роботи з нереляціоннимі даними.

Звідси випливає, що MongoDB добре працює з наборами даних (колекціями і документами), які не пов'язані між собою, і її можна використовувати для створення Web додатків, що забезпечують високу продуктивність і необмежену горизонтальне масштабування колекцій даних.

У процесі створення кожного документа (ObjectID) призначається "_id" унікальний ідентифікатор документа, який призначений для автоматичного сортування документів (рисунок 3.3). СУБД MongoDB призначена для роботи з нереляціоннимі даними, наприклад, для роботи з базою даних Website, яка може включати в себе наступні колекції (сутності): статті, новини, фото, архів і т.д.

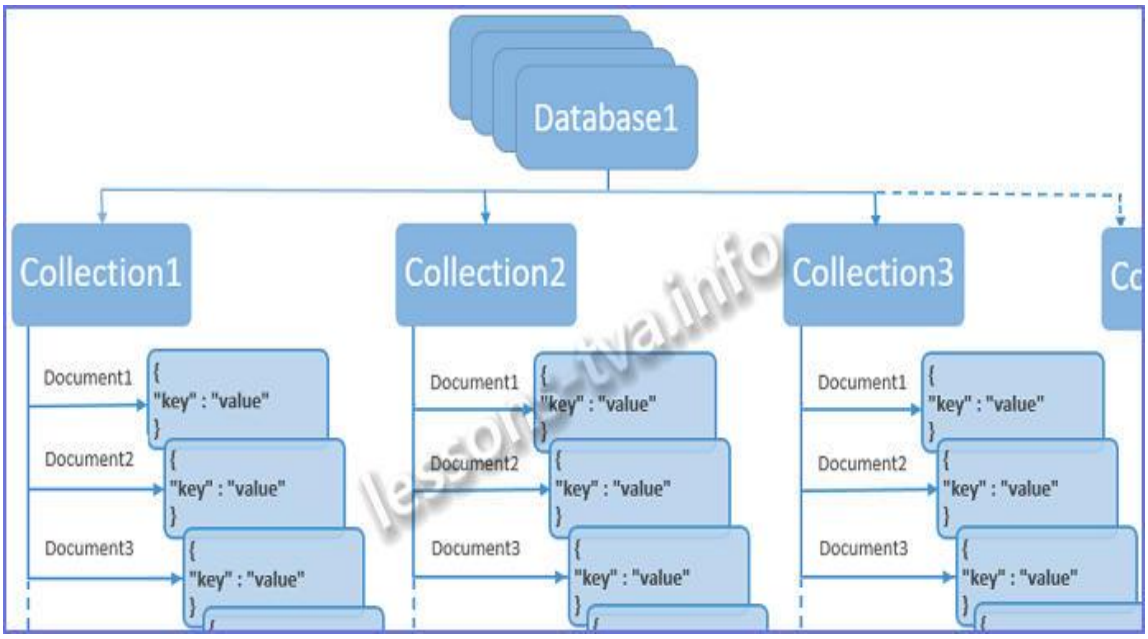


Рисунок 3.3 – Структура бази даних MongoDB

Нижче наведені основні переваги MongoDB:

– Відсутність схеми. Дана БД заснована на колекціях різних документів. Кількість полів, зміст і розмір цих документів може відрізнятися. Тобто різні сутності не повинні бути ідентичні за структурою.

– Вкрай зрозуміла структура кожного об'єкта.

– Легко масштабується

– Для зберігання використовуваних в даний момент даних використовується внутрішня пам'ять, що дозволяє отримувати більш швидкий доступ.

– Дані зберігаються у вигляді JSON документів

– MongoDB підтримує динамічні запити документів (document-based query)

– Відсутність складних JOIN запитів

– Немає необхідності маппінга об'єктів додатки в об'єкти БД

3.3 Дослідження швидкодії роботи різних баз даних

На сьогодні в усьому світі спостерігається великий інтерес до технологій класу Big Data, пов'язаний з постійним зростанням даних, якими доводиться оперувати великим компаніям. Важливим активом будь-якої компанії є накопичена інформація, проте з кожним днем стає все складніше і дорожче обробляти її і отримувати від неї користь.

Завдання полягає в порівнянні продуктивності систем управління баз даних, що спираються на SQL і NoSQL підходи для конкретного класу задач.

В ході вивчення теми, було виявлено, що порівняння продуктивності вже проводилися для MongoDB і PostgreSQL, але на відміну від нашої роботи це було порівняння на рівні самих систем, а не на рівні завдань. У цих порівняннях розглядали одні і ті ж функції для PostgreSQL і MongoDB, можливо, ігноруючи більш вдалі способи розташування інформації в Mongo і інші варіанти написання запитів. Варто також відзначити, що одне з досліджень проводилося розробниками PostgreSQL, тому є ймовірність того, що вони не були цілком об'єктивні.

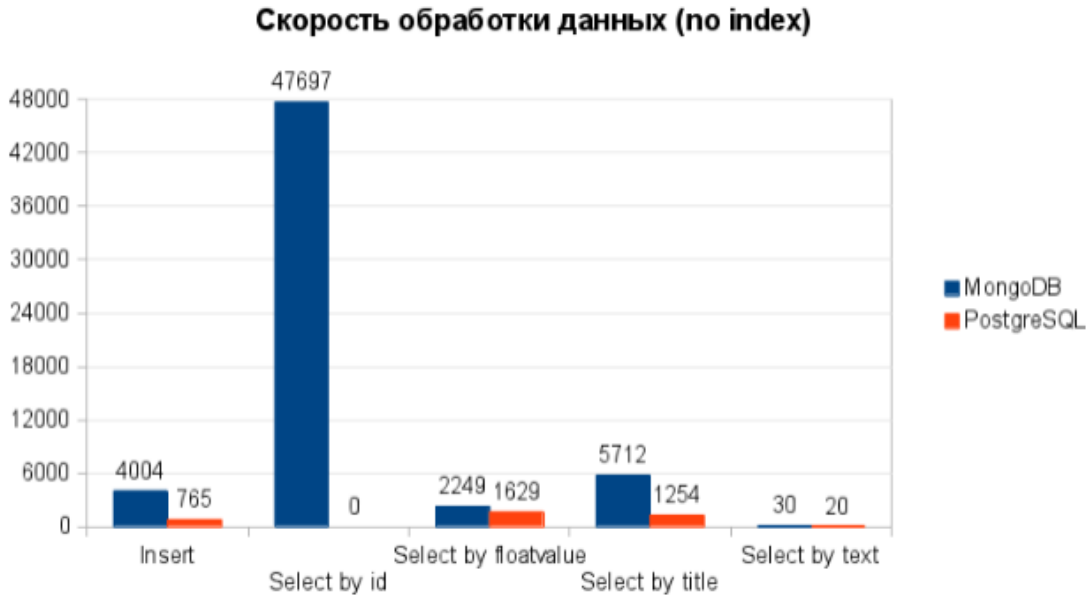


Рисунок 3.4 – Швидкість обробки даних

В ході виконання курсової роботи була розроблена схема бази даних, яка була реалізована у вигляді таблиць для PostgreSQL. У MongoDB були створені

колекції, в документах яких були розміщені дані для зберігання тієї ж самої інформації. Для більш ретельного дослідження були заповнені 3 варіанти бази для MongoDB, щоб врахувати ймовірність того, що на різних схемах якісь запити будуть давати кращі результати. Відзначимо також, що порівняння продуктивності виконувалося для баз даних розмірами 812 Мб, 1.5 Гб і 2 Гб.

Після створення таблиць в PostgreSQL потрібно згенерувати і завантажити в них дані. Для генерації даних використовувався Faker gem. Невелике додаток, написане на Ruby завантажувати дані в файли з розширенням sql, які потім завантажувалися в базу. Для вивантаження в базу використовувалася команда в консолі. **psql -h localhost -U postgres -d study -f 'C:/filepath/file.sql'**

Ця команда підключається до бази study і виконує скрипт file.sql. В скрипте написані команди для вставки.

INSERT INTO table VALUES(),(),()...();

Аналогічним чином створюються дані і заповнюються колекції в MongoDB. Але на відміну від, тут нам потрібно згенерувати дані тільки для 3х, 2х або 1 колекції. А так як MongoDB надає нам можливість вкладення одних документів в інші, то в цілому ми вставляємо меншу кількість записів, хоча і більшого обсягу.

Згенеровані дані зберігаються в документ формату .js. В ньому прописані рядки в форматі.

db.collection.insert([{},{},{},...{}]).

Слід зазначити, що для заповнення бази в MongoDB ми не повинні вручну створювати колекції. Якщо вони не створені, то вони створюються при виклику db.collection.insert ().

load('C:/filepath/file.js');

Спочатку був вибір між тим, завантажувати дані з файлу, в якому за одну команду Insert відбувається вставка n записів або завантажувати дані з файлу, в якому на n записів пріод n Insert. Практика показала, що перший варіант працює в рази швидше.

В рамках поставленої задачі були придумані і реалізовані запити, що вимагають перегляду всієї бази даних. Заміри проходили на "Гарячої" базі, так як перевірка ефективності на таких запитах менш трудомістка, ніж на "холодної", яка вимагає перезапуску СУБД після кожного разу, і більш наближена до життя.

```
gr_ids = db.student.aggregate(
  {$unwind: "$student.results"},
  {$group: {
        _id: {id: "$_id",
        gr_id: "$student.groups_id"},
        mark: {$avg: "$student.results.mark"}}},
  {$match:  {mark: {$gt: 4}}},
  {$group: {_id: "$_id.gr_id",
          students: {$sum: 1}}}).
  result.map(function(d){
    return parseInt(d._id)});

db.groups.find({_id: {$in: gr_ids}})
```

Рисунок 3.5 – Запит для MongoDB бази даних з трьох колекцій

```
SELECT group_id,count(student_id) FROM
 (SELECT student.group_id,student.student_id, AVG(mark)
  FROM
    student, studygroup, student_result
  WHERE
    student.student_id = student_result.student_id and
    student.group_id = studygroup.group_id
    GROUP BY student.group_id, student.student_id
   ) AS "p"
WHERE
 p.AVG > 4
GROUP BY group_id
```

Рисунок 3.6 – Запит для PostgreSQL бази даних

Для MongoDB розглядалося кілька варіантів виконання деяких запитів: через aggregation framework або через MapReduce. Вибирався варіант, який давав кращі результати. Так як для виконання запитів в MongoDB ми попередньо Ніяк не відсівали інфомарцію, тобто в aggregation framework запитах не використовувати функції $ project, $ limit, $ skip, то і індексування нам ніяк не знадобилося.

3.4 Результати дослідження

Як згадувалося вище, результати вимірювань отримані для базданих розмірами 812 Мб, 1.5 Гб і 2 Гб. Всього було вироблено по 10 вимірювань для кожного запиту. Вимірювання проводились на "гарячій" базі даних, так як для забезпечення вимірювань на "холодної" базі доводилося б після кожного виміру перезапускати сервер. На діаграмах postgresql - база даних PostgreSQL, mongodb 1

- база даних MongoDB з трьома колекціями, mongodb 1 - база даних MongoDB з двома колекціями, mongodb 1 - база даних MongoDB з однією колекцією(рис. 3.6)

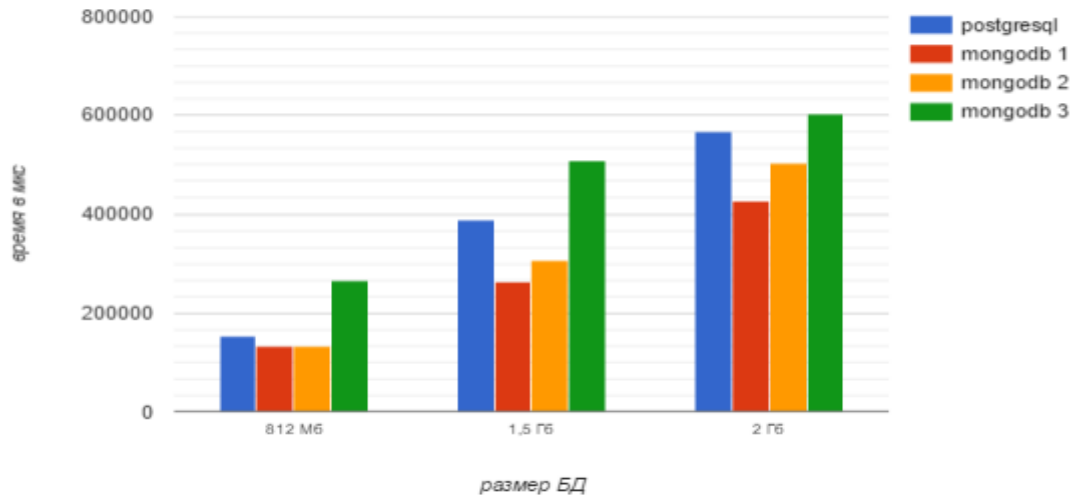


Рисунок 3.7 – Порівняння часу вставки

Для всіх запитів використовували технолгія MapReduce, яка, можливо, трохи програє aggregation в інших випадках. Найкраще в Mongo проявила себе база даних, в якій інформація була розподілена за трьома колекціям, що дозволило уникнути дублювання даних (крім зберігання масиву id пов'язаних об'єктів). В цілому, можливо, MongoDb програє PostgreSQL але тільки через особливостей побудови запитів і відсутності можливості в даних обставинах використовувати індексування.

Для кожного запиту отримана вибірка нормально-розподілених випадкових величин, для яких пораховано середньоквадратичне відхилення за формулою:

$$s = \sqrt{\frac{\sum_{i=1}^{n} (X - \bar{X})}{n}},$$

Де X - випадкова величина, $\bar{X}$ - середнє арифметичне вибірки, n - кількість вимірювань, s - середньоквадратичне відхилення. Значення всіх величини задовольняють правилу "трьох сигм", тобто лежать в інтервалі.

$$(\bar{x} - 3\sigma; \bar{x} + 3\sigma),$$

Де в якості сигми виступає s з формули. Виконання цього правила дає нам право використовувати отримані результати.

3.5 Проектування програмного продукту

Проектування програмного продукту було проведено з використанням діаграм UML. За допомогою цих діаграм було відображено роботу та функціонал системи (з доданням майбутньої функціональності). Було побудовано 9 діаграм, за допомогою яких було розглянуто систему з різних сторін.

Першою створеною діаграмою була діаграма варіантів використання (Use-Case). Основними акторами системи є гість та авторизований користувач. Гість може авторизуватися або зареєструватися у системі.

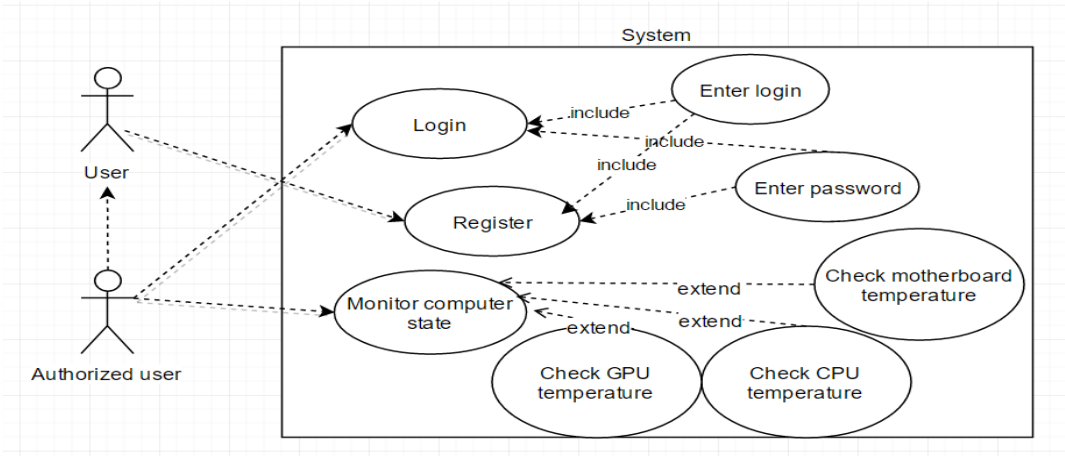
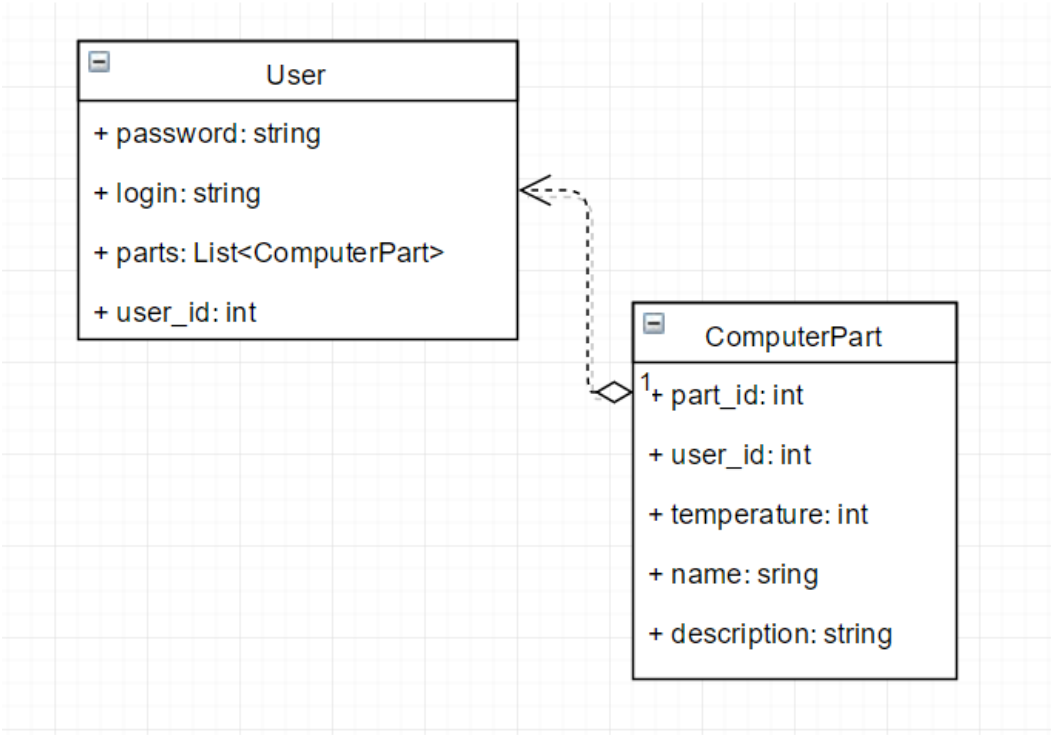


Рисунок 3.8 – Діаграма прецендентів

Рисунок 3.9 зображує діаграму класів. Ця діаграма демонструє відношення статичних, декларативних елементів моделі. Такий тип діаграм може застосовуватися як при розробці нової системи, так і при зворотньому

проектуванні. Дана діаграма є останнім кроком на фазі проектування і початком фази розробки. На даній діаграмі відображені основна моделі. Використані при розробці програмної системи.

На даній діаграмі наявні 2 основних елементи: користувач та датчик. Відношення між User та “Extended User” - один до одного. Від розширеної моделі користувача наслідується модель датчика, відношенням один до багатьох.Застосовуватися як при розробці нової системи, так і при зворотному проектуванні



.

Рисунок 3.9 – Діаграма класів

Наступною використаною схемою була діаграма станів. Вона демонструє процес переходу станів системи при використанні користувачем. На даній діаграмі зображено основний набір переходів для веб-версії системи. При використанні мобільного додатку, зміниться частина діаграми. Замість перегляду профілю буде

можливість додати розклад чи перевірити фотографію користувача на рівень втомленості. Як видно з діаграми, ключовий цикл додатку полягає в стадії після авторизації — користувач може працювати зі своїм профілем чи зі своїми столами, після чого він має обов'язково підтвердити внесені зміни.

Треба зауважити, що на діаграмі береться до уваги також можливість користувача в будь який момент припинити роботу з програмою.

Також, на даній діаграмі станів зображено обробку помилкового вводу даних користувачем, що приведе його до повторного вводу своїх логіну та пароля акаунта. Реєстрація не приведена, як стан. Оскільки ця діаграма валідна для вже зареєстрованих користувачів.

Зображення даної діаграми приведено на рисунку 3.10

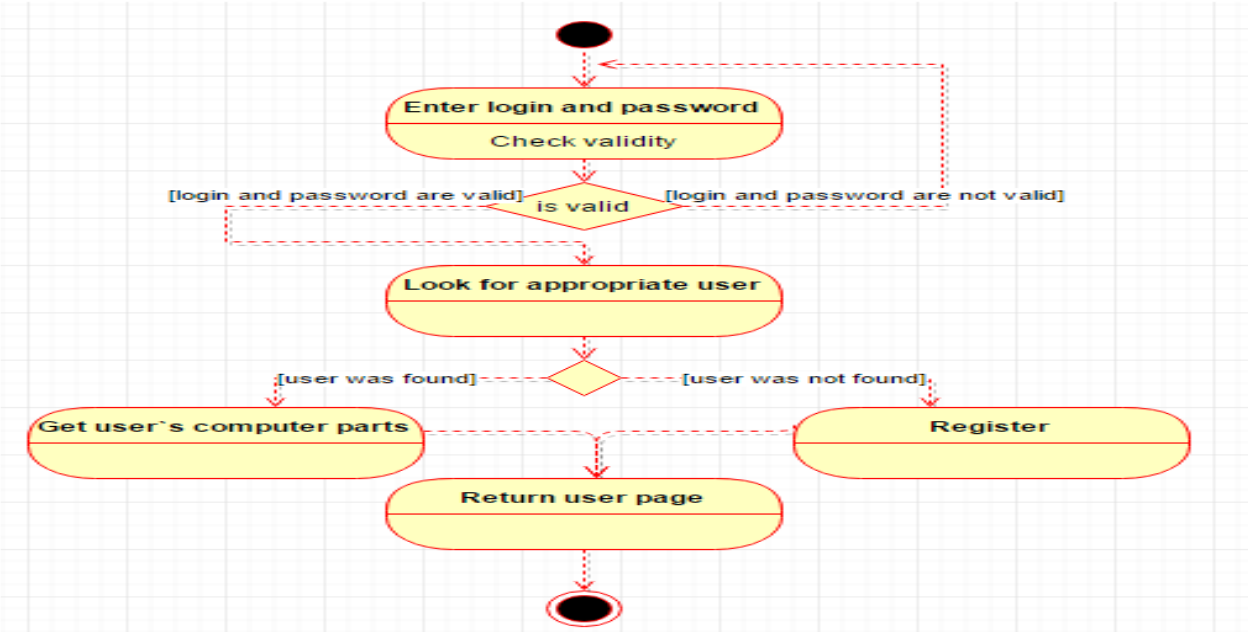


Рисунок 3.10 – Діаграма станів

Як видно з діаграми, вся взаємодія в сервісі проходить, власне, через API.

Загальна взаємодія об'єктами добре демонструється на прикладі діаграми послідовностей. На рисунку 3.11 зображено приклад даної діаграми для авторизації користувача.

Клієнт вводить свої дані на формі, а потім передає їх через модуль контролю, а потім, в разі успішного підтвердження, створюється нова сесія з авторизованим користувачем.

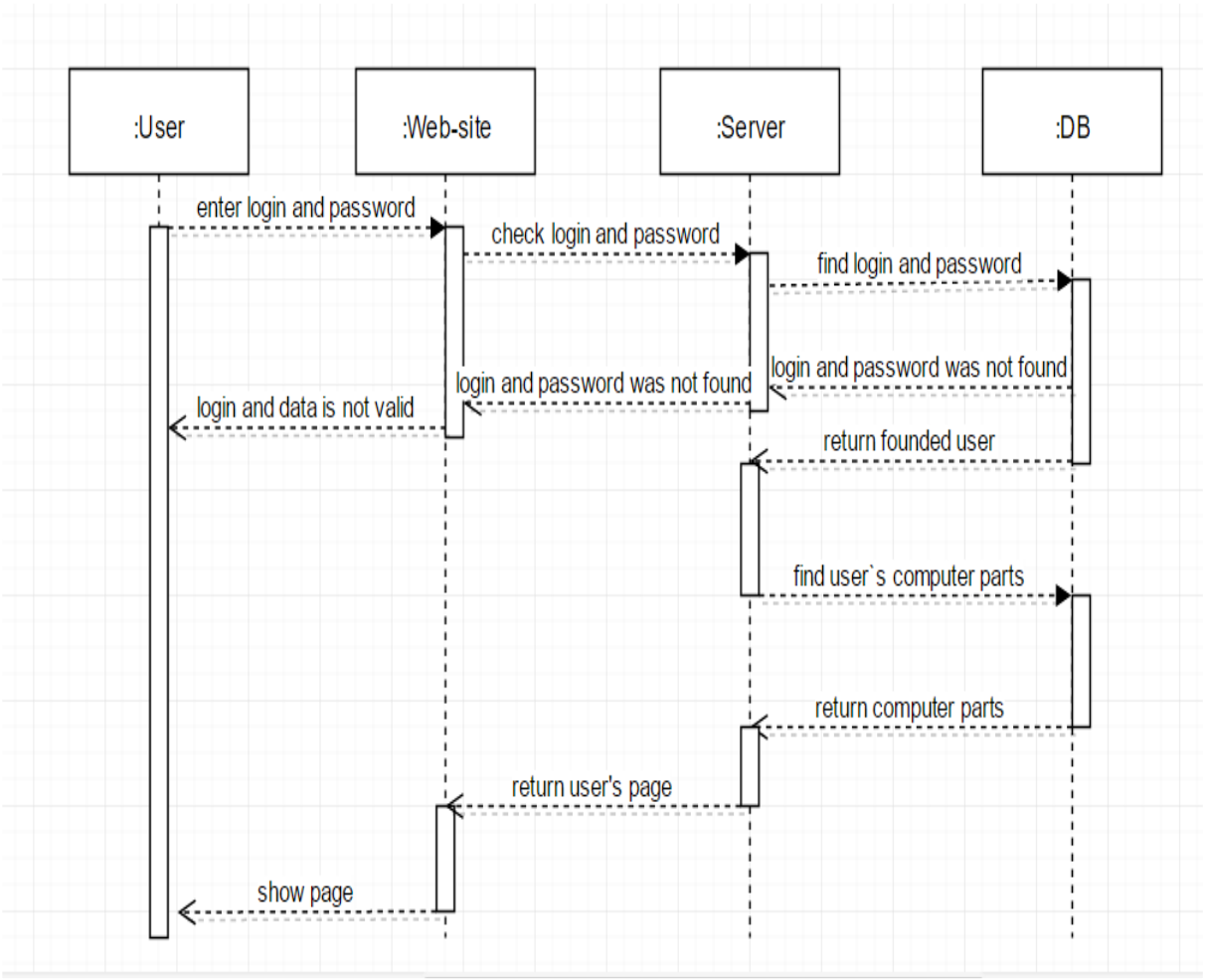


Рисунок 3.11 — Діаграма послідовностей

Після діаграми послідовностей варто зобразити діаграму кооперації, оскільки вони уявляють собою схожі схеми, як за семантикою, так і за структурою. (див. рис. 3.12)

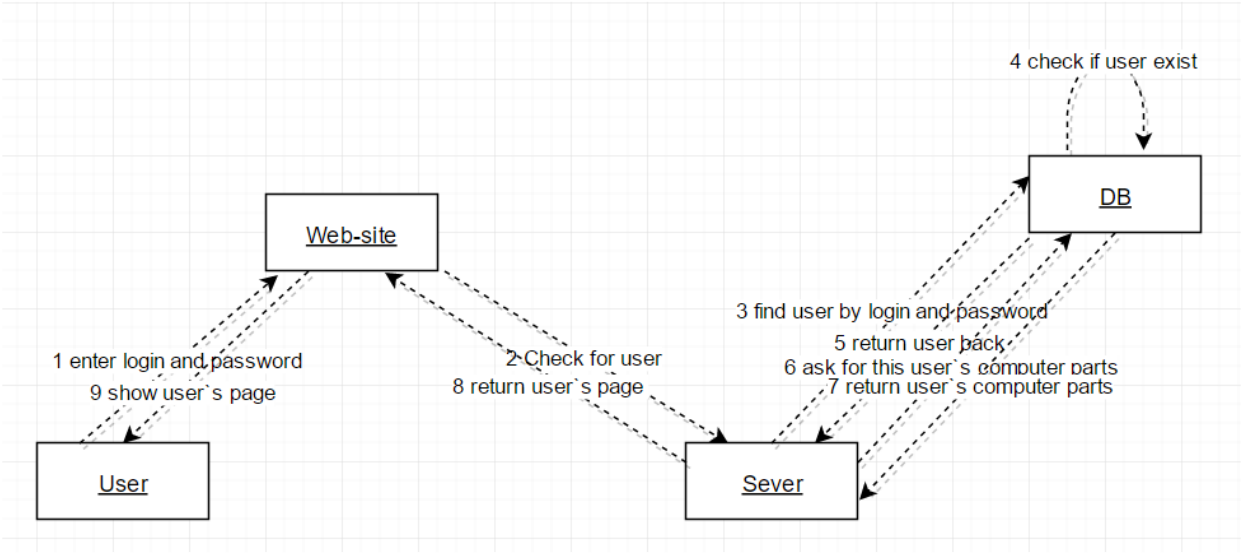


Рисунок 3.12 — Діаграма кооперації

На даній діаграмі зображено той самий модуль авторизації, але тут більший акцент зроблено власне на об'єкти, що приймають участь у взаємодії.

Наступна діаграма, яку варто зобразити — це діаграма активності системи. На рисунку 3.8 зображена діаграма активності, яка демонструє основну активність користувача в системі, а саме спочатку клієнт входить до системи будь-то веб-сайт чи мобільний додаток. Потім інформація поступає до API в нашій програмній системі це Nest.js та для з'єднання з базою даних використовується така ORM як – TypeORM. Після удачного запиту до API, коли дані пройшли верифікацію, ми даємо acsess token для юзера.

Верифікація ж була спроектована за допомогою Passport – це перевірка промежуточного слою для API.

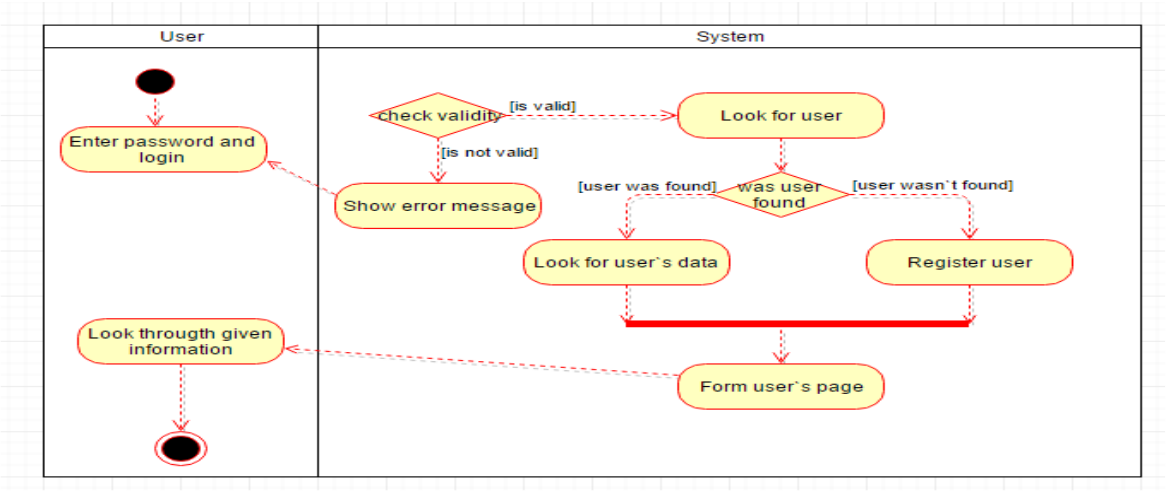


Рисунок 3.13 — Діаграма активності

Авторизуючись, гість може або увійти до аккаунту, де йому буде надано функціонал системи (робота з профілем та робота з датчиками), або ж він отримає помилку при спробі авторизації, якщо введені дані не співпадають з очікуваними. Після цього йому буде повернена помилка.

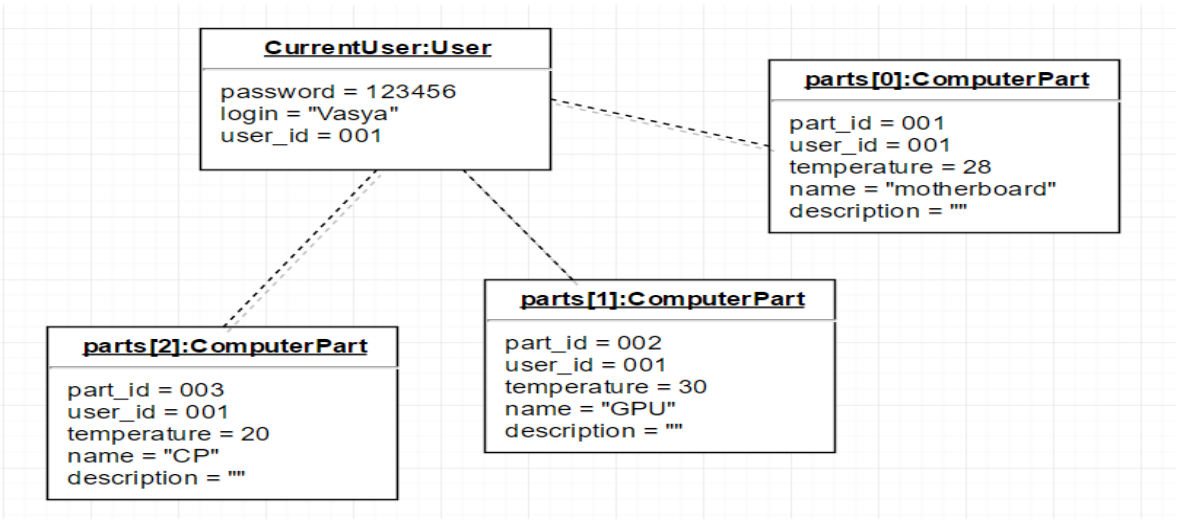


Рисунок 3.14 — Діаграма об'єктів

Дана діаграма є логічним продовженням діаграми класів з відображенням відповідних об'єктів та відношень між ними.

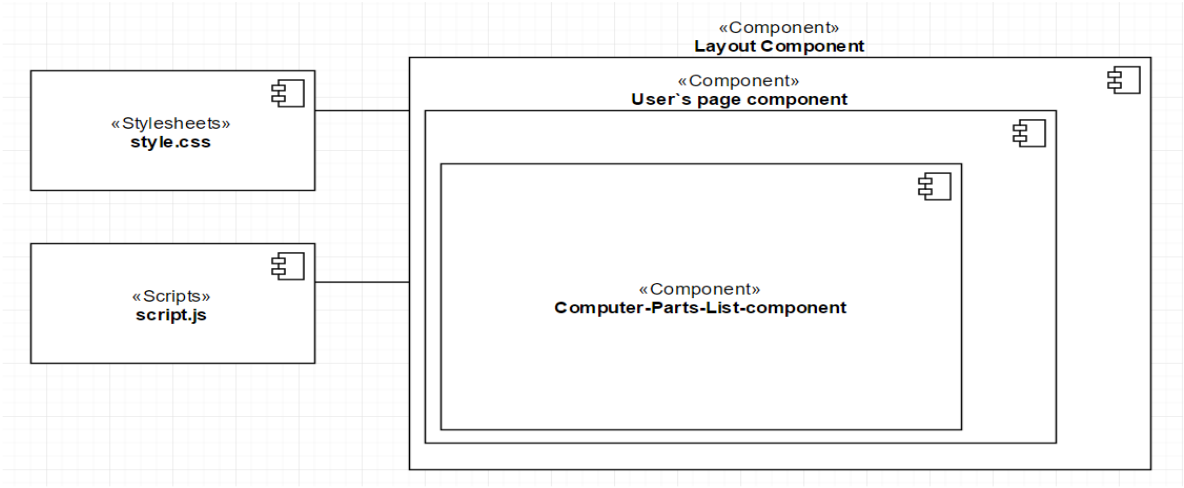


Рисунок 3.15 – Діаграма компонентів

Тепер, маючи діаграму компонентів, ми можемо побудувати діаграму розгортання для завершальної фази проекту. Вона зображена на рисунку 3.15.

На даній діаграмі зображується розміщення компонентів системи: на сервері додатку, сервері баз даних або мобільному додатку.

Фактично, діаграма розгортання повторює діаграму компонентів, за винятком того, що вона пояснює, які вузли працюють з якими програмними елементами.

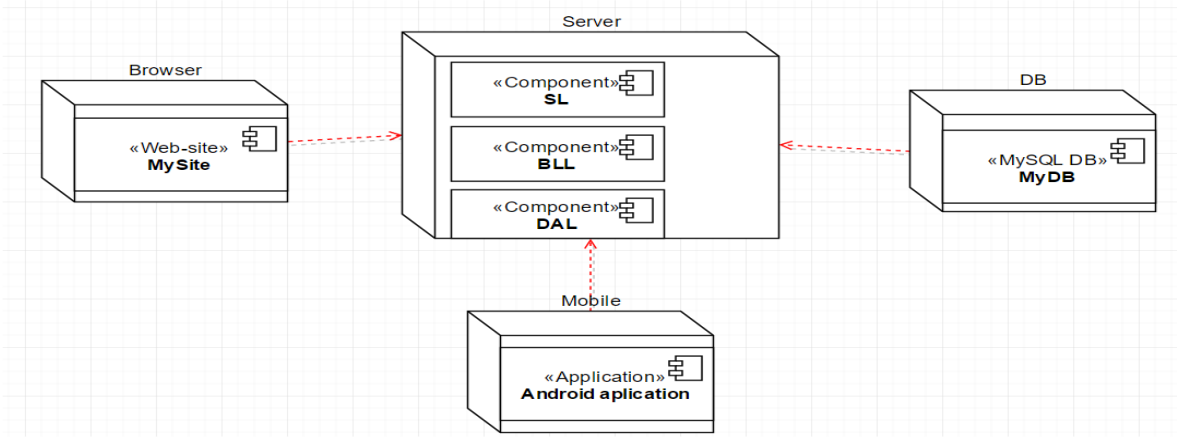


Рисунок 3.16 — діаграма розгортання

3.6 Розробка бази даних для зберігання інформації з розроблених моделей

Проектування бази даних програмної системи складається з етапів:

- концептуального моделювання;
- логічного моделювання;
- фізичного моделювання.

Основний компонент концептуальної моделі - це опис об’єктів предметної області та зв’язків між цими об’єктами.

Концептуальне моделювання використовують для загального огляду предметної області та створення концептуальної моделі, яка формує уявлення про вміст бази даних, отриманий в результаті аналізу предметної області. Тобто концептуальна модель - це опис структури бази даних, виконаний на природній мові, з використанням таблиць, діаграм та інших засобів. Модель повинна бути зрозумілою всім людям, що працюють з проектованою базою даних.

Для обраної програмної системи основними поняттями, які повинні зберігатися в базі даних є користувач та температура комплектуючих цього користувача. Щоб визначити допоміжні сутності необхідно провести аналіз цих понять та виявити їх характеристики.

Користувач системи матиме змогу реєструватися у системі, моніторити температуру його комп’ютера та приймати повідомлення на мобільний додаток. Також користувач має змогу переглядати загальну історію у вигляді діаграми роботи його комплектуючих.

Отже, у ході аналізу та концептуального моделювання були виявлені основні сутності системи та їх атрибути, що мають стати базовими при моделюванні та розробці системи.

Сутність «Користувач» має наступні атрибути:

– унікальний ідентифікатор;

– email;

– логін;

– пароль;

– ідентифікатор сутності температура;

Сутність «Температура» повинна мати такі атрибути:

– унікальний ідентифікатор;

– дата;

– температура материнської плати

– температура процесора

– температура відеокарти

– температура SSD

– температура жоского диску

На рисунку 3.17 зображена логічна схема бази даних системи. Пунктирною лінією позначується логічне відношення включення сутності ігрових об’єктів у серіалізованому стані.

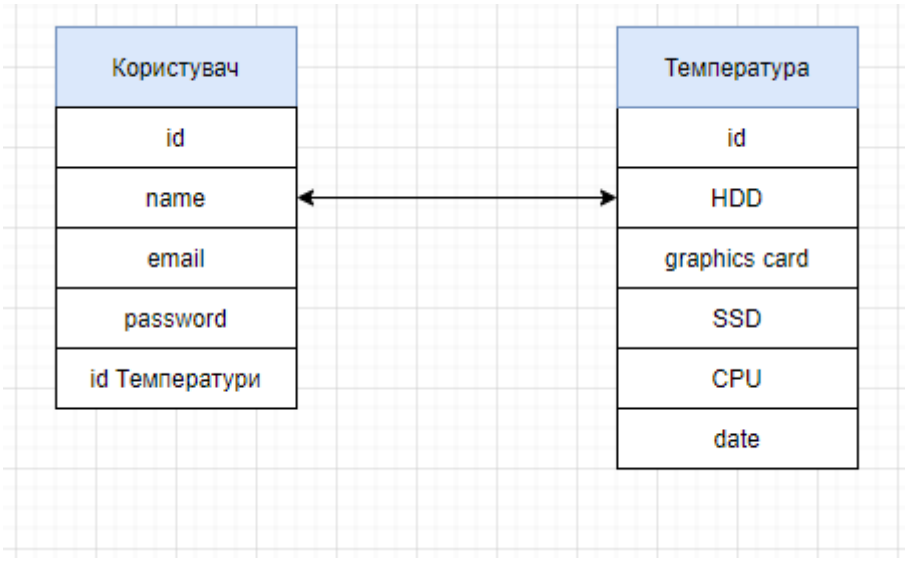


Рисунок 3.17 – Логічна схема бази даних

Схема бази даних знаходиться в третій нормальній формі тоді і тільки тоді, коли вона знаходиться в другій нормальній формі і жоден з не ключовийх атрибутів не є транзитивно залежним від первинного ключа.

З представленої схеми видно, що всі її неключові атрибути функціонально залежні від цілих потенційних ключів, тобто вона відповідає другій нормальній формі.

Крім того, кожен не ключовий атрибут не залежить один від одного, таких чином можна відзначити, що база даних відповідає третій нормальній формі.

Фізична реалізація бази даних буде складатись із двох фізічно незалежних серверів, що будуть зберігати різні об’єкти.

У процесі створення кожного документа (ObjectID) призначається "_id" унікальний ідентифікатор документа, який призначений для автоматичного сортування документів. СУБД MongoDB призначена для роботи з нереляціоннимі даними, наприклад, для роботи з базою даних Website, яка може включати в себе наступні колекції (сутності): статті, новини, фото, архів і т.д.

Якщо створення relational DBMS починається зі створення структури таблиць і установки зв'язків між таблицями, а потім заповнення таблиць і формування SQL запитів або інструкцій SQL, то в MongoDB спочатку створюють нову базу даних і задають її ім'я, потім при додаванні в неї даних вона автоматично створює колекцію і об'єкт (документ) з описом, що додається об'єкта в форматі JSON. Для зберігання даних в MongoDB застосовується формат, який називається BSON, скорочення від binary JSON.

В СУБД MongoDB для створення, читання, оновлення, видалення об'єктів застосовуються операції CRUD (create, read, update, delete).

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

4.1 Реалізація серверної та клієнтської частини

Для реалізації бази даних проекту було обрано MongoDB [21] – не реляційну систему керування базами даних.

Під час реалізації серверної частини було обрано технологію Nest.js та TypeORM, що використовується для створення REST сервісів. Веб–сервіс системи складається з трьох основних частин – шару доступу до даних, шару бізнес–логіки та кінцевою точкою доступу для клієнтів (API).

Для взаємодії між веб-сервісами та обміну даних використовується черга повідомлень, яка реалізована з використанням технології Rabbit MQ.

Завдяки використанню веб-сервісу та багаторівневої архітектури досягається висока масштабованість системи в цілому та її компонентів окремо. Також завдяки тому що було спроектовано як мобільний додаток та веб-сервіс, інформація що потрібна користувачу та звіти з роботи системи, а саме температурні показники комплектуючих, передаються швидше. Це також заслуга нашого дослідження з обробки великих даних, а саме методу – дерево рішень. За допомогою цієї утиліти можливо не тільки моніторити показники, а саме завантаженість, поточну, максимальну чи мінімальну завантаженість у процентах.

Одним з найцікавіших алгоритмів утиліти є синхронізація даних контролеру із програмою.

Під час розробки виникли наступні запитання:

- яким чином отримати інформацію про дані у контролеру;
- яким чином надсилати повідомлення, та у яких випадках.

Розглянемо перелічені пункти. Яким чином отримати інформацію для користувача? Це приклад як вивести температуру GPU для відеокарт NVIDIA

```
function Getemperatura: cardinal;

type
  NvCplGetThermalSettings = function(WindowsMonitorNumber: UINT;pGpuTemp, pUmgebTemp, pSlowDownTemp: Pointer): BOOL;stdcall;
var
  hInstNvcpl: THandle;
  GetThermalSettings: NvCplGetThermalSettings;
  UmgebTemp, SlowDownTemp: cardinal;
begin
  Result := 0;
  UmgebTemp := 0;
  SlowDownTemp := 0;
  hInstNvcpl := LoadLibrary('nvcpl.dll');
  if hInstNvcpl <> 0 then
    try
      GetThermalSettings:= GetProcAddress(hInstNvcpl,'NvCplGetThermalSettings');
      if Assigned(GetThermalSettings)  then
        GetThermalSettings(0, Addr(Result), Addr(UmgebTemp), Addr(SlowDownTemp));
    finally
      FreeLibrary(hInstNvcpl);
    end;
end;
```

Рисунок 4.1 — Приклад виводу інформації з BIOS’а для відеокарт NVIDIA

Яким же чином знати коли потрібно надсилати повідомлення, у випадку якщо один з комплектуючих перегрівся? Для цього нам потрібно в загалом знати допустиму температуру[11].

Давайте детально розглянемо температури, яких варто уникати.

Для температур процесора.

Стелею, звідки починаються проблеми (наприклад, пригальмовування), я звик вважати 60 (і більше) градусів. Температуру в 65-80 градусів вважаю досить критичною, тому що починається так званий троттлінг (а саме, режим пропуску тактів, тобто процесор спеціально починає працювати в кілька разів слабкіше, пропускаючи такти, щоб знизити свою температуру), аварійна перезавантаження \ самовиключеніе комп'ютера і т.п. Простіше кажучи, важливо стежити, щоб температура процесора не перебиралася за планку в 55 градусів, а краще і в 45-50.

Нормальними температурами я вважаю 35-40 градусів в просте і 45-55 при 100% багатогодинний навантаженні. Багато непогано розбираються в цьому люди будуть сперечатися, але я і до цього дня вважаю, що чим нижче температура, тим вища продуктивність, а саме процесор з температурою в 30 градусів впорається зі своїм завданням швидше, ніж процесор з температурою в 50, само собою при умови, що обидва процесора однакової потужності.[12]

Для температур материнської плати. В ідеалі температура чіпсета не повинна перевищувати 35 градусів. На практиці терпимі температури 40-45, для деяких моделей плат до 55. Взагалі з перегрівом чіпсетів на материнських платах майже не стикався, тому боятися особливо нічого.

Для температур відеокарти. Тут все залежить від того, наскільки вона потужна, що це за модель, який тип охолодження на ній встановлений і для яких цілей вона взагалі призначена (наприклад: для ігор, для роботи, або для медіацентру). Для сучасних відеокарт температури в 65-75 градусів повною багатогодинний навантаженні, - це нормально[13]. Для відносно стареньких моделей це може бути критично. Тому при появі перших ознак перегріву (про те, які вони бувають, читайте нижче) слід звернути пильну увагу на температурні режими і систему охолодження відеокарти.

Температури всередині корпусу. Мало хто знає, але температура повітря в корпусі грає дуже важливу роль, так як від неї залежать температури всіх компонентів системи, бо кулери обдувають всі корпусним повітрям. На жаль, заміряти точну корпусні температуру не вийде, але настійно рекомендується встановити кілька кольорів вдув-видув в корпусі.

Жорсткі диски. Нормальна температура для жорстких дисків - це все, що нижче 35-45 градусів, але, в ідеалі тримати ону в рази нижче, а саме в районі 30[14].

Приймаючи до уваги ці данні ми можемо знати коли відправляти повідомлення.

Приклад реалізації фабрики для відправки повідомлень можна побачити на рисунку 3.11

```
public class NotificationFactory {

    public static INotification get(DeviceType deviceType) {
        switch (deviceType) {
            case ONE_SIGNAL: {
                return new OneSignalNotification();
            }
            case BASIC: {
                return new BasicNotification();
            }
            default: {
                return new DefaultNotification();
            }
        }
    }
}
```

Рисунок 4.2 – Реалізація фабрики для відправки повідомлень

```
public class OneSignalNotification implements INotification {

    private String appId = "";

    private String authorization = "";

    @Override
    public void sendNotification(History history, List<String> keys, CardHolder holder) {
        try {
            String json = new Gson().toJson(new OneSignalDTO(this.appId, history, keys));

            URL url = new URL( spec: "https://onesignal.com/api/v1/notifications");
            HttpURLConnection con = (HttpURLConnection) url.openConnection();
            con.setUseCaches(false);
            con.setDoOutput(true);
            con.setDoInput(true);
            con.setRequestProperty("Content-Type", "application/json; charset=UTF-8");
            con.setRequestProperty("Authorization", "Basic " + this.authorization);
            con.setRequestMethod("POST");
            byte[] sendBytes = json.getBytes( charsetName: "UTF-8");
            con.setFixedLengthStreamingMode(sendBytes.length);
            OutputStream outputStream = con.getOutputStream();
            outputStream.write(sendBytes);
            int httpResponse = con.getResponseCode();
        } catch (Throwable t) {
            t.printStackTrace();
        }
    }
}
```

Рисунок 4.3 – Реалізація відправки повідомлень через сервіс OneSignal

4.2 Реалізація фізичної моделі бази даних

Для реалізації бази даних проекту було обрано MongoDB [15] - не реляційну систему керування базами даних.

Загальна схема фізичної моделі бази даних наведена на рисунку 4.2.

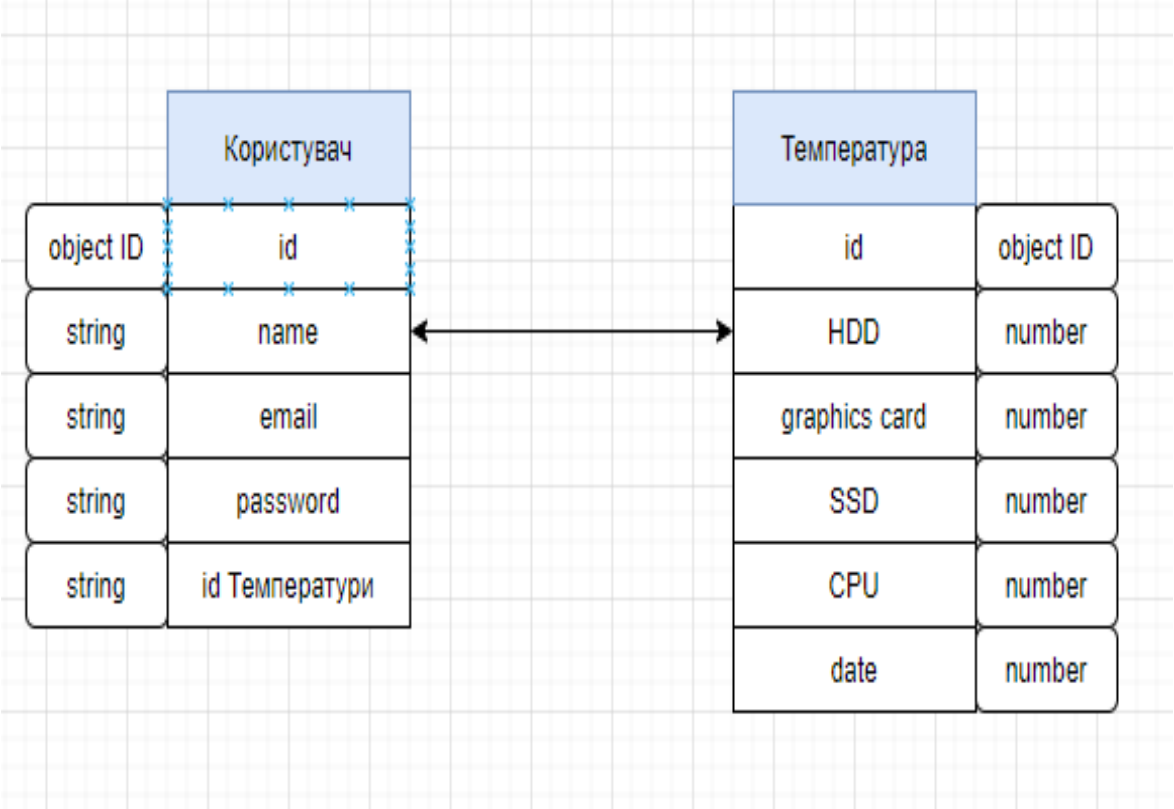


Рисунок 4.4 – Фізична модель бази даних

Потрібно зазначити, шо фізично база даних системи складається з двох фізічно незалежних серверів баз даних – ці сервери зберігають різні сутності системи, та обслуговують різні сервери[16].

5 ІНТЕРФЕЙС ТА ФУНКЦІОНАЛ

Після запуску програми, ми можемо побачити нагруженість та температуру процесора на рисунку 5.1

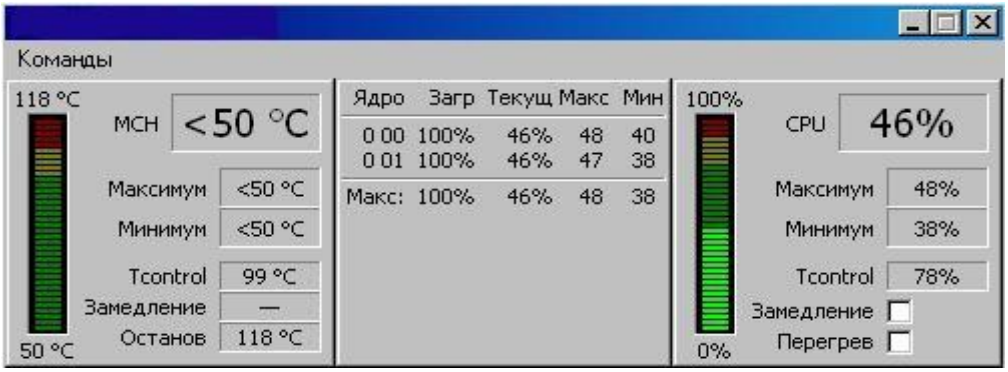


Рисунок 5.1 — Утиліта для контролю температури

За допомогою цієї утиліти можливо не тільки моніторити показники,,а саме завантаженість, поточну, максимальну чи мінімальну завантаженість у процентах. Також можливо убезпечити свій комп’ютер від перегріву зробивши позначку з боку від “Перегрев” як на рисунку 5.2

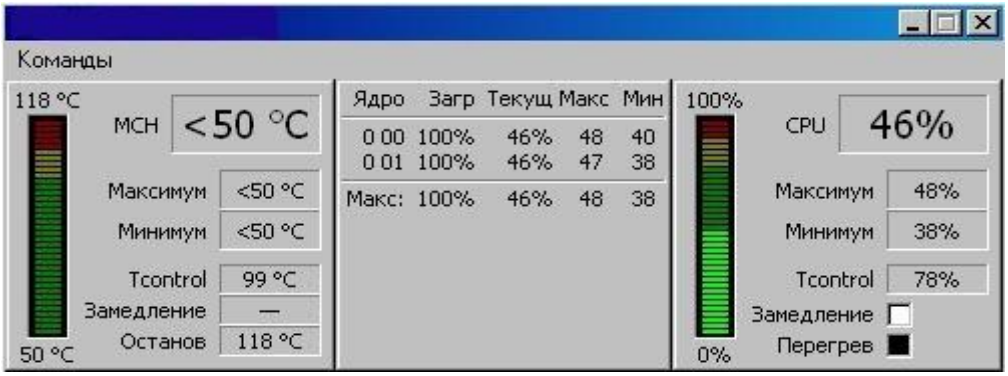


Рисунок 5.2 — підключення функції “Перегрев”

Як це працює? Знаючи критичну температуру для виключно цього процесору, а саме 78%, ми можемо знати на якій позначці треба прийняти заходи безпеки. В багатьох моделях процесорів вже є вбудований обмежувач що запобігає цьому шляхом зниження частоти процесора, цією функцією обладнана ця утиліта, а саме функція “Замедление” як на рисунку 5.3

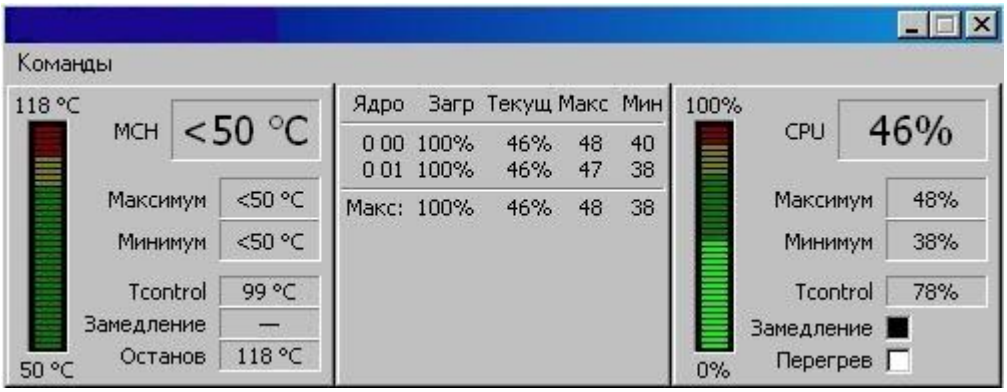


Рисунок 5.3 — підключення функції “Замедление”

Завдяки цією функції можливо зменшити навантаження на процесор, що дуже знизить показники температури, на 20%. Робоча частота зміниться, наприклад було 4.4 Мгц на 3.4 Мгц.

ВИСНОВКИ

В результаті виконання атестаційної роботи магістра було спроектовано та розроблено серверну частину для системи контролю температур, що взаємодіє із зчитувачами та BIOS’ом [6]. Відзначимо, що серверна частина цілком відповідає вимогам та набору функцій, що висувалися до неї, обробляє дані за допомогою метода Data Mining – дерев’я рішень.

Було проаналізовано методи роботи з великими даними, які можуть бути використані для побудови аналітичних звітів, а також робота з даними у розподілених файлових системах Було розглянуто основні аспекти побудови аналітичних систем, зокрема вилучення та обробка денормалізованих даних з існуючих сховищ. Можна з впевненістю зробити висновки, що для підвищення швидкодії опрацювання запитів однім із важних етаапів є структурування вхідних даних, які знаходяться в БД.

Була досліджена і реалізована схема бази даних в PostgreSQL і MongoDB, що дозволяє реалізувати складні запити і порівняти вищезгадані СУБД на рівні завдань, в якості яких виступили аналітичні запити під час перегляду цього всю базу даних. Були отримані результати вимірювань часу виконання запитів на "Гарячої" базі даних, які показують, що для даного класу задач MongoDB є більш ефективною, ніж PostgreSQL.

В ході проектування та розробки були використані набуті навички та знання з роботи із уніфікованим язиком моделювання UML, застосовані навички із роботи та проектування архітектури програмного забезпечення, відпрацьовані навики написання та форматування програмного коду та інших етапів продукту. Було проведено базове unit – тестування коду серверної частини.

Реалізація проекту відбувалася за допомогою мови програмування JS, C#. Під час роботи використовувалися WebStorm та Visual Studio 2015. У якості бази даних виступає MongoDB та API – Nest.js.Фреймворк IONIC4, Angular.

В процесі виконання роботи були закріплені теоретичні знання та навички розробки програмного продукту, починаючи з проектування і закінчуючи тестуванням програмного продукту, були підвищені знання в розробці програм, пов’язаних з платформою Node.js, TS, Angular .

В ході атестаційної роботи магістра було:

– розроблена програма контролю температури
– розроблений кросплатформений веб-додаток
– були проведенні дослідження методів обробки даних

ПЕРЕЛІК ПОСИЛАНЬ

1. Вільна енциклопедія Википедия [Електронний ресурс] / Мережева енциклопедія Wikipedia. 2000. – Forefront TMG 2010: Режим доступу: www/ URL : http://ru.wikipedia.org/

2. Освітній сайт «http://ru.iobit.com/gameassistant/» [Електронний ресурс] / Утиліта Game Assistant – Режим доступу: http:ru.iobit.com/gameassistant/ – 10.05.2018 р. Загол. з екрану.

3. Освітній сайт «http://www.aida64.ru/» [Електронний ресурс] / Утиліта Aida64– Режим доступу: http:www.aida64.ru/ – 15.05.2018 р. Загол. з екрану.

4. Освітній сайт «http://www.wisecleaner.com/wise-system-monitor.html/» [Електронний ресурс] / Утиліта Wise Systmer Monitor – Режим доступу: http://www.wisecleaner.com/wise-system-monitor.html/ – 13.05.2018 р. Загол. з екрану.

5. Ionic [Електронний ресурс] Режим доступу: www/ URL: https://ionicframework.com/docs

6. Зозуля Ю.Н. BIOS на 100%[Текст].Пітер, 2009. – 336с

7. Csikszentmihalyi M. Flow: The Psychology of Optimal Experience. —Нью-Йорк, NY, США: Harper and Row, 1990. — 456 с.

8. Bakkes S., Spronck P., van den Herik J. Opponent modelling for case-based adaptive game AI//Entertainment Computing. 2009. No 1 (кн. 1). С. 27–37.

9. Champandard A. 10 reasons the age of finite state machines is over. 2008. URL: http://aigamedev.com/open/article/fsm-age-is-over/

10. Vlasenko L., Chikrii A. On a differential game in a system with distributed parameters // Proceedings of the Steklov Institute of Mathematics, 2016, vol. 292 (1), P. 276-285 URL: https://link.springer.com/article/10.1134/ S0081543816020243

11. Millington I., Funge J. Artificial Intelligence for Games, 2nd ed. — Сан-Франциско, CA, США: Morgan Kaufmann Publishers Inc., 2009. — 896 с.

12. Bevilacqua F. Finite-state machines: Theory and implementation. 2013. URL: https://gamedevelopment.tutsplus.com/tutorials/finite-state-machines-theory-and-implementation-gamedev-11867

13. Лотов А.В., Поспелова І.І. Багатокритеріальні задачі прийняття рішень: Навчальний посібник, 197 с. – 2008.

14. Романьков, В.А. Вступ в теорію ігор: Навчальний посібник / Романьков В А. Москва, 2016. – 834 с.

15. Матузко Ю.О. Теорія прийняття рішень. Навчально-методичний посібник для студентів всіх спеціальностей / Матузко Ю.О. Запоріжжя, 2009. – 70 с.

16. Bertsimas, D. and Tsitsiklis, J. N. Introduction to Linear Optimization, 608 с. — 1997.

17. Волченська Т.В., Князьков В.С. Комп’ютерная математика: Частина 2. Теорія графів: Навчальний посібник. Пенза, 2002. – 101 с.

18. Playground [Електронний ресурс] Режим доступу: www/ URL: https://www.playground.ru/misc/sovetuem_poigrat_luchshie_igry_v_zhanre_tower_defense-134511 – Загол. з екрану.

19. Фаулер М., Дейвид Р. Архітектура корпоративних програмних додатків / Мартин Фаулер, Дейвид Райс; Москва, 2010. – 544 с.

20. Лотів А.В., Поспєлова І.І. Багатокритеріальні задачі прийняття рішень: Навчальний посібник. Москва, 2008. – 197 с.

21. MongoDB [Електронний ресурс] Режим доступу: www/ URL: https://www.mongodb.com/