НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І
ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету	В.о. завідувача кафедри
Інформаційних технологій	Комп'ютерних систем, мереж та кібербезпеки
_____ Глазунова О.Г., д.пед.н, проф.	_____ Касаткін Д.Ю., к.п.н., доц.
підпис ПІБ, вчене звання і ступінь	підпис ПІБ, вчене звання і ступінь
«___» _____ 2022 р.	«___» _____ 2022 р.

МАГІСТЕРСЬКА РОБОТА

На тему: «Дослідження комп'ютерної системи прогнозування стану обчислювальної програмно-апаратної платформи»

Спеціальність 123 «Комп’ютерна інженерія»

Освітня програма Комп’ютерні системи та мережі

Орієнтація освітньої програми _____

Керівник магістерської роботи: _____ / Шкарупило В.В. /
підпис ПІБ

Виконав: _____ / Семенов А.В. /
підпис ПІБ

КИЇВ-2022

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

«ЗАТВЕРДЖУЮ»
В.о. завідувача кафедри
комп'ютерних систем, мереж та кібербезпеки
_____ /Касаткін Д.Ю., к.п.н., доц./
підпис ПІБ, вчене звання і ступінь
«__» _____ 20__ р.

З А В Д А Н Н Я
ДО ВИКОНАННЯ МАГІСТЕРСЬКОЇ РОБОТИ СТУДЕНТУ

Семенов Артем Вікторович
(прізвище, ім'я, по батькові)

Спеціальність (напрям підготовки): комп'ютерна інженерія
Освітня програма: комп'ютерні системи та мережі
Орієнтація освітньої програми: _____

Тема магістерської роботи: «Дослідження комп'ютерної системи прогнозування стану обчислювальної програмно-апаратної платформи»

затверджена наказом ректора НУБіП України від "____" _____ 20___ р. № _____
Термін подання завершеної роботи на кафедру _____

Вихідні дані до магістерської роботи _____

Перелік питань, що підлягають дослідженню:
1. Аналітичний огляд
2. Проєктування комп'ютерної системи
3. Реалізація комп'ютерної системи

4.Дослідження комп'ютерної системи

Перелік графічного матеріалу (за потреби) _____

Дата видачі завдання "_____"_____ 2021 р. Керівник магістерської роботи _____ Шкарупило В.В., к.т.н., доц.
(підпис) (прізвище та ініціали)

Завдання прийняв до виконання _____ Семенов А.В.
(підпис) (прізвище та ініціали студента)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів проекту (роботи)	Примітка
1	Аналіз предметної області		Виконано
2	Проектування системи		Виконано
3	Реалізація системи		Виконано
4	Дослідження системи		Виконано
5	Оформлення пояснювальної записки		Виконано
6	Оформлення графічного матеріалу		Виконано

Студент _____

(підпис) (ініціали та прізвище)

Керівник проекту (роботи) _____

(підпис) (ініціали та прізвище)

РЕФЕРАТ

Пояснювальна записка: 79 сторінок, 37 рисунків, 3 таблиці, 19 джерел.

КОМП'ЮТЕРНА СИСТЕМА, МОНІТОРИНГ РЕСУРСІВ, ШВИДКОДІЯ, АДМІНІСТРАТОР, ВІЗУАЛЬНА ЧАСТИНА, ІНТЕРФЕЙС, ВІКНО КОРИСТУВАЧА, ANGULAR, .NET, WINDOWS FORMS, ТЕСТУВАННЯ

Мета – Розробка та дослідження комп'ютерної системи прогнозування стану обчислювальної програмно-апаратної платформи.

Об'єкт – комп'ютерна система для прогнозування та замовлення товарів на склад.

Предмет розробки – методи та програмні додатки для моніторингу стану обчислювальної програмно-апаратної платформи.

Проект складається з чотирьох розділів.

Перший розділ присвячено аналізу предметної області. Проводиться детальний огляд об'єкта, аналіз призначення системи. Проведено огляд існуючих рішень.

У другому розділі розкриті питання щодо вимог автоматизованої системи та її проектування.

Третій розділ присвячено реалізації компонентів системи.

У четвертому розділі проведено дослідження швидкодії та відмовостійкості системи.

Проведено моделювання поведінки та структури системи.

В результаті виконання дипломної роботи проведено аналіз, моделювання, розробка та дослідження розроблюваної автоматизованої системи.

ЗМІСТ

ПК – персональний комп’ютер

HTML – мова розмітки гіпертексту, HyperText Markup Language

CSS – каскадні таблиці стилів, Cascading Style Sheets

front-end – інтерфейс для взаємодії між користувачем і back end

JS – JavaScript

cookies – дані отримані від веб-сайту на веб-сервері

OPD – методологія об'єктних процесів, Object Process Methodology

мс – мілісекунда

UI – інтерфейс користувача, User interface

API – програмний інтерфейс застосунку, Application Programing Interface

TCP – Основний протокол передачі даних, Transmission Control Protocol

HTTP – Протокол прикладного рівня передачі даних

HTTPS – Розширення протоколу HTTP для підтримки

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

шифрування з метою підвищення безпеки

ВСТУП

Проблема з мережею може виникнути майже через будь-яку несправність в інфраструктурі, будь то вузьке місце в пропускній здатності, проблеми з конфігурацією або несправний мережевий компонент. Але більше половини збоїв в роботі IT-систем спричинені саме апаратними несправностями. Здатність швидко виявляти і вирішувати проблеми з апаратним забезпеченням має велике значення для забезпечення оптимізації продуктивності.

Хоча це може здатися простим рішенням, існує багато потенційних точок відмови обладнання, кожна з яких може сприяти уповільненню роботи.

Візьмемо, наприклад, сервери. Несправність сервера може бути викликана перевантаженим процесором, перевантаженим дисковим простором або пам'яттю, або несправним джерелом живлення. Але будь-якому серверу в мережі відомства можуть також загрожувати проблеми з навколишнім середовищем, такі як відмова вентилятора, підвищення температури сервера, а також піки або падіння напруги.

Успішний моніторинг обладнання підвищує продуктивність. Почніть з пропозиції з відображенням стану обладнання в реальному часі: робочий, попереджувальний або критичний. З цією можливістю часто з'являється додаткова можливість дивитися на ці дані в перспективі; встановлювати базові значення для таких речей, як швидкість обертання вентилятора процесора, температура сервера і робота джерела живлення - і відправляти оповіщення, коли це доречно.

Також важливо бачити в реальному часі стан використання ресурсів - і, при необхідності, оповіщення про такі речі, як завантаження процесора, використовувана пам'ять і ємність диска. Базові графіки прогнозування та метрики допоможуть визначити, коли ресурси досягнуть потужності, тому

ІТкоманда може працювати набагато раніше цих зростаючих вимог до потужностей.

ІТ-фахівці повинні шукати рішення з єдиним вікном, щоб гарантувати, що вся інформація про стан, незалежно від постачальника, типу обладнання або місця розташування обладнання, відображається на одному екрані стану. Це єдине, безперервне представлення моніторингу забезпечує один останній, критично важливий фактор в моніторингу обладнання: контекст.

Одна справа знати, що, наприклад, процесор наближається до граничної потужності. Додавання контексту може підказати ІТ-професіоналу, що перевантажений процесор відповідає за підтримку найважливішої місії агентства і його необхідно негайно виправити.

Кожна з цих можливостей - єдина скляна панель, моніторинг в режимі реального часу, контекст і моніторинг змін - сама по собі важлива для оптимізації продуктивності мережі. Разом вони можуть допомогти федеральним ІТфахівцям оптимізувати продуктивність і забезпечити більш ефективну систему безпеки.

1 АНАЛІТИЧНИЙ ОГЛЯД

1.1 Дослідження предметної області

Моніторинг параметрів роботи апаратного забезпечення комп'ютера дуже важливий для обслуговування обладнання. Під час використання апаратного забезпечення висока температура призводить до нестабільної роботи комп’ютерної системи та навіть впливає на термін служби обладнання. Таким чином, моніторинг параметрів апаратного забезпечення особливо важливий в автоматичних серверних кімнатах, кластерних серверах і масивних платформах хмарних серверів. Настільні комп’ютери, портативні комп’ютери, смартфони, планшети або інтелектуальні термінали, які можуть запускати програми, часто виробляються різними виробниками з дуже різним апаратним складом і виробничим процесом. Важко ефективно аналізувати стан працездатності апаратного пристрою [1]. За допомогою системи прогнозування стану ми можемо ефективніше оцінювати стан працездатності апаратного забезпечення та проводити своєчасне технічне обслуговування, якщо ми можемо виконувати повний аналіз різних параметрів апаратного забезпечення та даних про працездатність подібного обладнання. Крім того, більшість існуючих підприємств або шкіл встановлюють системи моніторингу навколишнього середовища в комп’ютерній кімнаті, для створення таких параметрів комп’ютерного обладнання, як моніторинг температури. Недолік полягає в тому, що для роботи потрібні значні інвестиції в апаратне забезпечення, і воно не може безпосередньо визначити температуру комп’ютера та різні стани та параметри. При цьому не вбачається можливим повністю відслідковувати ресурси існуючої комп'ютерної системи. Персонал обслуговування різноманітних обчислювальних пристроїв і інтелектуальних терміналів часто не в змозі переглянути історію параметрів роботи обладнання користувача. Це також створює великі незручності для обслуговування обладнання. У цій тезі, з точки зору обслуговування користувачів на основі параметрів моніторингу мережі, ми

пропонуємо систему моніторингу комп’ютерного обладнання на основі хмарної платформи. Це система моніторингу апаратного забезпечення хмарних обчислень, яка розділяє збір даних і керування зберіганням. Система моніторингу комп’ютерного обладнання на базі хмарної платформи розділяє зберігання та доступ до даних. Базуючись на платформі, вона уніфікує моделювання аналізу апаратних параметрів зберігання великих обсягів даних, щоб надати користувачам достовірну інформацію про обслуговування апаратного забезпечення.

1.2 Призначення системи

За допомогою програмного забезпечення моніторингу можна отримати дані шляхом моніторингу комплектуючих різноманітного комп'ютерного обладнання, робочих параметрів та супутнього програмного забезпечення та відправити їх на сервер моніторингу; на сервері моніторингу в хмарі отримати статистику, аналіз та опрацювання всіх видів параметричної інформації, за єдиним алгоритмом, на основі даних про різні параметри, визначені шляхом моніторингу стану працездатності комп'ютерного обладнання, зберегти параметри та стан, оперативно сповістити користувачів про стан працездатності. Система має складатися з трьох частин: клієнтської складової, серверної частини та сайту. Клієнтський компонент запускається на комп'ютері, показники якого відстежуються, або на інтелектуальному терміналі, а серверний компонент розгортається на кластері серверних моніторів.

Як показано на рисунку 1.1, вся система складається з трьох частин:

-клієнтського забезпечення;

-серверу;

-сайту.

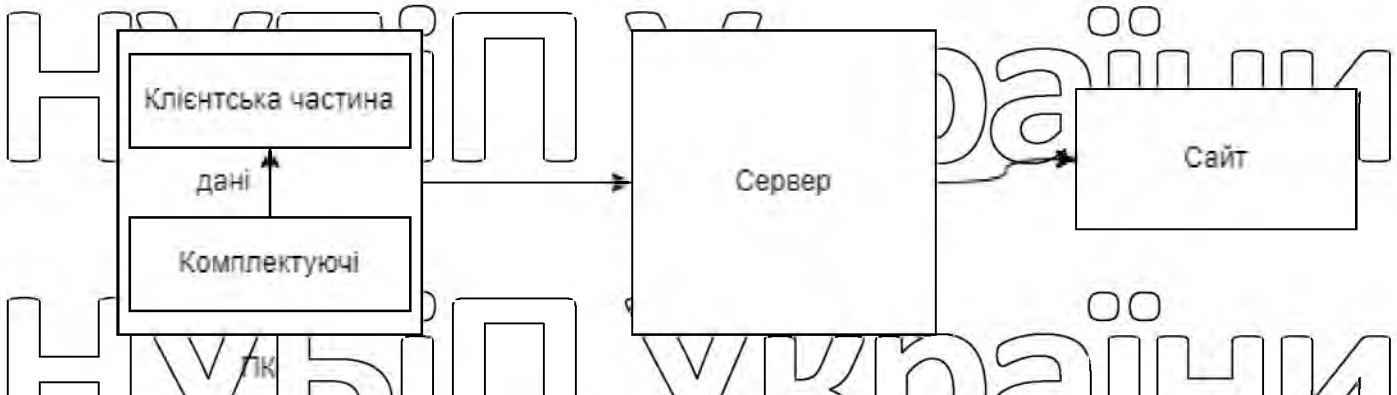


Рисунок 1.1 - Склад системи

В даний час материнські плати комп'ютерних систем, процесор та інші апаратні пристрої оснащені датчиками, які можуть зчитувати температуру процесора, напругу процесора, швидкість обертання вентилятора процесора, температуру хост-вузла і жорсткого диску, а також зчитувати дані жорсткого диска, такі як температура і постачальники обладнання; забезпечують зчитування даних датчика; параметри інтерфейсу доступу до даних можуть бути забезпечені функцією інтерфейсу вводу / виводу операційної системи, зчитуванням BIOS материнської плати, що зберігається в різних датчиках; дані в реальному часі. Важливими параметрами є, зокрема, наступні: температура процесора, напруга процесора, швидкість обертання вентилятора процесора, температура хоствузла, температура жорсткого диску тощо. За необхідності, можна також зчитувати дані операційної системи, щоб знати завантаження на процесор та інші параметри. В наш час материнська плата клієнтської системи, процесор та інші апаратні пристрої обладнані датчиками, які можуть зчитувати температуру процесора, напругу процесора, швидкість обертання вентилятора процесора, температуру обчислювальної системи, а постачальники обладнання забезпечують інтерфейс доступу до даних, пов'язаних із зчитуванням показів датчиків. Основним методом є зчитування в реальному часі параметрів різних датчиків, що зберігаються в апаратному забезпеченні – за допомогою функцій API, що надаються операційною системою. Наприклад, в системі Windows можна спочатку викликати функцію API CreateFile для відкриття пристрою і повернути

дескриптор на пристрій, з яким він асоціюється. Потім – викликати API-функцію DeviceIoControl і керуючу комунікаційну програму, а потім – зчитувати різні параметри роботи обладнання. Основним методом отримання робочих параметрів операційної системи є виклик інтерфейсних функцій, що надаються операційною системою для завершення роботи.

1.3 Огляд існуючих засобів

Інструмент моніторингу обладнання - це програмне забезпечення, яке взаємодіє з різними апаратними компонентами та надає значення фізичних властивостей цих компонентів у зручному для розуміння форматі, отримуючи дані з різних датчиків[2].

Фізичні компоненти, такі як материнська плата, процесор, вентилятори тощо, мають апаратні датчики для виявлення та вимірювання певних змін фізичних властивостей, таких як температура та напруга. Це програмне забезпечення буде представляти ці властивості в числових значеннях.

Розглянемо існуючі рішення.

1.3.1 SolarWinds Server and Application Monitor

SolarWinds SAM надає вам інструменти для моніторингу ваших серверів і додатків через єдину веб-консоль. Він надає користувацькі колекції шаблонів, монітори додатків та оповіщення для інтелектуального моніторингу стану та проблем додатків. Моніторинг понад 200 типів додатків, включаючи сервери додатків, сервери аутентифікації, сервери баз даних і багато іншого.

Інтерфейс даного продукта ви можете побачити на рисунку 1.2.

За допомогою SAM можна здійснювати моніторинг публічних, приватних або гібридних середовищ різними способами, в тому числі:

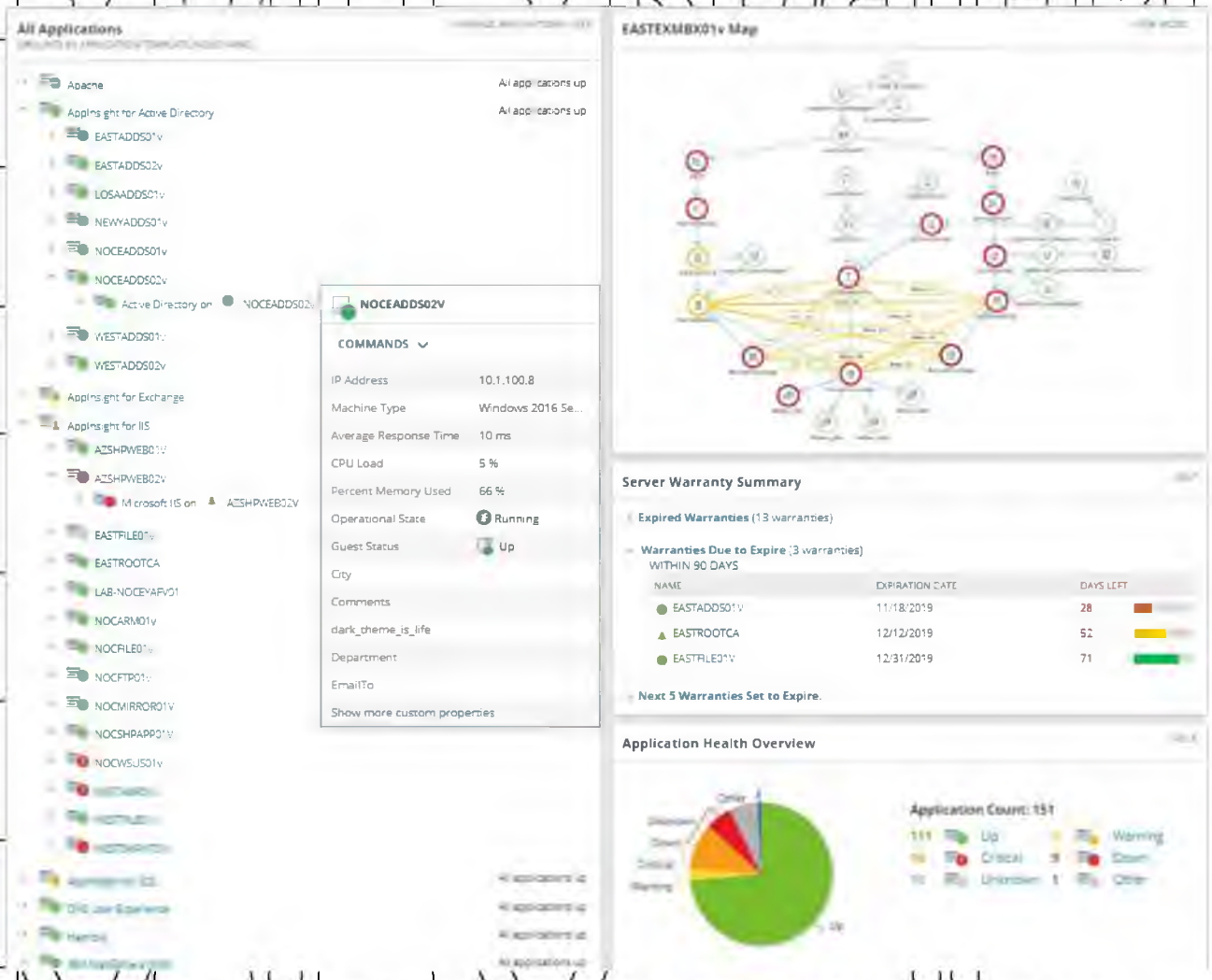


Рисунок 1.2 – Інтерфейс SolarWinds Server and Application Monitor

– appStack: Використовуйте інтерактивне візуальне відображення, щоб отримати поглиблену перспективу через все середовище, щоб допомогти виявити першопричину проблем з продуктивністю і доступністю;

– інформаційні панелі аналізу продуктивності (PerfStack): Співвідносити історичні дані і дані в реальному часі з декількох продуктів SolarWinds і типів об'єктів в єдиному поданні для поглибленого пошуку і усунення несправностей;

– хмарний моніторинг: Моніторинг додатків і серверів IaaS, PaaS і SaaS локально, за допомогою додаткових агентів для хмарних сервісів, таких як AWS, Microsoft Azure, Rackspace і т.д. Гібридний хмарний моніторинг дозволяє відстежувати ваші додатки, навіть коли вони переміщуються з локальної мережі в хмару;

– моніторинг контейнерів: Перегляд, відстеження і кореляція ключових показників продуктивності, включаючи процесор, пам'ять і час безвідмовної роботи, для контейнерів Docker, Docker Swarm, Kubernetes і Apache Mesos;

– опитувачі API: Якщо у вас є доступ до зовнішнього REST API, ви можете використовувати функцію API Poller для збору даних з сучасних стеків додатків і відображення їх на дашбордах SAM;

– додатки AppInsight: Моніторинг серверів Microsoft Active Directory, Exchange, IIS і SQL, і відображення метрик, стану і проблем для управління і обслуговування додатків і серверів;

– шаблони моніторингу додатків: Об'єднують монітори процесів, доступність портів і лічильники продуктивності для оцінки кожного аспекту програми, включаючи стан і загальний стан;

– залежності додатків: Виявляють, як програми і вузли взаємодіють, щоб забезпечити моніторинг важливих даних для ключових додатків;

– моніторинг серверної інфраструктури: Виявляйте і зберігайте дані інвентаризації IT-активів на ваших серверах, одночасно відстежуючи стан обладнання для забезпечення безперебійної роботи серверів і додатків.

SAM є частиною платформи SolarWinds SolarWinds Platform, що дозволяє легко додавати інші модулі для розширення можливостей моніторингу. Існує багато способів інтеграції SAM з іншими продуктами SolarWinds Platform, в тому числі:

- Пакет управління системами включає в себе SAM і наступні модулі:
  - virtualization Manager (VMAN): Отримайте уявлення про продуктивність, ємність і використання вашої віртуальної інфраструктури, включаючи хости, віртуальні машини, кластери, контейнери, віртуальні мережі зберігання даних (vSAN) та інші сховища даних;
  - монітор ресурсів зберігання (SRM): Забезпечує моніторинг продуктивності та оповіщення для всіх масивів зберігання даних;
  - web Performance Monitor (WPM): Виявлення та усунення проблем з продуктивністю веб-додатків і додатків SaaS до того, як це вплине на користувачів. При використанні з SAM, WPM може відображати взаємозв'язок між веб-транзакціями і підтримуючою інфраструктурою, дозволяючи вам бачити загальну інформацію про стан в одному місці. Наприклад, SAM може показати, що Microsoft IIS працює, але WPM може повідомити вас, якщо сайти завантажуються занадто довго. Після того, як ви виправите ситуацію, WPM може підтвердити, що транзакції користувачів працюють успішно.
- пакет продуктивності і конфігурації сервера поєднує в собі SAM і Server Configuration Monitor (SCM), пропонуючи моніторинг продуктивності і виявлення змін в єдиному рішення;
- пакет Log and Systems Performance Pack об'єднує SAM з Log Analyzer, що дозволяє агрегувати, шукати і будувати графіки даних журналів, одночасно відстежуючи продуктивність системи:
  - пакет оптимізації продуктивності додатків включає в себе SAM і аналізатор продуктивності баз даних (DPA);
  - додайте аналіз часу відгуку, щоб побачити першопричину проблем з додатками.

- використовуйте історичний аналіз і динамічні базові лінії для виявлення проблем з налаштуванням SQL;
- IT Operations Manager поєднує в собі SAM з VMAN і SRM.

1.3.2 NinjaOne (Formerly NinjaRMM)

Найкраще підходить для постачальників керованих послуг (MSP), ITсервісних компаній, а також малих та середніх підприємств з невеликими ITвідділами.

Ціноутворення: NinjaOne пропонує безкоштовну пробну версію свого продукту. Ціна на Ninja встановлюється за кожен пристрій в залежності від необхідних функцій.

NinjaOne надає інтуїтивно зрозуміле програмне забезпечення для управління кінцевими точками для постачальників керованих послуг (MSP) та IT-фахівців для проактивного управління IT-проблемами з будь-якого місця.

З Ninja ви отримуєте повний набір інструментів для моніторингу, управління, захисту і поліпшення всіх ваших мережевих пристроїв, робочих станцій Windows, Mac, ноутбуків і серверів незалежно від їх місця розташування.

Інтерфейс даного продукта ви можете побачити на рисунку 1.3.

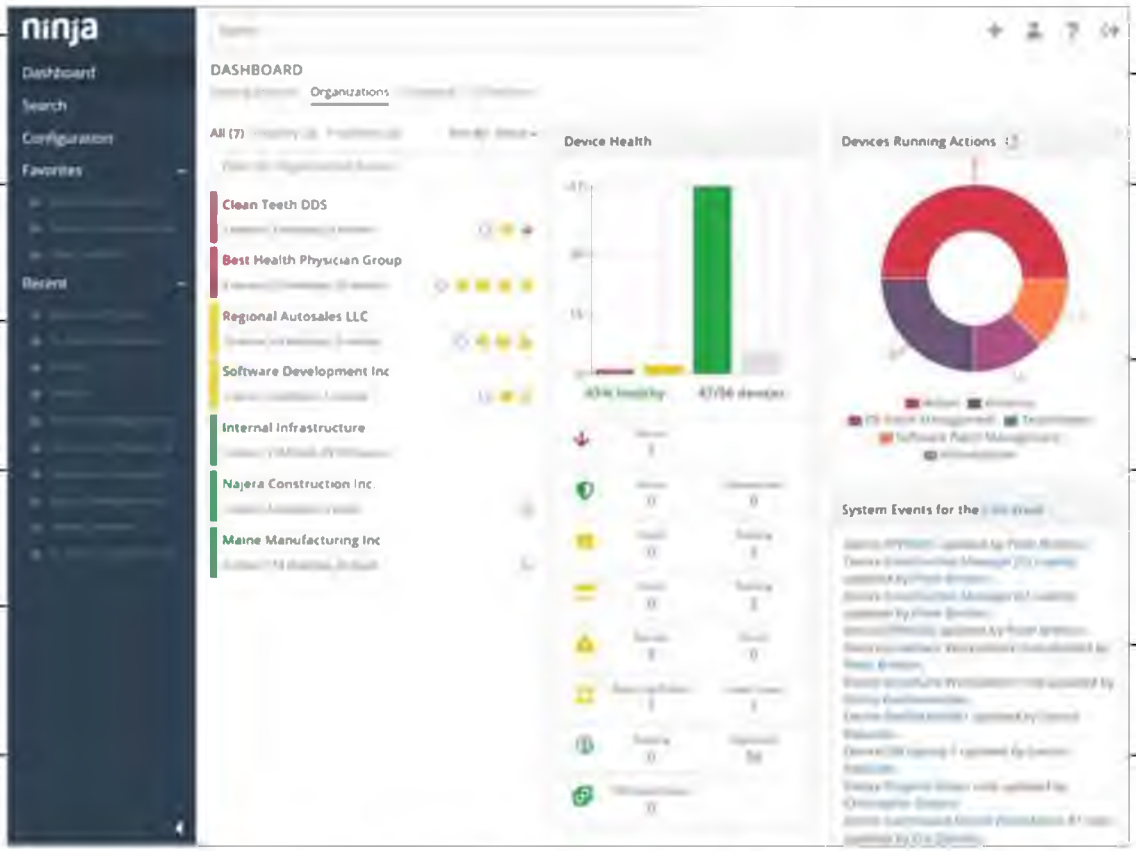


Рисунок 1.3 – Інтерфейс SolarWinds Server and Application Monitor

Можливості

- відстежуйте стан і продуктивність всіх ваших робочих станцій, ноутбуків і серверів на базі Windows і MacOS;
- отримайте повну інвентаризацію апаратного і програмного забезпечення;
- віддалено керуйте всіма пристроями, не перериваючи роботу кінцевих користувачів за допомогою надійного набору віддалених інструментів;
- автоматизуйте виправлення ОС і сторонніх додатків для пристроїв з Windows і MacOS;
- стандартизуйте розгортання, конфігурацію та управління пристроями за допомогою потужної IT-автоматизації;
- безпосередній контроль над пристроями з віддаленим доступом.

Вердикт: NinjaOne створила потужну, інтуїтивно зрозумілу платформу ІТмоніторингу, яка підвищує ефективність, зменшує обсяги квитків, покращує час вирішення тікетів і яку люблять використовувати ІТ-профі.

1.3.3 Atera

Atera - це хмарне рішення для віддаленого управління ІТ, яке забезпечує потужне та інтегроване рішення для MSP, ІТ-консультантів та ІТ-відділів.

За допомогою Atera ви можете контролювати необмежену кількість пристроїв і отримувати інформацію про продуктивність і статистику з плином часу, а також виконувати віддалене управління, таке як установка програмного забезпечення, командний рядок, PowerShell, запуск сценаріїв, управління файловими системами і багато іншого!

Atera - це універсальний набір інструментів для управління ІТ, який включає в себе все необхідне в одному інтегрованому рішенні. Atera включає в себе віддалений моніторинг і управління (RMM), віддалений доступ, управління виправленнями, звітність, бібліотеку сценаріїв, тікетінг, службу підтримки і багато іншого!

Інтерфейс даного продукта ви можете побачити на рисунку 1.4.

Рисунок 1.4 – Інтерфейс Atera

Atera пропонує доступну і проривну модель ціноутворення на кожну технологію, що дозволяє управляти необмеженою кількістю пристроїв і кінцевих точок за фіксованою низькою ціною.

Ви можете вибрати гнучку щомісячну підписку або річну підписку зі знижкою. У вас буде три різних типи ліцензій на вибір, і ви зможете випробувати всі можливості Atera на 100% безкоштовно.

Функції:

- моніторинг і управління необмеженою кількістю кінцевих точок, серверів і настільних комп'ютерів, як Mac, так і Windows.
- моніторинг SNMP-пристроїв, принтерів, брандмауерів, комутаторів і маршрутизаторів;
- усунення неполадок в мережі за допомогою однієї інтегрованої платформи, включаючи квитанції і автоматичне виставлення рахунків;

- проактивний моніторинг продуктивності і доступності всіх керованих пристроїв. процесор, пам'ять, використання HD, апаратне забезпечення, доступність і багато іншого;
- автоматизовані звіти, які відстежують і вимірюють ваші мережі, активи, стан системи і загальну продуктивність;
- індивідуальні налаштування оповіщень і порогові значення, а також запуск автоматичного обслуговування і оновлень.

Вердикт: Завдяки фіксованій ціні для необмеженої кількості пристроїв, Atera є дійсно найкращим універсальним рішенням для моніторингу мережі, яке потрібно IT-фахівцям. Спробуйте 100% безкоштовно. Це безризиково, кредитна картка не потрібна, і ви отримаєте доступ до всього, що може запропонувати Atera.

1.3.4 Paessler PRTG Network Monitor

PRTG - це універсальне програмне забезпечення для моніторингу мережі. Всі ваші послуги хмарних обчислень можуть контролюватися і управлятися централізовано. Він може контролювати всі типи серверів в режимі реального часу на предмет наявності, доступності, продуктивності та надійності. Найкраще підходить для малого та великого бізнесу.

Ціноутворення: Paessler надає 30-денну безкоштовну пробну версію для необмеженої версії. Він також пропонує безкоштовну версію. Тарифні плани починаються від $1600 за 500 датчиків і установку 1 сервера.

Особливості:

- ви зможете визначити вузькі місця, знаючи про пропускну здатність, що використовується пристроями і додатками;
- він надає функції для моніторингу конкретних наборів даних з баз даних;

- він надасть детальну статистику для додатків, що працюють у вашій мережі;
- допоможе відстежувати всю локальну мережу, яка включає робочі станції, маршрутизатори, комутатори і т.д.

Інтерфейс даного продукта ви можете побачити на рисунку 1.5.

Рисунок 1.5 – Інтерфейс Paessler PRTG Network Monitor

Вердикт: PRTG – це комплексне рішення, і вам не знадобляться додаткові плагіни. Це проста у використанні і потужна платформа для моніторингу систем, пристроїв, трафіку і додатків у вашій IT-інфраструктурі.

1.3.5 HWMonitor

Ця програма моніторингу обладнання працює шляхом зчитування напруги, температури, вентиляторів та швидкості. Він підтримує ОС Windows. Через S.M.A.R.T. вона може зчитувати температуру жорстких дисків. Він може читати температуру графічного процесора відеокарти. Також доступна розширена версія HWMonitor, тобто HWMonitor Pro.

Найкраще підходить для малого та середнього бізнесу.

Ціна: HWMonitor доступний безкоштовно, HWMonitor Pro доступний у двох типах ліцензій: Стандартна ($22.31) та Розширена ($39.10).

Інтерфейс даного продукта ви можете побачити на рисунку 1.6.

Рисунок 1.6 – Інтерфейс HWMonitor

Особливості:

- HWMonitor Pro доступний для платформ Windows і Android.
- HWMonitor Pro надає функції віддаленого моніторингу, генератора графіків та покращеного інтерфейсу;
- використовуючи просте з'єднання TCP/IP, він може спостерігати за датчиками одного або декількох віддалених ПК або пристроїв Android;

- його генератор графіків може зберігати дані моніторингу та створювати графіки реєстрації.

Вердикт: HWMonitor - легка програма, яка регулярно оновлюється.

1.3.6 Open Hardware Monitor

Open Hardware Monitor може контролювати датчики температури, швидкість обертання вентилятора, напругу, навантаження і тактову частоту комп'ютера. Він забезпечує підтримку більшості мікросхем апаратного моніторингу.

Інтерфейс даного продукта ви можете побачити на рисунку 1.7.

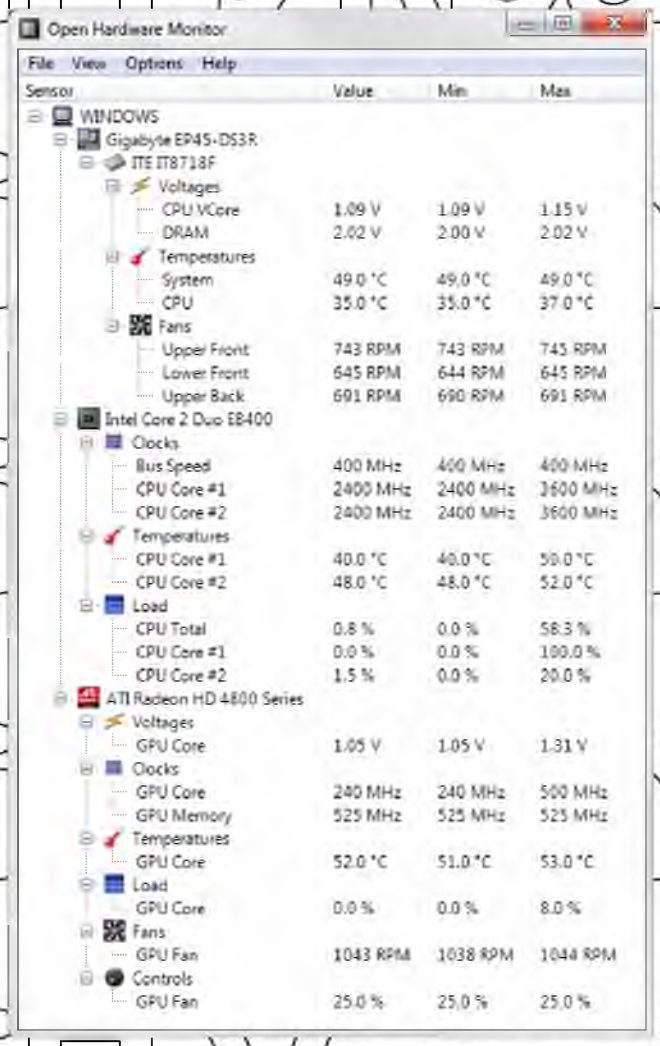


Рисунок 1.7 – Інтерфейс Open Hardware Monitor

Ціна: Це програмне забезпечення з відкритим вихідним кодом доступне безкоштовно.

Можливості:

- Open Hardware Monitor зчитує температурні датчики ядра процесорів Intel і AMD для моніторингу температури процесора;
- він може відображати температуру жорсткого диска SMART;
- ви можете переглядати відстежувані значення в головному вікні, в настроюваному гаджеті робочого столу або в системному треї;

Вердикт: Open Hardware Monitor - просте у використанні, безкоштовне програмне забезпечення з відкритим вихідним кодом для моніторингу температурних датчиків, швидкості обертання вентиляторів, напруги, навантаження і тактової частоти.

2 ПРОЄКТУВАННЯ КОМПЮТЕРНОЇ СИСТЕМИ

2.1 Загальні вимоги до системи

Специфікація системних вимог (SRS) (також відома як специфікація вимог до програмного забезпечення) - це документ або набір документації, який описує особливості та поведінку системи або програмного додатку. Вона включає в себе різноманітні елементи, які намагаються визначити передбачувану функціональність, необхідну замовнику для задоволення різних користувачів[3].

На додаток до визначення того, як система повинна поводитися, специфікація також визнає на високому рівні основні бізнес-процеси, які будуть підтримуватися, які спрощують припущення були зроблені і які ключові параметри продуктивності повинні бути задоволені системою.

Взаємозв'язки типів інформації для вимог можна побачити з рисунку 2.1.

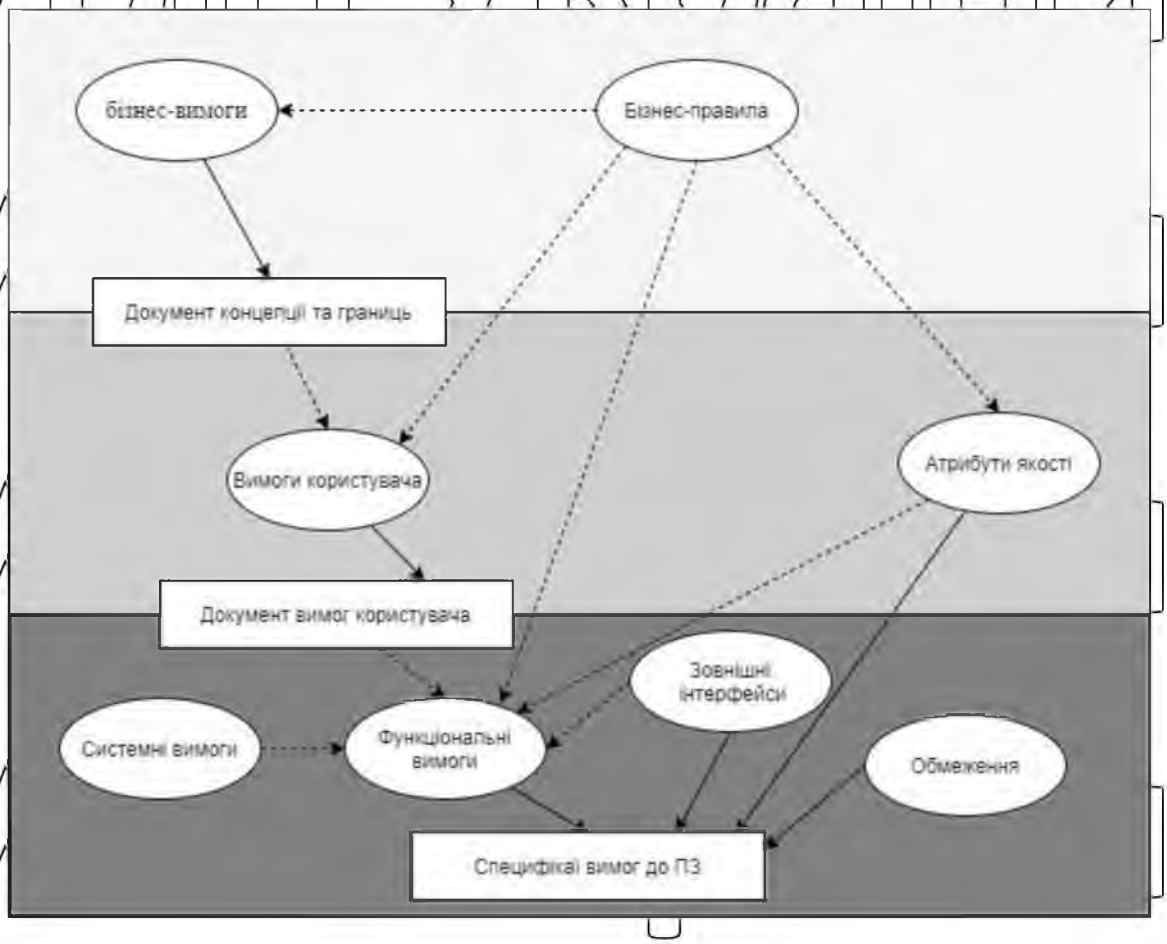


Рисунок 2.1 – Взаємозв'язки типів інформації для вимог

Основні елементи.

Залежно від використовуваної методології (гнучкої чи водоспадної), рівень формальності та деталізації СМР буде відрізнятися, але в цілому СМР повинна включати опис функціональних вимог, системних вимог, технічних вимог, обмежень, припущень та критеріїв прийнятності. Кожна з них описана більш детально нижче:

- бізнес-драйвери;
- бізнес-модель;
- функціональні та системні вимоги;
- бізнес та системні сценарії використання;
- технічні вимоги;
- системні якості;
- обмеження та припущення, - критерії прийнятності.

У цьому розділі описуються причини, з яких замовник прагне створити систему. Обґрунтування нової системи є важливим, оскільки воно визначатиме рішення, що приймаються бізнес-аналітиками, системними архітекторами та розробниками. Ще однією вагомою причиною для документування бізнесобґрунтування системи є те, що замовник може змінити персонал під час реалізації проекту. Документація, яка чітко визначає бізнес-обґрунтування системи, допоможе зберегти підтримку проекту, якщо первісний спонсор піде.

Рушійні сили можуть включати як проблеми (причини, через які існуючі системи/процеси є недостатніми), так і можливості (нові бізнес-моделі, які система зробить доступними). Зазвичай для забезпечення мотивації для нової системи необхідна комбінація проблем і можливостей.

Бізнес-модель

У цьому розділі описується основна бізнес-модель замовника, яку система повинна буде підтримувати. Сюди входять такі елементи, як організаційний

контекст, діаграми поточного та майбутнього стану, бізнес-контекст, ключові бізнес-функції та діаграми потоків процесів. Цей розділ зазвичай створюється на етапі функціонального аналізу.

Функціональні та системні вимоги

Цей розділ зазвичай складається з ієрархічної організації вимог, з бізнес/функціональними вимогами на найвищому рівні та детальними системними вимогами, перерахованими як їх дочірні елементи.

Як правило, вимоги записуються у вигляді тверджень на кшталт "Система повинна мати можливість робити x" з додатковою інформацією та деталями, включеними в міру необхідності.

Бізнес та варіанти використання системи

Цей розділ зазвичай складається з діаграми варіантів використання UML, яка ілюструє основні зовнішні сутності, які будуть взаємодіяти з системою разом з різними варіантами використання (цілями), які їм потрібно буде виконувати. Для кожного варіанту використання буде формальне визначення кроків, які необхідно здійснити для виконання бізнес-цілі, разом з будь-якими необхідними передумовами та пост-умовами.

Бізнес варіанти використання зазвичай випливають з функціональних вимог, а системні варіанти використання зазвичай випливають з системних вимог.

Етапи варіантів використання також можуть бути представлені у вигляді блок-схеми.

Технічні вимоги.

Цей розділ використовується для перерахування будь-яких "нефункціональних" вимог, які по суті втілюють технічне середовище, в якому повинен працювати продукт, і включають технічні обмеження, в яких він повинен працювати. Ці технічні вимоги мають вирішальне значення для визначення того, як функціональні вимоги більш високого рівня будуть декомпоновані в більш конкретні системні вимоги.

Системні якості.

Цей розділ використовується для опису "нефункціональних" вимог, які визначають "якість" системи. Ці вимоги часто називають "-функціональними", оскільки більшість з них закінчуються на "ility". До них відносяться такі елементи, як: надійність, доступність, ремонтопридатність, безпека, масштабованість, ремонтопридатність.

Вимоги до комп'ютерної системи прогнозування стану обчислювальної програмно-апаратної платформи можна побачити з таблиці 2.1.

Таблиця 2.1 Вимоги до системи автоматизованого обліку енергоспоживання

Тип вимоги	Вимоги до системи
Бізнес - правило	Система повинна працювати впродовж всього часу доки працює ПК.
Функціональні вимоги	- зчитування даних з сенсорів комплектуючих ПК; - створення графіків використання ресурсів ПК; - перегляд інформації про процеси які навантажують ПК;
Системні вимоги	Система повинна підтримувати стандартні периферійні пристрої у вигляді миші, клавіатури та монітора.
Зовнішні вимоги до інтерфейсу	Доступ до глобальної мережі ethernet.
Вимоги користувачів	Користувач: - реєстрація та авторизація; - перегляд інформації про стан власної системи; - перегляд графіків стану ПК;

Продовження таблиці 2.1

Тип вимоги	Вимоги до системи
Нефункціональні вимоги	Система повинна підтримуватися операційними системами Windows.
Обмеження	Інтерфейс програми не повинен заважати роботі за ПК;
Артибут якості	- програма повинна зберігати всі дані на сервері; - доступ до власних даних повинні мати лише користувачі.

На відміну від функціональних вимог (які зазвичай мають описову форму), системні якості зазвичай складаються з таблиць конкретних показників, яким повинна відповідати система, щоб бути прийнятою.

2.2 Розробка SRS специфікації вимог

Специфікація вимог до програмного забезпечення - це документ, який надає план основної програмної системи, яку бізнес має намір створити та впровадити. Цей документ містить деталі програмної системи на функціональному, технічному та комерційному рівнях. Це, в свою чергу, представляє системні вимоги, операційні процедури та стратегії, організаційні та еталонні моделі, стандарти та державні нормативні акти, що стосуються програмного забезпечення. Зрештою, вона містить достатньо інформації, щоб керувати розробкою та розгортанням програмного забезпечення на підприємстві[4].

Ідея створення такої документації ґрунтується на тому, що програмне забезпечення або системи часто змінюються, і ці зміни можуть легко призвести

до технічних проблем. Зробіть крок назад і подумайте про це, як про будівництво корабля. У вас є зображення щогли або вітрила, і ви розумієте, як вони подорожують по хвилях. Однак, щоб поставити корабель на воду і утримати його від відплиття, потрібно надати більшої конкретики.

SRS - це, по суті, документ, який визначає, що повинна робити дана програмна система, і піклується про різні вимоги. Він написаний відповідно до потреб програмного забезпечення і гарантує, що програмне забезпечення не створить жодних проблем для кінцевих користувачів. Різні функції програмного забезпечення чітко деталізуються і їм приділяється особлива увага. SRS використовується для забезпечення визначеності, вартості, ризиків та цілей планування часу при розробці програмного забезпечення, включаючи управління проектами, для збільшення обсягу, покращення якості та зниження вартості програмної системи.

Документ є об'єднуючим стандартом, яким користується вся команда. Використовуючи SRS, проект може бути виконаний успішно та ефективно.

Кому потрібне технічне завдання на програмне забезпечення

Сьогодні поширеним трендом електронної комерції серед власників малого та середнього бізнесу є використання SRS для написання бізнес-плану створення програмних рішень. Незалежно від розміру вашого програмного проекту, SRS є ключовою фундаментальною практикою управління програмними проектами, якої повинна дотримуватися кожна команда або проект. Завдання вашої компанії - створити стійкий програмний продукт. Ви також повинні мати можливість залишити слід у житті вашої команди розробників. Тобто, забезпечити їм стабільне робоче середовище і слідувати своїм довгостроковим цілям.

Чому важливий документ специфікації вимог до програмного забезпечення?

Написання вимог до програмного забезпечення - це відповідальна робота, яка зазвичай виконується різними командами. Незалежно від того, чи є ви

менеджером проекту, розробником або бізнес-аналітиком, вам, ймовірно, було важко спілкуватися з усіма сторонами щодо вимог до вашого проекту. Уважне вивчення їхніх ролей проллє світло на важливу спільну роботу між ними і допоможе перевірити та вдосконалити процес.

Найкращий спосіб гарантувати, що команда розробників створить програмне забезпечення, яке відповідає потребам ваших клієнтів, - це забезпечити наявність чітких вимог до програмного забезпечення. Вимоги до програмного забезпечення та валідація повинні відбуватися на самих ранніх стадіях розробки продукту. Перш ніж продовжити розробку продукту і випустити його на ринок, важливо ретельно продумати мету і функції вашого рішення. Це дозволить вам врахувати майбутні зміни і допоможе уникнути марної трати часу і грошей на розробку продукту.

По суті, SRS слугує дорожньою картою для потенційного проекту, оскільки вона є керівництвом для кожного рішення та дії, які необхідно здійснити під час проекту:

- в ньому викладені високорівневі реквізити програмного забезпечення, яке буде створено;
- дає дизайнерам чітке уявлення про проект;
- надає тестувальникам поради щодо створення тестових кейсів, які відповідають потребам бізнесу;
- слугує посібником для користувачів продукту, які бажають зрозуміти програмне забезпечення, яке вони придбали;
- дозволяє всім зацікавленим сторонам мати чітке уявлення про майбутній продукт і прийняти рішення про його валідацію;
- без добре написаної специфікації продукт може бути створений занадто швидко і не в повній мірі відповідати потребам кінцевого споживача.

Система прогнозування стану обчислювальної програмно-апаратної платформи направлена на керування роботою великою кількістю працівників не зважаючи на специфікацію підприємства. Головне завдання системи – облік та збір даних про стан комп'ютерів, серверів та інших апаратних плафторм, які підтримують Windows для подальшого аналізу та відслідковування їх стану та запобігання виходу з ладу їх компонентів.

Специфікація вимог до програмного забезпечення.

Введення.

Призначення.

Ця специфікація вимог до ПЗ описує функціональні та нефункціональні вимоги до розроблювальної системи. Цей документ призначений для підприємства, яке буде перевіряти та реалізовувати коректність роботи системи.

Область дії.

Система дозволить підприємствам, незважаючи на специфікацію роботи, але розміром не більше 10000 чоловік, проводити аналіз стану їх техніки.

Короткий огляд.

Далі у документі специфікації буде надано інформацію про спеціалізацію продукту, його функціональні та специфічні вимоги, обмеження системи.

Повний опис.

Контекст продукту.

Система прогнозування стану обчислювальної програмно-апаратної платформи це система для автоматизованого обліку стану комплектуючих їх ЕОМ, яка використовується на підприємствах на яких до 10000 робочих машин(ПК). Дана система повинна зчитувати дані з датчиків на комплектуючих ПК. Ця система збирає відомості про стан (загрузку, температуру комплектуючих та детальну інформацію про процеси які використовують ресурси системи).

Інтерфейси користувача:

- користувач керує системою за допомогою комп'ютерної миші;
- система відображає поточний стан вашого ПК та окремо по кожному комплектуючому;
- система дає змогу створювати новий аккаунт; Апаратні інтерфейси – сенсори вашого ПК.

Інтерфейси програмного забезпечення – не виявлено.

Інтерфейси зв'язку – не виявлено.

Пам'ять – наявність як мінімум 100Мб вільного місця на жорсткому диску вашого ПК та 40 вільних Мб ОЗУ.

Вимоги щодо адаптації місця використання – система адаптована під сімейство ОС Windows.

Функції продукту.

Система запускається подвійним натисканням лівої кнопки миші по іконці програми. Для початку роботи потрібно зайти за наявним логіном та паролем, якщо немає то створити новий аккаунт. Після перевірки логіна та пароля, якщо вони вірні програма дасть доступ до таких можливостей:

- перегляд інформації про використання ПК на сторінці веб-сайту;
- перегляд детальної інформації про використання ресурсів ПК;
- перегляд графіків використання та температури комплектуючих ПК;

Характеристики користувача.

Клієнт – це користувач, який повинен мати базові навички з роботою ПК.

Обмеження.

Лише зареєстрований користувач має доступ до роботи з системою.

Системою можливо користуватися на ОС сімейства Windows. Для роботи системи обов'язково повинен бути доступ до мережі інтернет.

Специфічні вимоги.

Вимоги до зовнішніх інтерфейсів.

Інтерфейси користувача – користувач може увійти в систему лише під зареєстрованим логіном та паролем. Апаратні інтерфейси – наявність комп'ютера.

Інтерфейси програмного забезпечення – наявність комп'ютера під керуванням ОС сімейства Windows.

Функціональні вимоги.

Клас користувачів – «Користувач».

Функціональне вимога 1 – Перегляд інформації про використання ПК на сторінці веб-сайту.

Функціональне вимога 2 – Перегляд детальної інформації про використання ресурсів ПК.

Функціональне вимога 3 – Перегляд графіків використання та температури комплектуючих ПК.

Функціональне вимога 4 – Перегляд повної інформації по процесах, які запущені на ПК.

Вимоги до робочих характеристик.

Кількість підключених до системи користувачів до 10000.

Велика кількість одночасних користувачів, які роблять операції змін даних у базі даних.

Атрибути якості програмного забезпечення.

Надійність – обов'язкова авторизація та аутентифікація користувачів.

2.3 Моделювання системи

На етапі системного аналізу або об'єктно-орієнтованого аналізу при розробці програмного забезпечення визначаються системні вимоги, ідентифікуються класи та визначаються взаємозв'язки між класами.

Три методи аналізу, які використовуються в поєднанні один з одним для об'єктно-орієнтованого аналізу, - це об'єктне моделювання, динамічне моделювання та функціональне моделювання[3].

Об'єктне моделювання.

Об'єктне моделювання розробляє статичну структуру програмної системи в термінах об'єктів. Воно визначає об'єкти, класи, в які об'єкти можуть бути згруповані, і взаємозв'язки між об'єктами. Також визначаються основні атрибути та операції, що характеризують кожен клас[6].

Процес об'єктного моделювання можна візуалізувати у вигляді наступних кроків:

- ідентифікація об'єктів та групування в класи;
- визначити взаємозв'язки між класами;
- створити діаграму об'єктної моделі користувача;
- визначити атрибути об'єктів користувача;
- визначити операції, які повинні виконуватись над класами; - динамічне моделювання.

Після того, як проаналізовано статичну поведінку системи, необхідно дослідити її поведінку по відношенню до часу та зовнішніх змін. Це і є метою динамічного моделювання.

Динамічне моделювання можна визначити як "спосіб опису того, як окремий об'єкт реагує на події, або внутрішні події, викликані іншими об'єктами, або зовнішні події, викликані зовнішнім світом"[7].

Процес динамічного моделювання можна візуалізувати у вигляді наступних кроків:

- ідентифікація станів кожного об'єкта;
- ідентифікація подій та аналіз застосовності дій;

- побудувати діаграму динамічної моделі, що складається з діаграм переходів станів;
- виразити кожен стан в термінах атрибутів об'єкта; - валідація побудованих діаграм переходів станів.

Функціональне моделювання

Функціональне моделювання є завершальним компонентом об'єктно-орієнтованого аналізу. Функціональна модель показує процеси, які виконуються всередині об'єкта, і те, як змінюються дані при їх переміщенні між методами. Вона конкретизує значення операцій об'єктного моделювання та дій динамічного моделювання. Функціональна модель відповідає діаграмі потоків даних традиційного структурного аналізу[8].

Процес функціонального моделювання можна візуалізувати у вигляді наступних кроків:

- ідентифікація всіх входів і виходів;
- побудувати діаграми потоків даних, що відображають функціональні залежності;
- вказати призначення кожної функції;
- визначити обмеження;
- вказати критерії оптимізації.

Діаграма випадків використання UML - це основна форма системних / програмних вимог до нової слаборозвиненої програми. У випадках використання вказується очікувана поведінка (що), а не точний спосіб її здійснення (як). Описані випадки використання можуть бути позначені як текстовим, так і візуальним поданнями (тобто діаграмою використання). Ключова концепція моделювання випадків використання полягає в тому, що це допомагає нам розробити систему з точки зору кінцевого користувача. Це ефективний прийом для комунікації поведінки системи з точки зору користувача шляхом вказівки всієї видимої зовні системи поведінки. Діаграма випадків використання зазвичай

проста. Вона не відображає деталі випадків використання: Вона лише узагальнює деякі взаємозв'язки між випадками використання, суб'єктами та системами. Вона не відображає порядок, у якому виконуються кроки для досягнення цілей кожного випадку використання. Як вже було сказано, схема використання може бути простою і містити лише кілька фігур. Якщо ваш містить більше 20 випадків використання, ви, мабуть, неправильно використовуєте схему використання. На (рис 2.2) нижче показано ієрархію діаграми UML та позиціонування діаграми використання UML. Як бачите, діаграми використання належать до сімейства поведінкових діаграм.

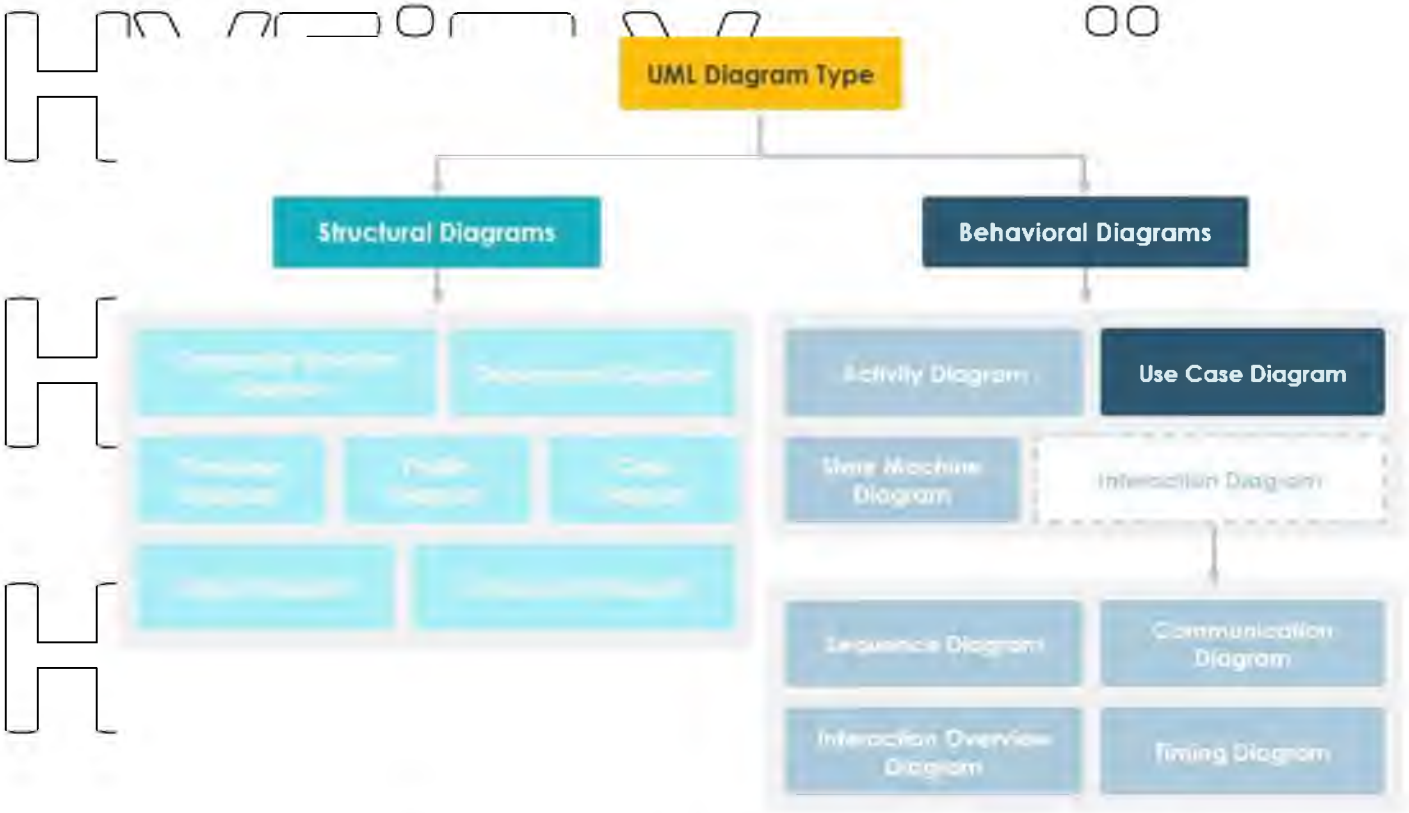


Рисунок 2.2 – ієрархія діаграм UML

Діаграма послідовностей - це тип діаграми взаємодії, оскільки вона описує, як - і в якому порядку - група об'єктів працює разом. Ці схеми використовуються розробниками програмного забезпечення та бізнес-професіоналами для розуміння вимог до нової системи або для документування існуючого процесу.

Діаграми послідовності іноді називають діаграмами подій або сценаріями подій. Зверніть увагу, що існує два типи діаграм послідовностей: діаграми UML та діаграми на основі коду.

Переваги діаграм послідовностей.

Діаграми послідовності можуть бути корисними посиланнями для підприємств та інших організацій. Спробуйте намалювати схему послідовності, щоб:

- представити деталі випадку використання UML;
- змоделювати логіку складної процедури, функції або операції;
- подивитись, як об'єкти та компоненти взаємодіють між собою для завершення процесу;
- спланувати та зрозуміти детальну функціональність існуючого або майбутнього сценарію.

У мові UML діаграми прецедентів якраз і дозволяють візуалізувати поведінку системи та класів, щоб користувачі могли збагнути як їх використовувати, а розробники – реалізувати відповідний елемент[9].

На діаграмі показані узагальнені варіанти використання програми користувачем(рис. 2.3).

Коли користувач запускає програму то він обов'язково повинен авторизуватись в ній. Якщо в нього немає облікового запису то для роботи з програмою йому потрібно зареєструватися тому, що програма працює лише з зареєстрованими користувачами.

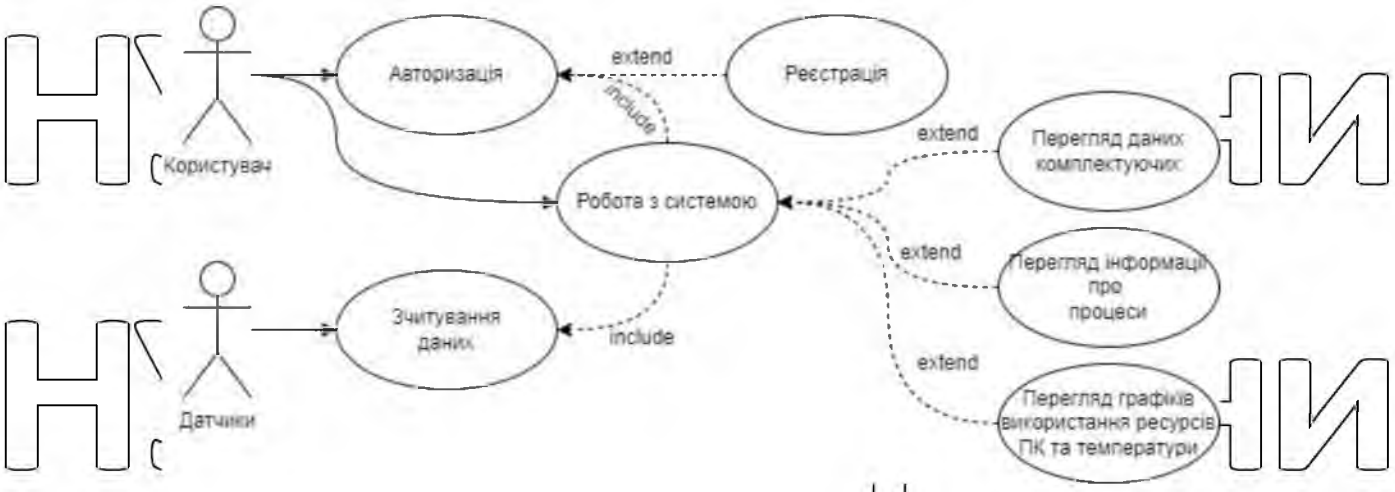


Рисунок 2.3 – Діаграма прецедентів

Якщо користувач авторизувався то програма починає зчитувати необхідну інформацію з датчиків і відправляти їх до серверу. Серед функціоналу системи для користувача можна виділити такі можливості:

- перегляд інформації про використання ПК на сторінці веб-сайту;
- перегляд детальної інформації про використання ресурсів ПК;
- перегляд графіків використання та температури комплектуючих ПК;
- перегляд повної інформації по процесах, які запущені на ПК.

3 РЕАЛІЗАЦІЯ КОМПЮТЕРНОЇ СИСТЕМИ

3.1 Розробка серверної частини системи

Back-end development - це розробка логіки на стороні сервера, яка забезпечує роботу веб-сайтів і додатків "з-за лаштунків". Вона включає в себе весь код, необхідний для створення бази даних сервера і додатків. Від міграції баз даних до інтеграції API та налаштування серверних технологій, які забезпечують роботу веб-сайту[10].

Для розробки серверної частини було обрано мову програмування C#.

C# ("Сі Шарп") - одна з найбільш швидко зростаючих, затребуваних і при цьому "зручних" мов програмування. Це модифікація фундаментальної мови C

від компанії Microsoft, покликана створити найбільш універсальний засіб для розробки програмного забезпечення для великої кількості пристроїв і операційних систем[11].

Коротка історія C# та її характеристики

C# - це об'єктно-орієнтована мова програмування. Вона була створена в період з 1998 по 2002 рік командою інженерів Microsoft під керівництвом Андерса Гейлсберга і Скотта Вільтаумота.

Мова входить до сім'ї C-подібних мов. Синтаксис наближений до Java і C++.

Її особливості:

- статистична типізація;
- підтримується поліморфізм;
- підтримується перевантаження операторів;
- доступна делегація, атрибути, події, узагальнені типи та анонімні функції.

Розробка Microsoft багато особливостей усладкувала у Delphi, Smalltalk і Java. Водночас творці нової мови виключили зі свого дітища багато практик і специфікацій, які вважаються "проблемними".

Основні переваги мови

C# популярна завдяки своїй "простоті". Простоти для сучасних програмістів і великих команд розробників, щоб ті могли в стислі строки створювати функціональні та продуктивні додатки. Цьому сприяють нетипові конструкції мови і специфічний синтаксис, що допомагає максимально органічно реалізувати намічені функції.

Популярність мови - ще одна значуща перевага. Велика кількість шанувальників C# сприяють її розвитку. Також це сприятливо впливає на зростання кількості вакансій, пов'язаних із розробкою мовою Microsoft.

Програмісти, добре знайомі з C#, затребувані в індустрії, незважаючи на їхню велику кількість, яка постійно збільшується.

Зрозумілий синтаксис C# помітно спрощує не тільки розробку як таку, а й інші важливі аспекти спільної роботи, наприклад, читання чужого коду. Це спрощує процес рефакторингу та виправлення помилок під час роботи над додатками у великих командах.

Також не можна не згадати низький поріг входження. C# - популярна і досить проста в освоєнні технологія. Уже за півроку можна набити руку в розробці та почати робити повноцінні програми.

Мова C# практично універсальна. Можна використовувати її для створення будь-якого ПЗ: просунутих бізнес-додатків, відеоігор, функціональних веб-застосунків, додатків для Windows, macOS, мобільних програм для iOS і Android.

Для реалізації серверної частини було обрано 3-рівневу архітектуру.

Трирівнева архітектура - це усталена архітектура програмних додатків, яка організовує додатки в три логічних і фізичних обчислювальних рівня: рівень представлення, або інтерфейс користувача; рівень програми, де обробляються дані; і рівень даних, де зберігаються і управляються дані, пов'язані з додатком[12].

Головною перевагою три рівневої архітектури є те, що оскільки кожен рівень працює на власній інфраструктурі, кожен рівень може розроблятися одночасно окремою командою розробників і може бути оновлений або масштабований за необхідності без впливу на інші рівні.

Протягом десятиліть трирівнева архітектура була переважаючою архітектурою для клієнт-серверних додатків. Сьогодні більшість три рівневих додатків є об'єктами модернізації з використанням хмарних технологій, таких як контейнери та мікросервіси, а також для міграції в хмару.

Три рівні в деталях

Презентаційний рівень (Presentation layer)

Рівень представлення - це користувальницький інтерфейс і комунікаційний рівень програми, де кінцевий користувач взаємодіє з програмою. Його основною метою є відображення інформації користувачеві та збір інформації від нього. Цей рівень верхнього рівня може працювати, наприклад, у веб-браузері, як настільний додаток або графічний інтерфейс користувача (GUI). Рівні вебпрезентації зазвичай розробляються з використанням HTML, CSS та JavaScript. Десктопні додатки можуть бути написані різними мовами залежно від платформи.

Рівень додатків (Business logic layer)

Рівень додатку, також відомий як логічний рівень або середній рівень, є серцем програми. На цьому рівні інформація, зібрана на рівні представлення, обробляється - іноді в порівнянні з іншою інформацією на рівні даних - з використанням бізнес-логіки, певного набору бізнес-правил. Рівень додатку також може додавати, видаляти або змінювати дані на рівні даних.

Рівень додатків зазвичай розробляється з використанням Python, Java, Perl, PHP або Ruby і взаємодіє з рівнем даних за допомогою викликів API.

Рівень даних (Data access layer)

Рівень даних, який іноді називають рівнем бази даних, рівнем доступу до даних або внутрішнім рівнем, - це місце, де зберігається і управляється інформація, що обробляється додатком. Це може бути реляційна система управління базами даних, така як PostgreSQL, MySQL, MariaDB, Oracle, DB2, Informix або Microsoft SQL Server, або сервер баз даних NoSQL, такий як Cassandra, CouchDB або MongoDB.

У трирівневому додатку вся комунікація проходить через рівень додатку. Рівень представлення та рівень даних не можуть взаємодіяти безпосередньо один з одним.

3.1.1 Розробка рівню даних (DAL)

Було створено 6 основних моделей, які ви бачите на рисунку 3.1.

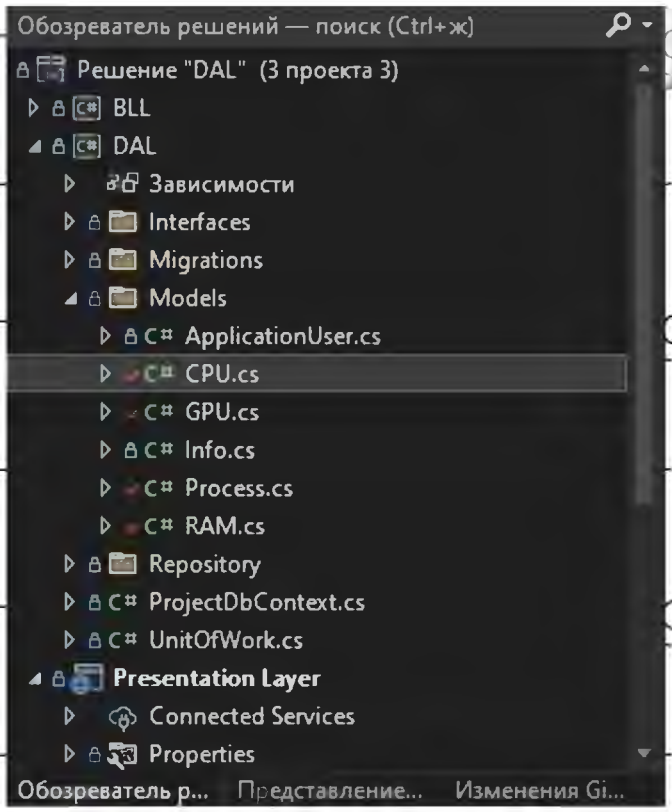


Рисунок 3.1 – Основні моделі

Для прикладу наведемо модель CPU яка має такі властивості(property)(Лістинг 3.1):

- Id – унікальний номер;
- Name – Назва процессора;
- Utilization – Загрузка процесора в %;
- Temperature – Температура процесора.

Лістинг 3.1 – Модель класу CPU

```
public class CPU
    {
        public int Id { get; set; }
public string Name { get; set; }        public
double Utilization { get; set; }        public
double Temperature { get; set; }
    }
```

Для доступу до даних ми застосуємо патерн Repository.

Що ми робимо традиційно? Ми отримуємо доступ до нашої бази даних безпосередньо, без використання будь-якого проміжного рівня або рівня доступу до даних (Data Access Layer, DAL).

Тісний зв'язок логіки бази даних з бізнес-логікою робить додатки складними, їх важко тестувати та розширювати. Прямий доступ до даних в бізнес-логіці викликає багато проблем, таких як складність виконання модульного тестування бізнес-логіки, неможливість тестування бізнес-логіки без залежностей від зовнішніх систем, таких як база даних, і дублювання коду доступу до даних на всьому бізнес-рівні. Сподіваюся, тепер ми знаємо, навіщо нам потрібні патерни проектування. Існує багато типів патернів проектування, але зараз ми використовуємо паттерн проектування сховища(Repository).

Що таке патерн проектування сховища?

За визначенням, патерн проектування сховища в C# є посередником між доменом та рівнями відображення даних, використовуючи інтерфейс, подібний до колекції, для доступу до об'єктів домену. Патерн проектування сховища відокремлює логіку доступу до даних і відображає її на сутності в бізнес-логіці. Він працює з сутностями домену і виконує логіку доступу до даних. У патерні "Сховище" сутності домену, логіка доступу до даних та бізнес-логіка взаємодіють між собою за допомогою інтерфейсів. Це приховує деталі доступу до даних від бізнес-логіки[13].

Переваги патерну проектування сховища

- спрощується тестування контролерів, оскільки тестовий фреймворк не потрібно запускати проти фактичного коду доступу до бази даних;
- repository Design Pattern відокремлює власне базу даних, запити та іншу логіку доступу до даних від решти додатку;
- бізнес-логіка може отримати доступ до об'єкта даних, не маючи знань про базову архітектуру доступу до даних;

– бізнес-логіка не знає, чи використовує додаток LINQ до SQL або ADO.NET. В майбутньому базові джерела даних або архітектура можуть бути змінені без впливу на бізнес-логіку;

– стратегія кешування для джерела даних може бути централізована;

– централізація логіки доступу до даних, що полегшує супроводження коду.

Спочатку створимо інтерфейси для наших репозиторіїв, які ми можемо побачити на рисунку 3.2.

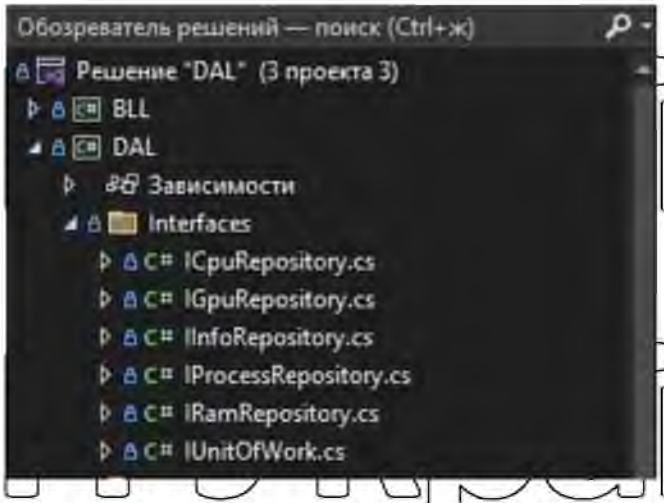


Рисунок 3.2 – Інтерфейси для реалізації репозиторіїв

Для прикладу наведемо інтерфейс IInfoRepository(Лістинг 3.2) який має такі методи:

- GetAll() – метод який повертає всі записи в базі даних типу Info;
- IQueryable<Info> GetAllWithDetails() – метод який повертає всі записи з БД типу Info включно з вкладеною інформацією;
- Task<Info> GetByIdAsync(int id) – метод який повертає 1 запис типу Info за його унікальним номером;
- Task<bool> AddAsync(Info info) – метод який робить запис до БД;
- Task<bool> UpdateAsync(Info info) – метод для обновлення вже існуючого запису;

- Task<bool> DeleteAsync(Info info) – метод для видалення запису по готовій моделі;
- Task<bool> DeleteByIdAsync(int id) – метод для видалення запису з БД за унікальним номером;
- Task<Info> GetByIdWithDetailsAsync(int id) – метод для отримання запису з вкладеною інформацією;

Реалізацію репозиторія InfoRepository можна побачити з лістингу 3.3.

Лістинг 3.2 – Інтерфейс IInfoRepository

```
public interface IInfoRepository
    {
        IQueryable<Info> GetAll();
        IQueryable<Info> GetAllWithDetails();
        Task<Info> GetByIdAsync(int id);
        Task<bool> AddAsync(Info info);
        Task<bool> UpdateAsync(Info info);
        Task<bool> DeleteAsync(Info info);
        Task<bool> DeleteByIdAsync(int id);
        Task<Info> GetByIdWithDetailsAsync(int id);
    }
```

Лістинг 3.3 – Клас InfoRepository

```
public class InfoRepository:IInfoRepository
    {
        private readonly ProjectDbContext _context;
public InfoRepository(ProjectDbContext context)
        {
            _context = context;
        }
        public async Task<bool> AddAsync(Info info)
        {
```

```
            await _context.Info.AddAsync(info);
        await _context.SaveChangesAsync();
        return true;
        }
        public async Task<bool> DeleteAsync(Info info)
        {
            var item = await _context.Info.FirstOrDefaultAsync(x
=> x.Id == info.Id);
        if (item != null)
            {
                _context.Info.Remove(info);
        return true;
            }
        return false;
        }
        public async Task<bool> DeleteByIdAsync(int id)
        {
            var item = await _context.Info.FirstOrDefaultAsync(x
=> x.Id == id);
            if (item != null)
            {
                _context.Info.Remove(item);
        await _context.SaveChangesAsync();
        return true;
            }
    return false;
        }
        public IQueryable<Info> GetAll()
        {
            return _context.Info;
        }
        public IQueryable<Info> GetAllWithDetails()
        {
            return _context.Info.Include(x => x.RAM).Include(x =>
x.GPU).Include(x => x.CPU).Include(x => x.Processes);
    }
```

```
        public async Task<Info> GetByIdAsync(int id)
        {
            return await _context.Info.Include(x =>
x.RAM).Include(x => x.GPU).Include(x => x.CPU).Include(x =>
x.Proceses).FirstOrDefaultAsync(x => x.Id == id);
        }
        public async Task<Info> GetByIdWithDetailsAsync(int id)
        {
            return await _context.Info.Include(x =>
x.RAM).Include(x => x.GPU).Include(x => x.CPU).Include(x =>
x.Proceses).FirstOrDefaultAsync(x => x.Id == id);
        }
        public async Task<bool> UpdateAsync(Info info)
        {
            Info item = await GetByIdAsync(info.Id);
await _context.SaveChangesAsync();            return
item != null;
        }
    }
```

Для роботи того щоб створити та під’єднати базу даних нам спочатку потрібно додати до проекту відповідний пакети для роботи з БД(рис 3.3):

- Microsoft.EntityFrameworkCore;
- Microsoft.EntityFrameworkCore.Design;
- Microsoft.EntityFrameworkCore.InMemory;
- Microsoft.EntityFrameworkCore.SqlServer;
- Microsoft.EntityFrameworkCore.Tools, - Microsoft.Extensions.Identity.Stores.

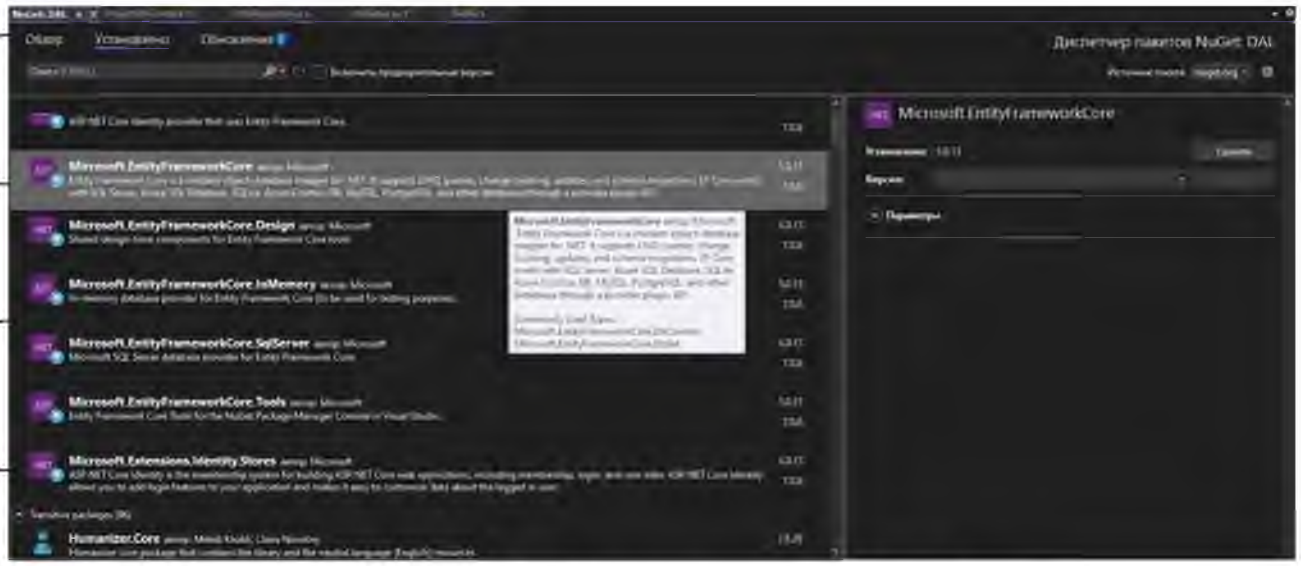


Рисунок 3.3 – Пакети для роботи з БД

Щоб створити БД нам потрібно реалізувати клас ProjectDbContext який ми можемо бачити з лістингу 3.4.

Лістинг 3.4 – Клас ProjectDbContext

```
public class
ProjectDbContext:IdentityDbContext<ApplicationUser>
    {
        public DbSet<CPU> CPUs { get; set; }
public DbSet<GPU> GPUs { get; set; }        public
DbSet<Info> Info { get; set; }        public
DbSet<Process> Processes { get; set;}
public DbSet<RAM> RAMs { get; set; }

        public ProjectDbContext(DbContextOptions<ProjectDbContext>
options) : base(options)
        {
        }
    }
```

Після чого в командній строці нам потрібно включити міграції (Лістинг 3.5).

Лістинг 3.5 – Активація міграцій

```
-enable-migrations
```

Коли міграції включені нам потрібно створити міграцію та БД за допомогою командної строки (лістинг 3.6).

Лістинг 3.6 – Створення міграції та БД

```
add-migration    "newmigration"
update-database
```

Після виконання команд створюється база даних(рис 3.4) з потрібними нам таблицями та ключами з якою ми будемо працювати за допомогою наших репозиторіїв.

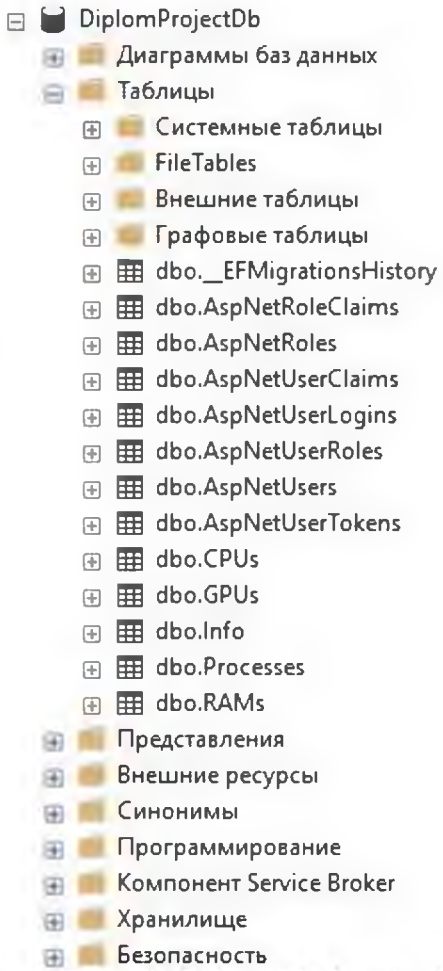


Рисунок 3.4 – Таблиці бази даних

Отже, коли репозиторії і БД вже створені ми створюємо клас UnitOfWork для того щоб застосувати патерн UnitOfWork.

Шаблон «UnitOfWork» використовується для об'єднання однієї або декількох операцій (зазвичай CRUD-операцій бази даних) в одну транзакцію або "одиницю роботи" таким чином, що всі операції або проходять, або не проходять як одна одиниця. Простими словами можна сказати, що для конкретної дії користувача, скажімо, бронювання на веб-сайті, всі операції, такі як вставка/оновлення/видалення і так далі, виконуються в одній єдиній транзакції, замість того, щоб виконувати кілька операцій з базою даних. Це означає, що одна одиниця роботи тут включає в себе операції вставки/оновлення/видалення, і все це в одній транзакції, так що всі операції або проходять, або не проходять як одна одиниця[14].

3.1.2 Розробка Бізнес-рівню (Business logic layer)

У програмуванні рівень бізнес-логіки (Business Logic Layer, BLL) служить посередником для обміну даними між рівнем представлення та рівнем доступу до даних (Data Access Layer, DAL). Рівень бізнес-логіки обробляє бізнес-правила, розрахунки та логіку в додатку, які визначають його поведінку. Тобто, BLL визначає, як використовуються дані з бази даних і що вони можуть і не можуть робити в самому додатку[15].

Розуміння рівня бізнес-логіки.

Рівень бізнес-логіки обробляє бізнес-правила, обчислення і логіку в додатку, які визначають його поведінку. Тобто BLL визначає, як використовуються дані з бази даних і що вони можуть і не можуть робити в самому додатку.

Таким чином, рівень бізнес-логіки управляє зв'язком між базою даних і рівнем представлення - іноді його називають інтерфейсом кінцевого користувача. Він є частиною багаторівневої архітектури програмної інженерії, де різні функції фізично розділені.

Для початку створимо DTO моделі(Data-Transfer-Object)(Рис 3.5).

Data-Transfer-Object - це об'єкт, який використовується для інкапсуляції даних і відправки їх з однієї підсистеми програми в іншу.

DTO найчастіше використовуються рівнем служб в додатку N-рівня для передачі даних між собою і рівнем PL. Основна перевага тут полягає в тому, що це зменшує кількість даних, які потрібно пересилати по мережі з розподілених додатках.

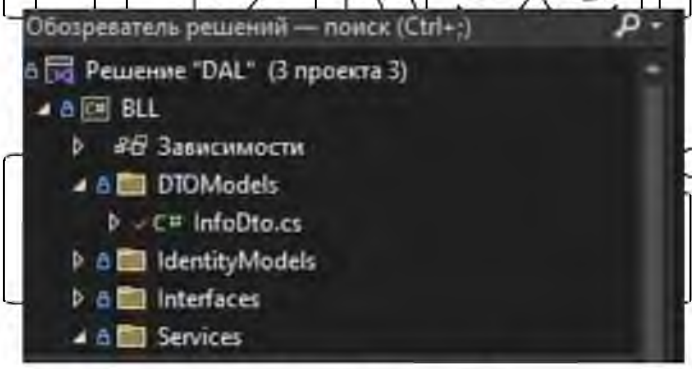


Рисунок 3.5 – DTO model

Коли розроблена DTO модель нам потрібно зробити мапер для того щоб привести нашу модель яка приходить до BLL з DAL, так щоб не виносити приватні зміні до вищого рівня де їх можна буде перехопити або дізнатись їх.

Об'єктні та реляційні бази даних мають різні механізми структурування даних. Багато частин об'єкта, таких як колекції та успадкування, відсутні в реляційних базах даних. Коли ви будуєте об'єктну модель з великою кількістю бізнес-логіки, корисно використовувати ці механізми для кращої організації даних та поведінки, яка з ними пов'язана. Це призводить до того, що об'єктна і реляційна схеми не збігаються, тобто об'єктна схема не збігається з реляційною схемою. Вам все одно потрібно передавати дані між двома схемами, і ця передача даних стає складністю сама по собі. Якщо об'єкти в пам'яті знають про структуру реляційної бази даних, то зміни в одному з них мають тенденцію поширюватися на інший.

Data Mapper - це шар програмного забезпечення, який відокремлює об'єкти в пам'яті від бази даних. Його обов'язком є передача даних між ними, а також

ізоляція їх один від одного. За допомогою Data Mapper об'єкти в пам'яті навіть не повинні знати, що існує база даних; їм не потрібен інтерфейсний код SQL, і, звичайно, не потрібно знати схему бази даних. (Схема бази даних завжди не знає про об'єкти, які її використовують.) Оскільки це форма Mapper (473), Data Mapper сам по собі навіть невідомий доменному рівню[16].

Для того щоб застосувати мапер, спочатку нам потрібно інсталювати потрібний пакет(Рис. 3.6).

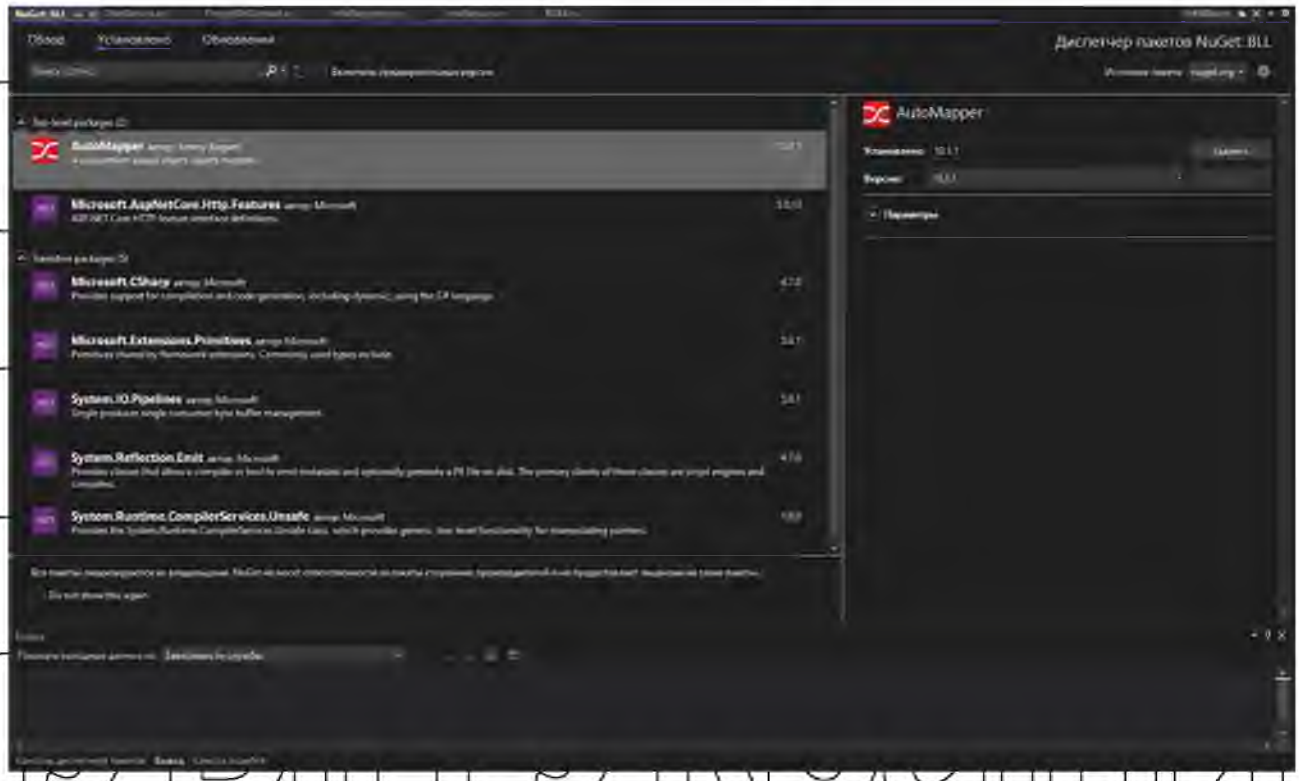


Рисунок 3.6 – Установка пакета AutoMapper

Після інсталяції пакета нам потрібно створити клас AutoMapperProfile реалізацію класа можна побачити з лістингу 3.7.

Лістинг 3.7 – Клас AutoMapperProfile

```
using AutoMapper; using BLL.DTOModels; using DAL.Models; using System;
```

```
using System.Collections.Generic; using
System.Text;

namespace
BLL
{
    public class AutoMapperProfile:Profile
    {
        public AutoMapperProfile()
        {
            CreateMap<Info, InfoDto>().ReverseMap();
        }
    }
}
```

Після створення всього що було наведено вище нам потрібно створити сервіси, які реалізують потрібну бізнес логіку для прикладу наведемо InfoService(Лістинг 3.8)

InfoService містить в собі такі методи:

- Task<bool> AddAsync(InfoDto infoDto) – отримує DTO модель після чого мапить її в Info перевіряє її і після чого вже передає до DAL і створює новий запис до БД;
- Task<bool> UpdateAsync(InfoDto infoDto,string userid) – цей метод отримує DTO модель та унікальний номер користувача, перевіряє і після цього вже оновлює вже існуючий запис;
- Task<IEnumerable<InfoDto>> GetAllInfoByUserId(string userId) – цей метод за унікальним номером користувача повертає всі записи типу Info цього користувача;
- Task<IEnumerable<InfoDto>> GetAllAsync() – цей метод дозволяє вигрузити всі дані типу Info;
- Task<bool> DeleteAsync(int id,string userid) – даний метод за унікальним номером користувача та запису типу Info видаляє запис з БД;

Лістинг 3.8 – Клас AutoMapperProfile

```
public class InfoService : IInfoService
{
    private readonly IUnitOfWork _unitOfWork;
    private readonly IMapper _mapper;
    private readonly UserManager<ApplicationUser> _userManager;
    public InfoService(IUnitOfWork unitOfWork, IMapper
mapper, UserManager<ApplicationUser> userManager)
    {
        _unitOfWork = unitOfWork;
        _mapper = mapper;
        _userManager = userManager;
    }
    public async Task<bool> AddAsync(InfoDto info)
    {
        var user = await
_userManager.FindByIdAsync(info.UserId);
        if (user == null)
        {
            throw new ArgumentException("User not
found");
        }
        if (await
_unitOfWork.InfoRepository.AddAsync(_mapper.Map<Info>(info)))
        {
            return true;
        }
        else
        {
            return false;
        }
    }
    public async Task<bool> DeleteAsync(int id, string
```

```
userid)
        {
            var user = await
_userManager.FindByIdAsync(userid);
            if (user == null)
            {
                throw new ArgumentException("User not
found");
            }
            var roles = await
_userManager.GetRolesAsync(user);
            Info info = await
_unitOfWork.InfoRepository.GetByIdAsync(id);
            if (info == null)
            {
                throw new ArgumentException($"{id}-Info.id
dont exist");
            }
            if (info.UserId == userid ||
roles.Contains("admin"))
            {
                if
(await
_unitOfWork.InfoRepository.DeleteAsync(info))
                {
                    return true;
                }
            }
            return false;
        }
        public async Task<IEnumerable<InfoDto>> GetAllAsync()
        {
            var info = await
```

```
        var info =
    unitOfWork.InfoRepository.GetAll().ToListAsync();
        return  mapper.Map<IEnumerable<InfoDto>>(info);
    }
    public async Task<IEnumerable<InfoDto>>
GetAllInfoByUserId(string userId)
    {
        var info = await
_unitOfWork.InfoRepository.GetAllWithDetails().Where(x => x.UserId
== userId).ToListAsync();          return
    _mapper.Map<IEnumerable<InfoDto>>(info);
    }
    public Task<bool> UpdateAsync(InfoDto infoDto, string
userid)
    {
        throw new NotImplementedException();
    }
}
```

3.1.3 Розробка презентаційного рівня (Presentation layer)

Презентаційний рівень – це найвищий рівень програми, де користувач виконує свою діяльність. Розглянемо приклад будь-якої програми, де користувачеві потрібно заповнити форму. Ця форма є нічим іншим, як рівнем презентації. У програмах Windows Windows Forms є рівнем презентації, а у веб-програмах веб-форма належить до рівня презентації. В основному перевірка введених даних користувача та обробка правил виконуються на цьому рівні.

При розробці презентаційного рівня першим ділом потрібно налаштувати всі залежності щоб прокинути зв’язки з BLL для того, щоб ми могли використовувати сервіси як ми розробляли. Для того щоб прокинути зв’язки нам потрібно в класі «Startup.cs» добавить такі строчки коду(лістинг 3.9).

Лістинг 3.9 – Прокидування зав’язків

```
services.AddScoped<ICpuRepository, CpuRepository>();
services.AddScoped<IGpuRepository, GpuRepository>();
services.AddScoped<IRAMRepository,
RamRepository>();
services.AddScoped<IInfoRepository,
InfoRepository>();
services.AddScoped<IProcessRepository,
ProcessRepository>();
services.AddScoped<IUnitOfWork, UnitOfWork>();
services.AddScoped<IInfoService, InfoService>();
services.AddScoped<IUserService, UserService>();
```

Після того як зв’язки прокинуті можна створювати Web Api контролери на які будуть приходити запити і відправляться дані для прикладу наведемо метод «GetAllByUserId»(Лістинг 3.10)

Лістинг 3.10 – Метод контролера GetAllByUserId

```
[Authorize]
[HttpGet("user/{userid}")]
public async Task<ActionResult<IEnumerable<InfoDto>>>
GetAllByUserId(string userid)
        {
            var infos = await
_infoService.GetAllInfoByUserId(userid);
if(infos == null)
            {
                return NotFound();
            }
            return Ok(infos);
        }
```

Для авторизації користувача було застосовано JWT-токен.

JSON Web Token - це відкритий галузевий стандарт, який використовується для обміну інформацією між двома об'єктами, як правило, клієнтом (наприклад, інтерфейсом вашого додатку) і сервером (внутрішньою частиною вашого додатку).

Вони містять об'єкти JSON, які містять інформацію, якою потрібно обмінюватися. Кожен JWT також підписується за допомогою криптографії (хешування), щоб гарантувати, що вміст JSON (також відомий як вимоги JWT) не може бути змінений клієнтом або зловмисником.

Наприклад, коли ви входите в систему Google, Google видає JWT, який містить наступні твердження / корисне навантаження JSON:

Використовуючи вищевказану інформацію, клієнтська програма, яка використовує вхід в Google, точно знає, хто є кінцевим користувачем(Рис 3.7).


```
{
    "iss": "https://accounts.google.com",
    "azp": "1234987819200.apps.googleusercontent.com",
    "aud": "1234987819200.apps.googleusercontent.com",
    "sub": "107691503500061507151130823 67",
    "at_hash": "HK6E_P6Dh8Y93mRNtsDB1Q",
    "email": "jsmith@example.com",
    "email_verified": "true",
    "iat": 1353601026,
    "exp": 1353604926,
    "nonce": "0394852-3190485-2490358",
    "hd": "example.com"
}
```


Рисунок 3.7 – Приклад вмісту JWT-токена

Вам може бути цікаво, чому сервер авторизації не може просто відправити інформацію у вигляді простого JSON-об'єкта і навіщо йому потрібно перетворювати її в "токен".

Якщо сервер аутентифікації відправляє інформацію у вигляді простого JSON, API клієнтських додатків не матимуть можливості перевірити правильність вмісту, який вони отримують. Зловмисник може, наприклад, змінити ідентифікатор користувача (підпункт у наведеному вище прикладі JSON), і API додатку не матиме можливості дізнатися, що це сталося.

У зв'язку з цією проблемою безпеки, серверу авторизації необхідно передавати цю інформацію таким чином, щоб її можна було перевірити клієнтською програмою, і саме тут з'являється поняття "токен".

Простіше кажучи, токен - це рядок, що містить деяку інформацію, яку можна безпечно перевірити. Це може бути випадковий набір алфавітноцифрових символів, які вказують на ідентифікатор в базі даних, або це може бути закодований JSON, який може бути самостійно перевірений клієнтом (відомий як JWT).

JWT складається з трьох частин(Рис 3.8):

- заголовок: Складається з двох частин:
  - Алгоритм підпису, який використовується.
  - Тип токена, який, в даному випадку, здебільшого є "JWT".
- корисне навантаження. Корисне навантаження містить вимоги або JSON-об'єкт;
- підпис: Рядок, який генерується за допомогою криптографічного алгоритму, який може бути використаний для перевірки цілісності корисного навантаження JSON.

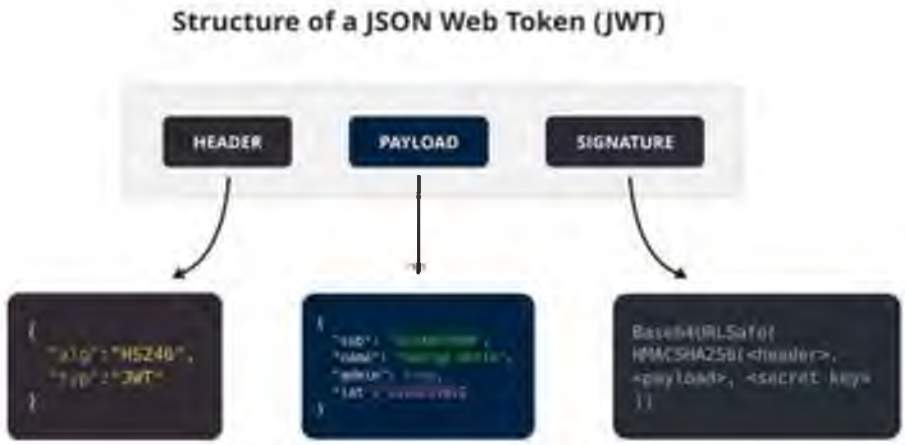


Рисунок 3.8 – Структура JWT-токена

Також було додано swagger щоб створити документацію та перевірити коректність роботи нашої серверної частини.

Swagger - це набір інструментів, які допомагають описувати API. Завдяки йому користувачі та машини краще розуміють можливості REST API без доступу до коду. За допомогою Swagger можна швидко створити документацію та надіслати її іншим розробникам або клієнтам[17].

Роботу Swagger можна побачити на рисунку 3.9.

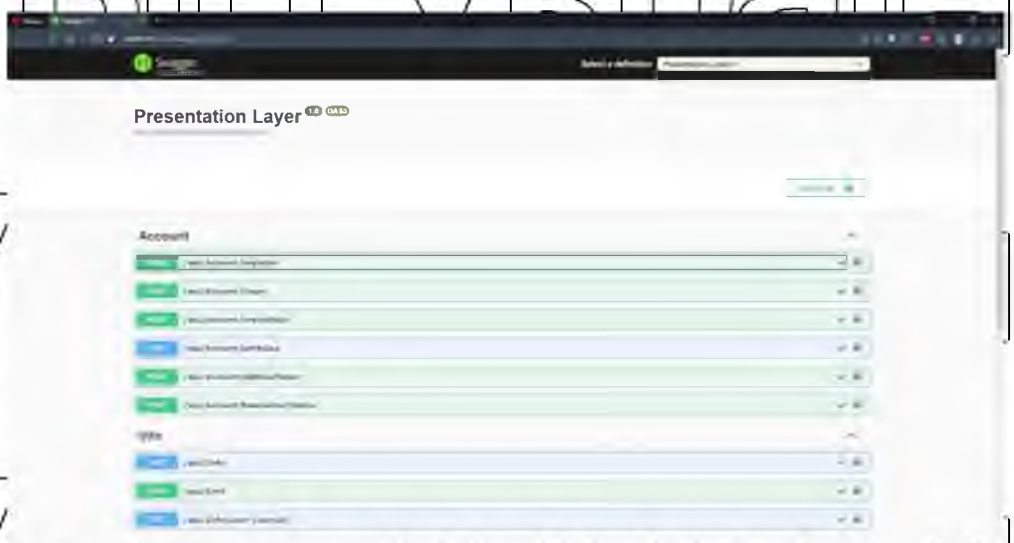


Рисунок 3.9 – Swagger

3.2 Розробка клієнтської частини системи

Значна частина Інтернету базується на моделі клієнт-сервер. У цій моделі користувацькі пристрої взаємодіють через мережу з центрально розташованими серверами для отримання необхідних їм даних, замість того, щоб спілкуватися один з одним. Пристрої кінцевих користувачів, такі як ноутбуки, смартфони та настільні комп'ютери, вважаються "клієнтами" серверів, як якщо б вони були клієнтами, які отримують послуги від компанії. Клієнтські пристрої надсилають запити на сервери для отримання веб-сторінок або додатків, а сервери надають відповіді.

Модель клієнт-сервер використовується тому, що сервери, як правило, більш потужні і надійні, ніж користувацькі пристрої. Вони також постійно підтримуються і зберігаються в контрольованому середовищі, щоб переконатися, що вони завжди ввімкнені і доступні; хоча окремі сервери можуть вийти з ладу, зазвичай є інші сервери, які створюють їх резервну копію. Тим часом, користувачі можуть вмикати та вимикати свої пристрої, втрачати або ламати їх, і це не повинно впливати на Інтернет-послуги для інших користувачів.

Сервери можуть обслуговувати декілька клієнтських пристроїв одночасно, і кожен клієнтський пристрій надсилає запити до декількох серверів у процесі доступу до Інтернету та перегляду веб-сторінок.

Кілька клієнтів і серверів взаємодіють між собою як зображено на рисунку 3.10.

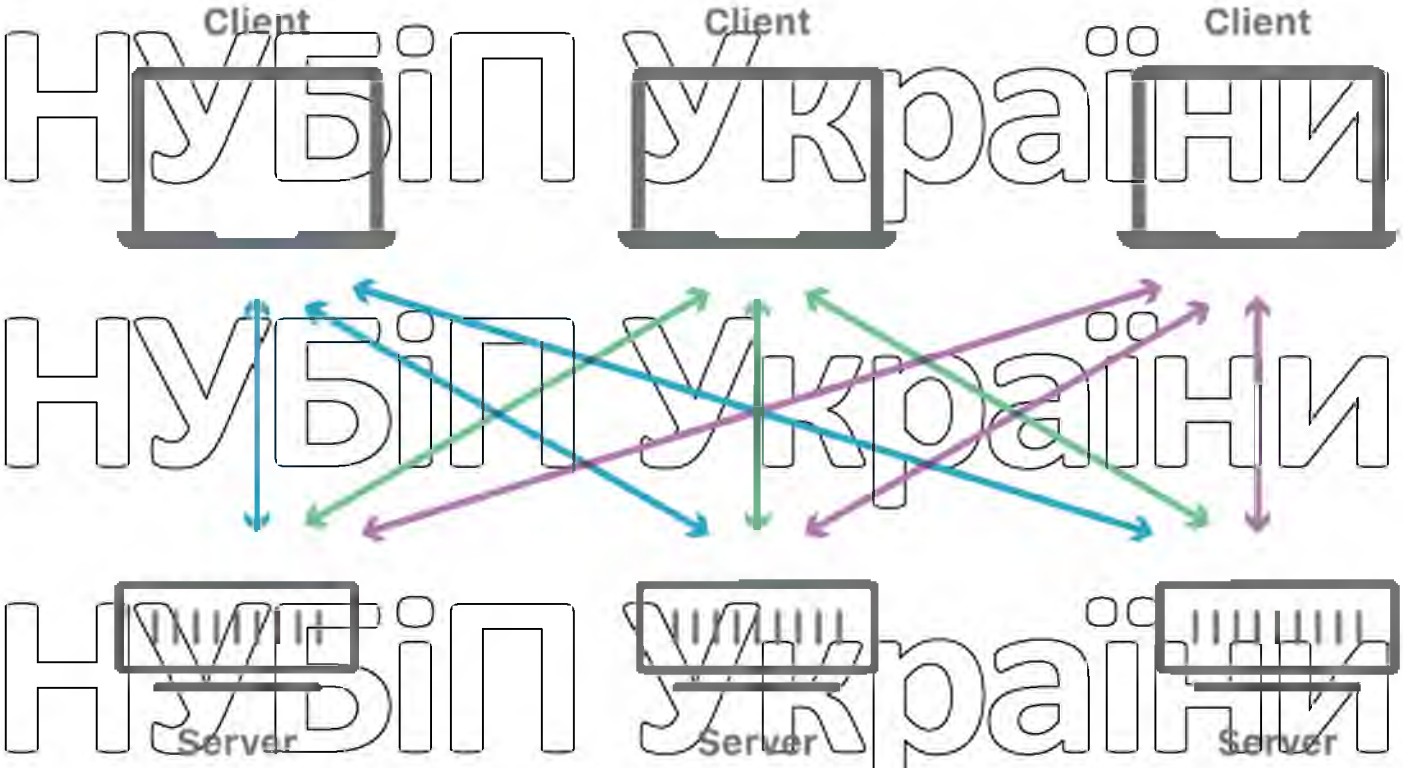


Рисунок 3.10 – Взаємодія клієнтів з сервером.

3.2.1 Розробка десктопної програми

Було розроблено десктопну програму за допомогою Windows Forms.

Windows Forms - це фреймворк інтерфейсу користувача для створення десктопних додатків Windows. Вона забезпечує один з найпродуктивніших способів створення настільних додатків на основі візуального конструктора, що входить до складу Visual Studio. Така функціональність, як розміщення візуальних елементів управління за допомогою перетягування, спрощує створення настільних додатків.

За допомогою Windows Forms ви розробляєте графічно насичені додатки, які легко розгортати, оновлювати та працювати в автономному режимі або при підключенні до Інтернету. Програми Windows Forms мають доступ до локального обладнання та файлової системи комп'ютера, на якому запущено програму.

Дана програма зчитує дані з потрібних датчиків:

- температура процесора;

- нагрузку процесора;
- загруженість відеокарти;
- температуру відеокарти;
- основні процеси які загружають систему.

Програма зчитує дані з комп'ютера постійно і раз в 5 хвилин відправляє усереднені дані. Дані мають такий вигляд в БД, як бачимо на рисунку 3.11.

Результаты | Сообщения

	Id	Name	Utilization	Temperature
1	1	intel i4	45	60
2	2	AMD Ryzen 7 4800H	11,5036507606506	0
3	3	AMD Ryzen 7 4800H	11,4581840688532	0
4	4	AMD Ryzen 7 4800H	17,3256135420366	0
5	5	AMD Ryzen 7 4800H	15,6634244918823	0
6	6	AMD Ryzen 7 4800H	16,1406489285556	0
7	7	AMD Ryzen 7 4800H	15,1987749446522	0
8	8	AMD Ryzen 7 4800H	32,2149549681565	76,7887931034483
9	9	AMD Ryzen 7 4800H	14,949142043655	64,8513513513514
10	10	AMD Ryzen 7 4800H	21,3720578143471	75,125
11	11	AMD Ryzen 7 4800H	16,00365064083	67,3782051282051
12	12	AMD Ryzen 7 4800H	14,2283452538883	65,1911764705882
13	13	AMD Ryzen 7 4800H	13,3604875171886	62,5551470588235
14	14	AMD Ryzen 7 4800H	14,0644949464237	63,5661764705882
15	15	AMD Ryzen 7 4800H	15,2894411367529	63
16	16	AMD Ryzen 7 4800H	16,0840476260466	63,8014705882353
17	17	AMD Ryzen 7 4800H	15,7814411275527	64,1801470588235

Рисунок 3.11 – Дані про процесор в БД

Створена наша десктопна програма працює в фоновому режимі(рис 3.12). Для того щоб запустити програму потрібно здійснити подвійний клік на ярлик програми після чого з'явиться вікно авторизації(рис 3.13). Якщо в користувача ще немає створеного аккаунт він також може створити собі новий аккаунт за допомогою кнопки «зареєструватися». Після натискання на кнопку відкриється нове вікно де потрібно зареєструватися(рис 3.14).

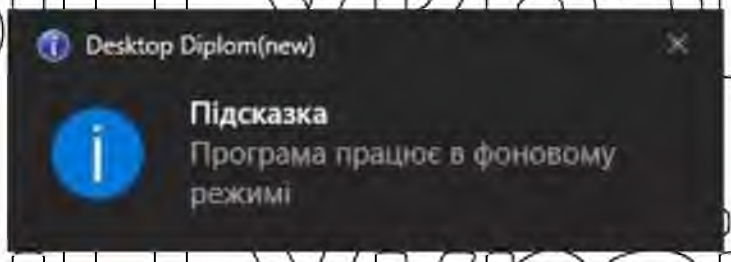


Рисунок 3.12 – Сповіщення про те що програма працює в фоновому режимі

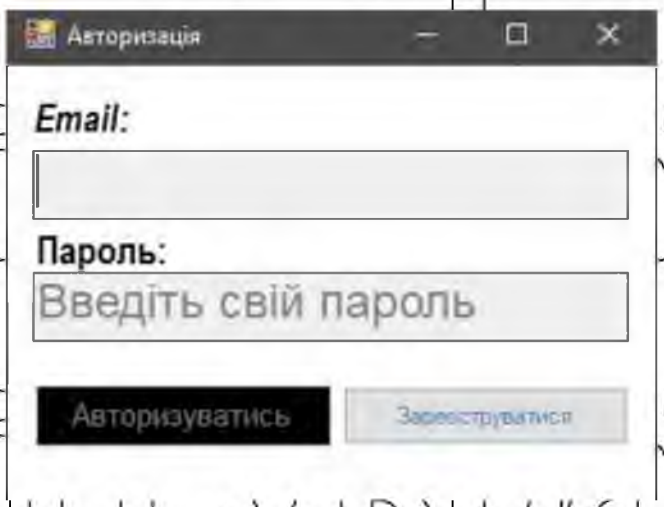


Рисунок 3.13 – Форма авторизації

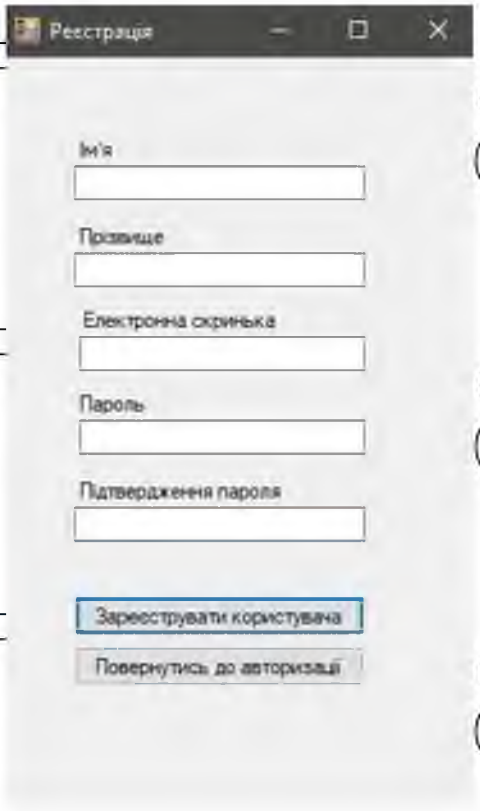


Рисунок 3.14 – Форма реєстрації

При авторизації програма отримує JWT токен, зберігає його і декомпілює щоб отримати унікальний номер користувача, щоб далі саме за ним відправляти дані до серверу(лістинг 3.11).

Лістинг 3.11 – Декомпіляція JWT токена

```
public void JwtDecode(string token)
        {
            var handler = new JwtSecurityTokenHandler();
var decodedValue = handler.ReadJwtToken(token);
            Program.userid = decodedValue.Subject;
        }
```

Також коли програма працює в фоновому режимі ми можемо її закрити або відкрити сайт за допомогою правого кліку по значку на панелі інструментів(Рисунок 3.15).

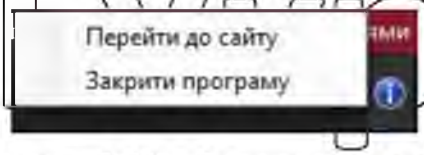


Рисунок 3.15 – Іконка програми

3.2.2 Розробка сайту

Фронтенд – це рівень представлення, це та частина програми, яку може бачити користувач. Наприклад, у вигляді графічного інтерфейсу користувача (скорочено: GUI)[18]. Фронтенд часто складається з частин які показані на рисунку 3.16.

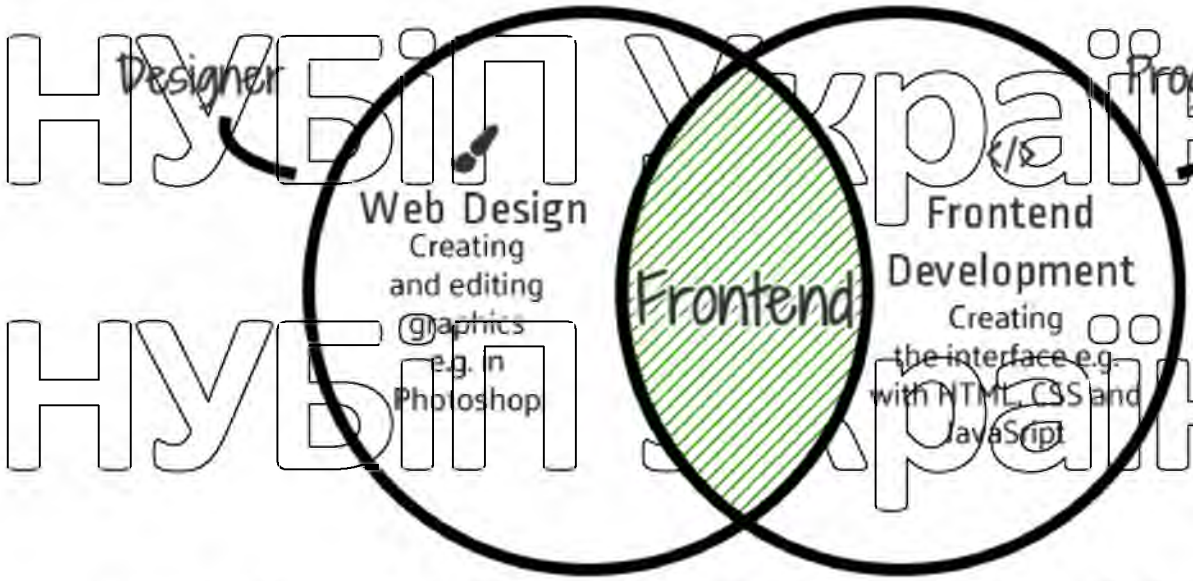


Рисунок 3.16 – Частини фронтенду

В цілому можна відзначити, що, говорячи про веб-сайт, ви також говорите про його інтерфейс. Все, що ви можете побачити на веб-сайті, - шрифти, кольори, меню, кнопки, таблиці та багато іншого – було закодовано та/або зібрано за допомогою html, css. JavaScript, а, наприклад, графіка для цього сайту була створена за допомогою Photoshop. Завдяки такому поєднанню кодування та дизайну у вас є широкий спектр можливостей для індивідуального дизайну для програми. Фронтенд закриває розрив між інтерфейсом і діями, які виконуються у фоновому режимі.

Це робить можливим взаємодію користувача з внутрішньою частиною, що в іншому випадку було б складно або вимагало б високого рівня спеціалізованого ноу-хау.

Для розробки сайту було використано фреймворк Angular.

Angular – це платформа і фреймворк для побудови односторінкових клієнтських додатків з використанням HTML і TypeScript. Angular написаний на мові TypeScript. Він реалізує основну і додаткову функціональність у вигляді набору бібліотек TypeScript, які ви імпортуєте в свої додатки.

Архітектура Angular-додатків спирається на певні фундаментальні концепції. Основними будівельними блоками фреймворку Angular є

Angularкомпоненти, які організовані в NgModules. NgModules збирають пов’язаний код у функціональні набори; Angular-додаток визначається набором NgModules. Додаток завжди має принаймні кореневий модуль, який забезпечує завантаження, і, як правило, має набагато більше функціональних модулів.

- компоненти визначають представлення, які є наборами екранних елементів, які Angular може вибирати з-поміж них та змінювати відповідно до логіки вашої програми та даних;
- компоненти використовують сервіси, які надають специфічну функціональність, не пов’язану безпосередньо з представленнями. Постачальники послуг можуть бути введені в компоненти як залежності, що робить ваш код модульним, багаторазовим та ефективним.

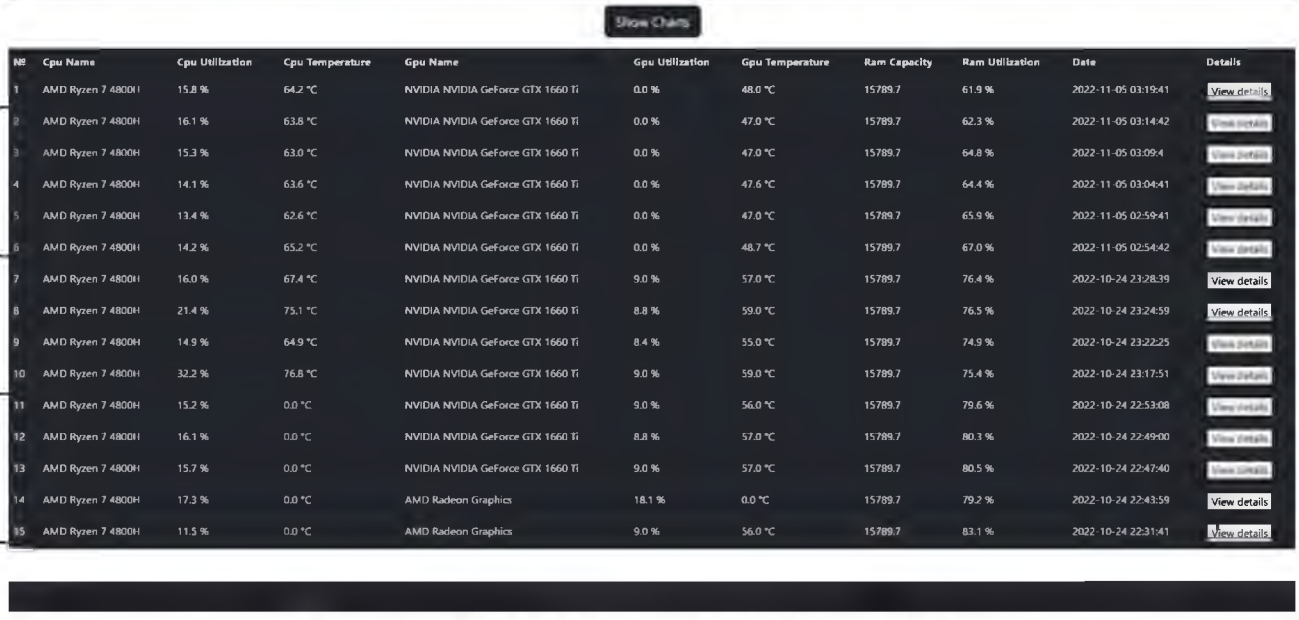
Модулі, компоненти та сервіси – це класи, які використовують декоратори. Ці декоратори позначають свій тип і надають метадані, які вказують Angular, як їх використовувати:

- метадані для класу компонента пов’язують його з шаблоном, який визначає вигляд. Шаблон поєднує в собі звичайний HTML з директивами Angular і прив’язуючою розміткою, які дозволяють Angular змінювати HTML перед відображенням на екрані;
- метадані для службового класу надають інформацію, необхідну Angular, щоб зробити її доступною для компонентів за допомогою ін’єкції залежностей (DI).

Зайшовши до сайту спочатку потрібно авторизуватись, так як без авторизації ми не можемо отримати доступ до інших функцій сайту(рисунок 3.17).

Рисунок 3.17 – Авторизація на сайті

Після авторизації ми відразу попадаємо до нашої інформації про наш ПК(рисунку 3.18).



Show Charts

№	Cpu Name	Cpu Utilization	Cpu Temperature	Gpu Name	Gpu Utilization	Gpu Temperature	Ram Capacity	Ram Utilization	Date	Details
1	AMD Ryzen 7 4800H	15.8 %	64.2 °C	NVIDIA NVIDIA GeForce GTX 1660 Ti	0.0 %	48.0 °C	15789.7	61.9 %	2022-11-05 03:19:41	View details
2	AMD Ryzen 7 4800H	16.1 %	63.8 °C	NVIDIA NVIDIA GeForce GTX 1660 Ti	0.0 %	47.0 °C	15789.7	62.3 %	2022-11-05 03:14:42	View details
3	AMD Ryzen 7 4800H	15.3 %	63.0 °C	NVIDIA NVIDIA GeForce GTX 1660 Ti	0.0 %	47.0 °C	15789.7	64.8 %	2022-11-05 03:09:4	View details
4	AMD Ryzen 7 4800H	14.1 %	63.6 °C	NVIDIA NVIDIA GeForce GTX 1660 Ti	0.0 %	47.6 °C	15789.7	64.4 %	2022-11-05 03:04:41	View details
5	AMD Ryzen 7 4800H	13.4 %	62.6 °C	NVIDIA NVIDIA GeForce GTX 1660 Ti	0.0 %	47.0 °C	15789.7	65.9 %	2022-11-05 02:59:41	View details
6	AMD Ryzen 7 4800H	14.2 %	65.2 °C	NVIDIA NVIDIA GeForce GTX 1660 Ti	0.0 %	48.7 °C	15789.7	67.0 %	2022-11-05 02:54:42	View details
7	AMD Ryzen 7 4800H	16.0 %	67.4 °C	NVIDIA NVIDIA GeForce GTX 1660 Ti	9.0 %	57.0 °C	15789.7	76.4 %	2022-10-24 23:28:39	View details
8	AMD Ryzen 7 4800H	21.4 %	75.1 °C	NVIDIA NVIDIA GeForce GTX 1660 Ti	8.8 %	59.0 °C	15789.7	76.5 %	2022-10-24 23:24:59	View details
9	AMD Ryzen 7 4800H	14.9 %	64.9 °C	NVIDIA NVIDIA GeForce GTX 1660 Ti	8.4 %	55.0 °C	15789.7	74.9 %	2022-10-24 23:22:25	View details
10	AMD Ryzen 7 4800H	32.2 %	76.8 °C	NVIDIA NVIDIA GeForce GTX 1660 Ti	9.0 %	59.0 °C	15789.7	75.4 %	2022-10-24 23:17:51	View details
11	AMD Ryzen 7 4800H	15.2 %	0.0 °C	NVIDIA NVIDIA GeForce GTX 1660 Ti	9.0 %	56.0 °C	15789.7	79.6 %	2022-10-24 22:53:08	View details
12	AMD Ryzen 7 4800H	16.1 %	0.0 °C	NVIDIA NVIDIA GeForce GTX 1660 Ti	8.8 %	57.0 °C	15789.7	80.3 %	2022-10-24 22:49:00	View details
13	AMD Ryzen 7 4800H	15.7 %	0.0 °C	NVIDIA NVIDIA GeForce GTX 1660 Ti	9.0 %	57.0 °C	15789.7	80.5 %	2022-10-24 22:47:40	View details
14	AMD Ryzen 7 4800H	17.3 %	0.0 °C	AMD Radeon Graphics	18.1 %	0.0 °C	15789.7	79.2 %	2022-10-24 22:43:59	View details
15	AMD Ryzen 7 4800H	11.5 %	0.0 °C	AMD Radeon Graphics	9.0 %	56.0 °C	15789.7	83.1 %	2022-10-24 22:31:41	View details

Рисунок 3.18 – Сторінка з інформацією.

Також ми можемо подивитись детальніше які саме процеси загружають наш ПК(рисунок 3.19).

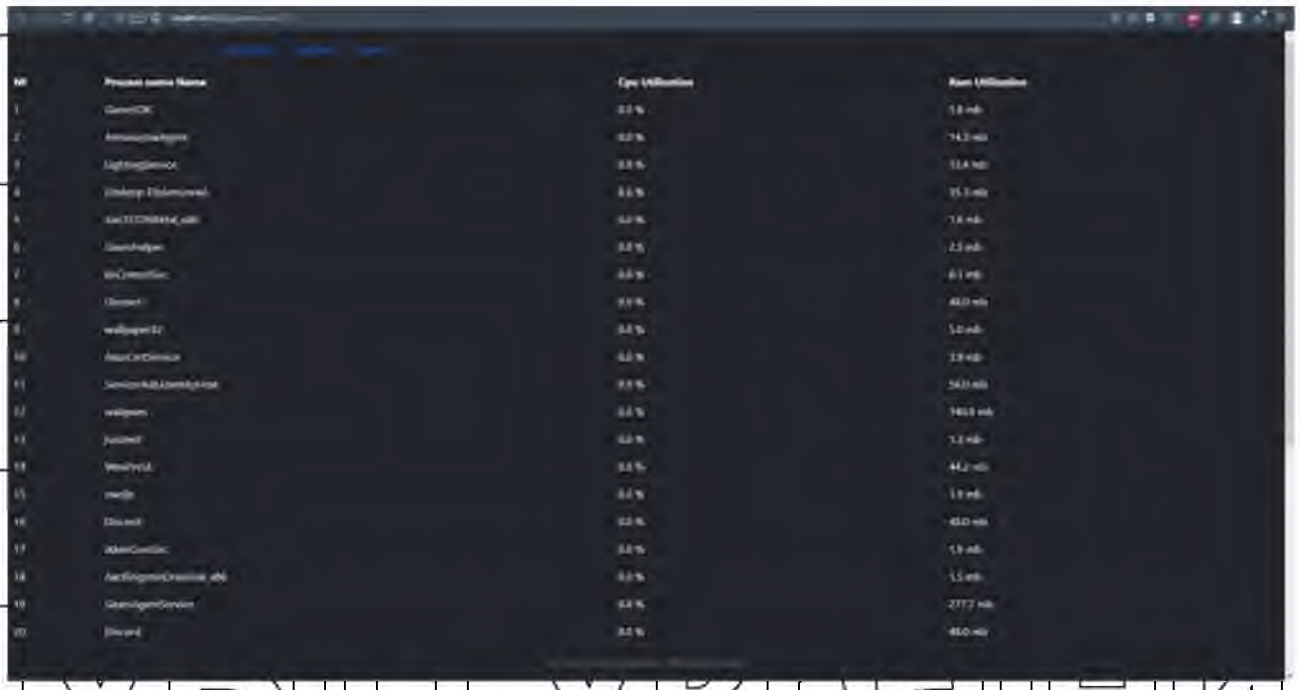


Рисунок 3.19 – Сторінка з інформацією про процеси.

Щоб детальніше відслідкувати динаміку температури та загрузки нашої системи є можливість переглянути графіки для цього потрібно натиснути кнопку «Show Charts»(рисунок 3.20).

Рисунок 3.20 – Графіки стану комплектуючого.

4 ДОСЛІДЖЕННЯ КОМП'ЮТЕРНОЇ СИСТЕМИ

4.1 Дослідження швидкодії системи

Моніторинг серверів важливий для забезпечення оптимальної продуктивності серверів, щоб гарантувати відсутність перебоїв у роботі вашого бізнесу. Однак моніторинг продуктивності сервера може бути розпорошеним і складним. Стежити за всім стало нелегкою справою. Інформація на сервері дозволяє краще зрозуміти, що пішло не так.

Що таке моніторинг серверів?

Моніторинг сервера передбачає відстеження різних показників для забезпечення його безперебійної роботи. Моніторинг різних показників допомагає легко визначити вузькі місця.

За кожну критично важливу для бізнесу онлайн-службу, як правило, відповідає кілька серверів - фізичних або віртуальних. Фізичний сервер може працювати на декількох двигунах, що призводить до виконання декількох серверних функцій. Деякі приклади фізичних серверів - це сервери баз даних, сервери додатків і веб-сервери

Чому важливий моніторинг серверів.

Моніторинг сервера необхідний для активного виявлення будь-яких проблем з продуктивністю до того, як вони вплинуть на кінцевого користувача. Крім того, моніторинг сервера допомагає зрозуміти використання системних ресурсів сервера. Це дозволяє краще планувати потужність сервера.

Моніторинг сервера забезпечує хороший показник швидкості реагування і доступності сервера - все для того, щоб забезпечити відсутність перебоїв в наданні послуг вашим клієнтам.

Показники моніторингу також можуть вказувати на загрозу кібербезпеки. Це важливо для веб-хостингу, де вплив Інтернету може призвести до підвищення рівня загрози профілю веб-сервера.

Досліджувати будем саме серверну частину системи так, як вона виконує ключову роль в нашій системі. Через сервер проходить найбільша частина інформації і клієнтська частина постійно взаємодіє з нею. Для перевірки швидкодії було обрано контролер на який приходять найбільше запитів. Щоб перевірити скільки часу затрачується на обробку та відправку пакетів. Було розроблено програму щоб відслідкувати це все. Час за скільки серверна частина справилась з 1 пакетом можемо побачити на рисунку 4.1.

Рисунок 4.1 – Час роботи сервера з 1 запитом.

Як бачимо сервер затратив на обробку даних 9мс а на те щоб отримати запит користувачу пішло 14мс. Маємо досить не поганий результат.

Тепер перевіримо скільки часу затратиться на 1,101,201…1001 запит. Таблиця 4.1.

Таблиця 4.1 Час роботи сервера

Кількість запитів	Час обробки сервером, мс	Час отримування відповіді від сервера, мс
1	9	14
101	1028	1462
201	1730	2417
301	2424	3378

401	3624	5091
501	4232	5979
601	5232	7194
701	5684	7907

Продовження таблиці 4.1

Кількість запитів	Час обробки сервером, мс	Час отримування відповіді від сервера, мс
801	6542	9050
901	7179	9909
1001	7859	10873

Побудуємо графік на основі отриманих даних(Рисунок 4.2 – 4.3).

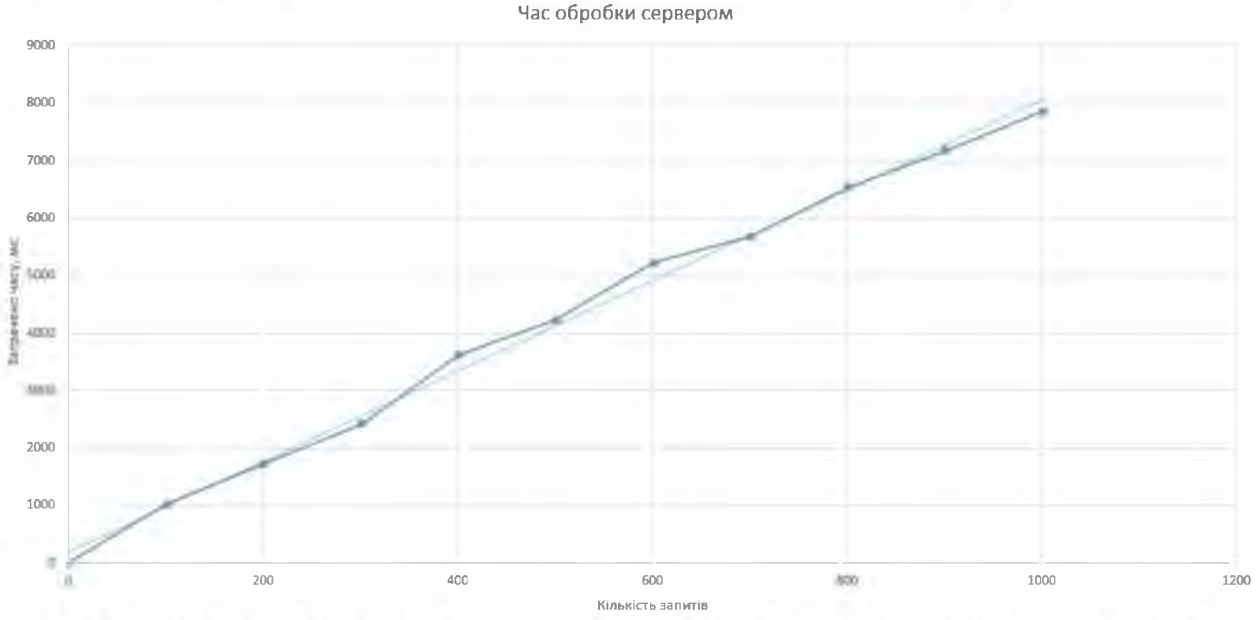


Рисунок 4.2 – Графік залежності кількості відправлених пакетів від часу обробки сервером

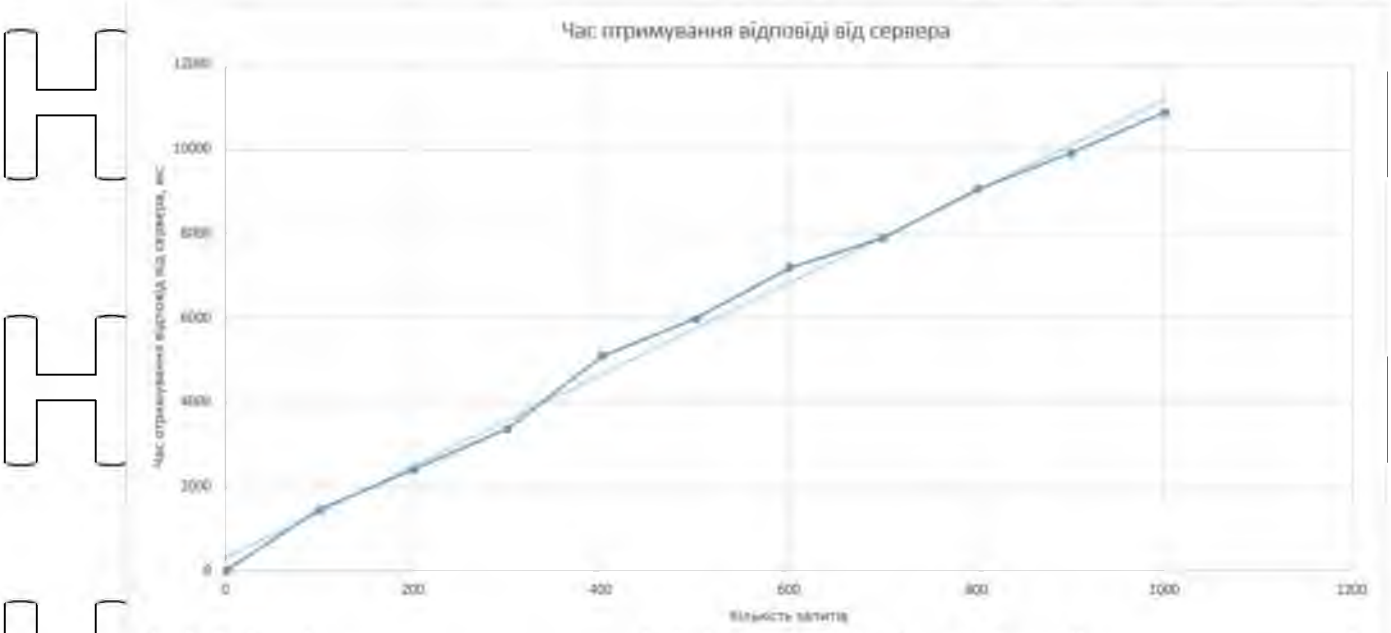


Рисунок 4.3 – Графік залежності кількості відправлених пакетів від часу отримування пакетів користувачем

Дивлячись з графіків сервер добре справляється з обробкою даних і з легкістю підтримує більше 1000 одночасних запитів.

4.2 Дослідження відмовостійкості системи

Стрес-тестування - це вид тестування програмного забезпечення, який перевіряє стабільність та надійність програмного додатку. Метою стрес-тестування є вимірювання надійності програмного забезпечення та можливостей обробки помилок в умовах екстремально високих навантажень, а також забезпечення того, щоб програмне забезпечення не вийшло з ладу в кризових ситуаціях. Він навіть тестує за межами нормальних робочих точок і оцінює, як програмне забезпечення працює в екстремальних умовах.

В інженерії програмного забезпечення стрес-тестування також відоме як тестування на витривалість. Під час стрес-тестування ПЗ піддається

навантаженню протягом короткого періоду часу, щоб дізнатися його витривалість. Найбільш відомим використанням стрес-тестування є визначення межі, на якій система або програмне чи апаратне забезпечення виходить з ладу. Воно також перевіряє, чи демонструє система ефективне управління помилками в екстремальних умовах.

Додаток, що тестується, зазнає навантаження, коли дані розміром 5 ГБ копіюються з веб-сайту і вставляються в блокнот. Блокнот зазнає навантаження і видає повідомлення про помилку "Not Responded".

Необхідність стрес-тестування

Розглянемо наступні приклади в реальному часі, де ми можемо виявити використання стрес-тестування:

- під час фестивалю на сайті інтернет-магазину може спостерігатися сплеск відвідуваності, або коли він оголошує про розпродаж;
- коли блог згадується в провідній газеті, він відчуває раптовий сплеск трафіку.

Необхідно проводити стрес-тестування, щоб впоратися з такими аномальними сплесками трафіку. Неспроможність впоратися з цим раптовим трафіком може призвести до втрати доходу та репутації.

Стрес-тестування також надзвичайно цінне з наступних причин:

- перевірити, чи працює система в аномальних умовах;
- відображення відповідних повідомлень про помилки, коли система перебуває під навантаженням;
- відмова системи в екстремальних умовах може призвести до величезних втрат доходу;
- краще бути готовим до екстремальних умов шляхом проведення стрестестування.

Цілі стрес-тестування

Метою стрес-тестування є аналіз поведінки системи після збою. Для того, щоб стрес-тестування було успішним, система повинна видавати відповідне повідомлення про помилку, перебуваючи в екстремальних умовах.

Для проведення стрес-тестування іноді використовуються великі масиви даних, які можуть бути втрачені під час стрес-тестування. Тестувальники не повинні втрачати ці дані, пов'язані з безпекою, під час проведення стрес-тестування. Основна мета стрес-тестування полягає в тому, щоб переконатися, що система відновлюється після збою, що називається відновлюваністю.

Для навантаження системи будемо використовувати програму LOIC (Low Orbit Ion Canon).

Low Orbit Ion Cannon (LOIC) - це широкодоступний додаток з відкритим вихідним кодом, розроблений компанією Praetox Technologies, який використовується для стрес-тестування мережі, а також атак на відмову в обслуговуванні (DoS) і розподілену відмову в обслуговуванні (DDoS). Також було випущено JS LOIC - JavaScript-версію додатку та веб-версію Low Orbit Web Cannon[19].

DDoS-зловмисники використовують LOIC для того, щоб засипати цільові системи небажаними TCP, UDP та HTTP GET-запитами. Однак, один користувач LOIC не в змозі згенерувати достатню кількість запитів, щоб суттєво вплинути на ціль. Для того, щоб атака була успішною, тисячі користувачів повинні координувати і одночасно направляти трафік в одну і ту ж мережу.

LOIC використовувався в ряді гучних DDoS-атак, в тому числі:

Project Chanology – Розпочата в 2008 році, ця кампанія була спрямована проти Церкви Саєнтології, яка заявила про порушення авторських прав на YouTube з метою видалення одного з її відеороликів.

Операція "Відплата" - ця широкомасштабна кампанія 2010 року була спрямована проти антипіратських організацій, Visa, MasterCard, PayPal, Sony та мереж PlayStation.

Версія LOIC, що використовувалася у вищезгаданих атаках, містила так званий режим HIVEMIND. Він використовував сервери ретрансляції інтернет-чатів для перехоплення небажаного трафіку, що генерується користувачами, тим самим дозволяючи окремим зловмисникам створювати бот-мережу та здійснювати атаки без попередньої координації.

Нашу атаку на систему можемо побачити на рисунку 4.4.

Рисунок 4.4 – Інтерфейс програми LOIC

Щоб перевірити як впливає наша DDos атака на сервер. Ми перевіримо як це впливає на обробку сервером пакетів та на їх відправку(таблиця 4.2). Таблиця 4.1 Час роботи сервера

Кількість запитів	Час обробки сервером, мс	Час отримування відповіді від сервера, мс
1	31	49

101	1139	1566
201	1820	2572
301	2941	4056
401	3471	4777
501	4253	5844

Продовження таблиці 4.1 Час роботи сервера

Кількість запитів	Час обробки сервером, мс	Час отримування відповіді від сервера, мс
601	4992	6875
701	6270	8547
801	6663	9169
901	7416	10187
1001	8482	11597

Як бачимо з графіків(рисунків 4.5-6) DDos атака злегка затримала час обробки та відправки пакетів, але сервер витримав нашу перевірку на відмовостійкість.

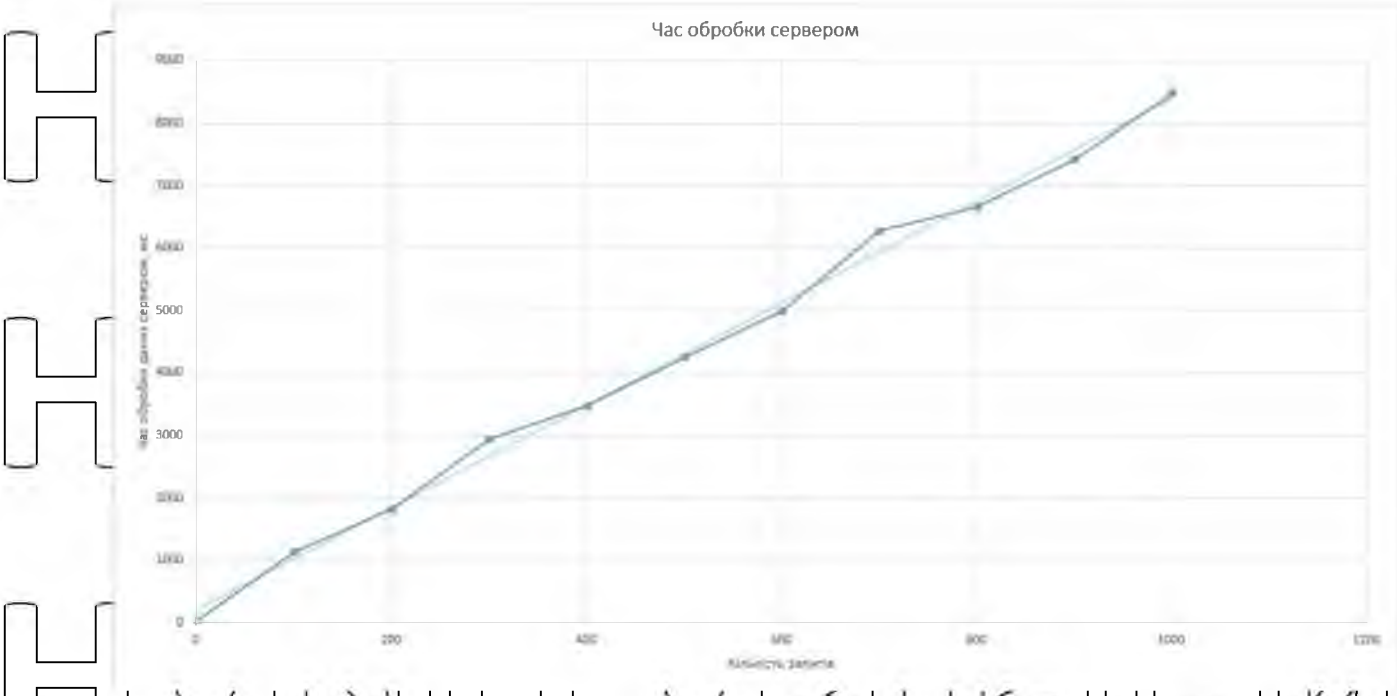


Рисунок 4.5 – Графік залежності кількості відправлених пакетів від часу обробки сервером

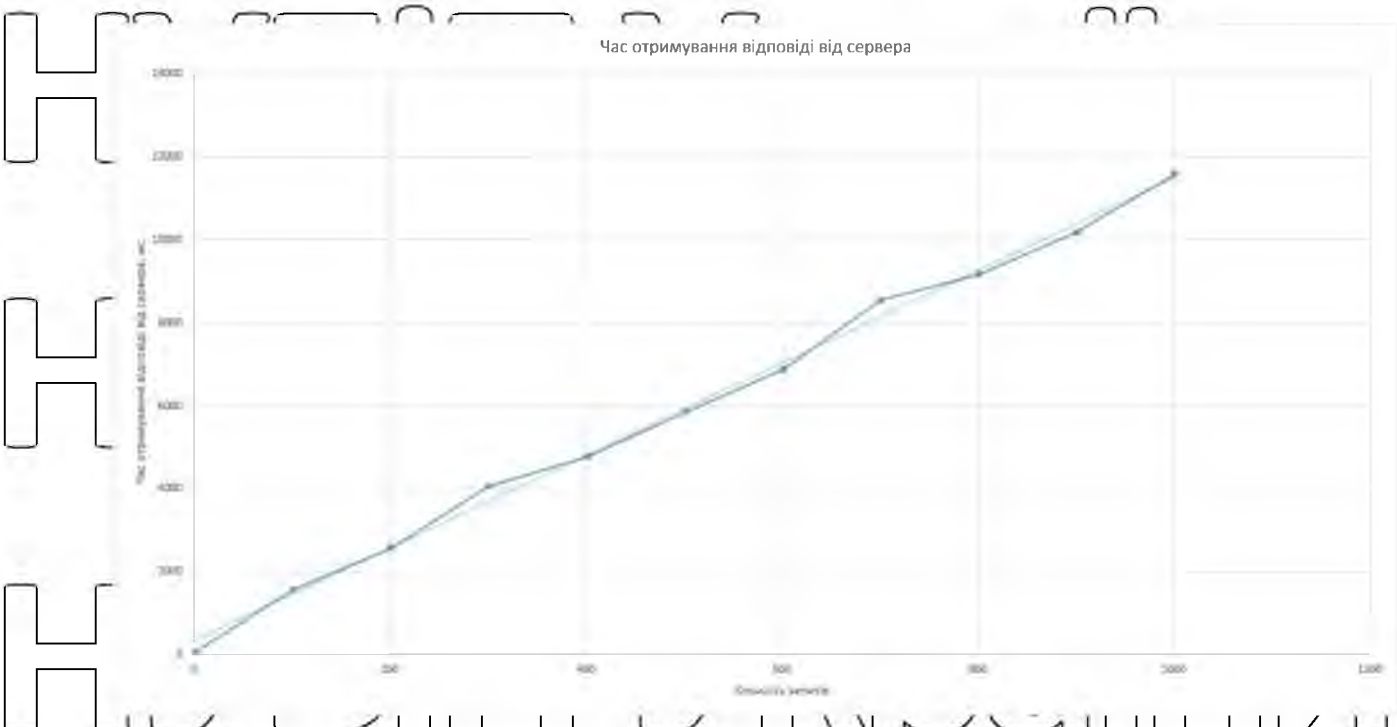


Рисунок 4.6 – Графік залежності кількості відправлених пакетів від часу отримування пакетів користувачем

За час перевірки на сервер DDos атакою було відправлено більше 350000 пакетів(рисунок 4.7).

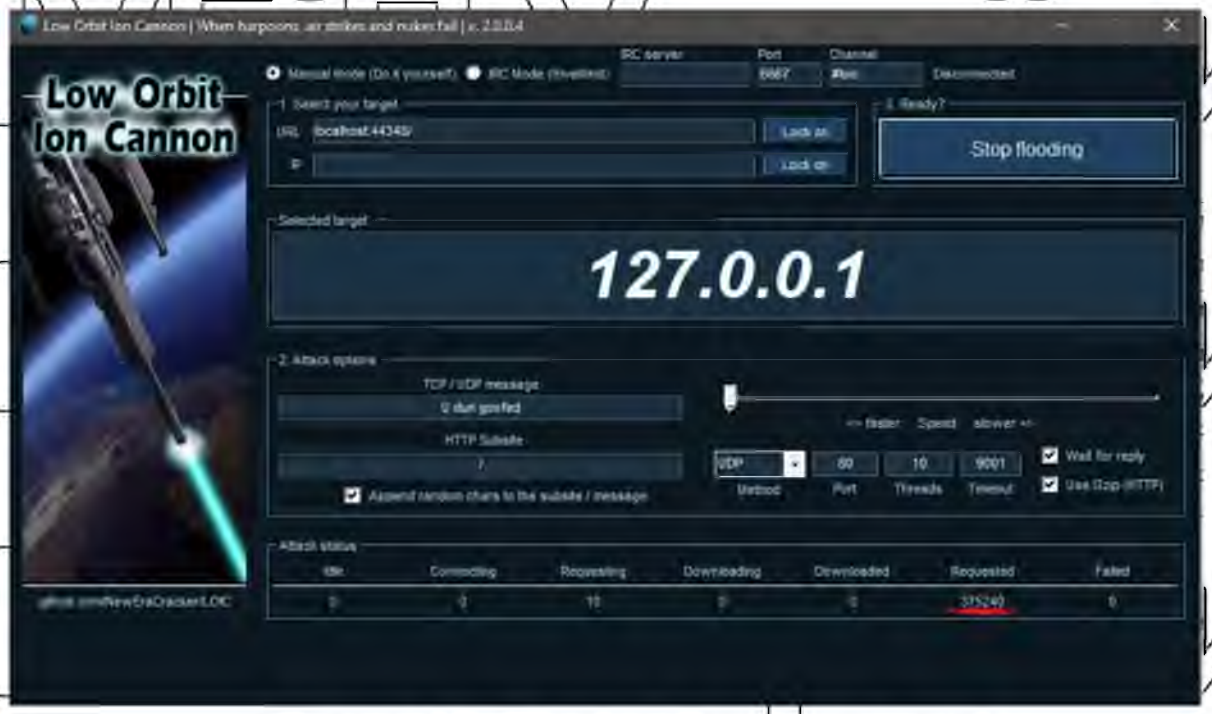


Рисунок 4.7 – Кількість відправлених пакетів

ВИСНОВКИ

Таким чином, у результаті виконаної дипломної роботи розроблено комп'ютерну систему прогнозування стану обчислювальної програмноапаратної платформи, призначену для використання на підприємствах, які потребують більш докладного слідкування за станом комп'ютерів.

У роботі вирішено наступні задачі:

1. Проведено порівняння аналогічних існуючих систем, розглянуто основні вимоги до системи, проведено аналіз предметної області.
2. Проведено проектування системи, у результаті якого запропоновано ряд вимог до системи щодо атрибутів, властивостей та якостей системи, розроблено специфікацію вимог до програмного забезпечення (SRS) згідно діючої методики складання специфікацій вимог до програмного забезпечення, що рекомендується Інститутом Інженерів з Електротехніки та Електроніки.

3. Обрано інтерфейс користувача віконного типу та розроблено прототипи дизайну згідно особливостей типу інтерфейсу, розроблено серверну частину, клієнтську частину, а саме десктопної програми та сайту.

4. В четвертому розділі досліджено систему на швидкодію та відмовостійкості.

В ході дослідження системи було виявлено що система витримала стрес атаку і показала досить великий рівень швидкодії. Під DDos атакою середня затримка збільшилася на 1мс.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. The New Hardware Development Trend and the Challenges in Data Management and Analysis URL:https://link.springer.com/article/10.1007/s41019-018-0072-6 (дата звернення: 20.10.2022)
2. Software Testing Help URL:https://www.softwaretestinghelp.com/hardware-monitoring-tools/ (дата звернення: 20.10.2022)
3. What is a System Requirements Specification (SRS)? URL:https://www.inflectra.com/Ideas/Topic/Requirements-Definition.aspx (дата звернення: 20.10.2022)
4. What Is SRS (Software Requirements Specification) Document? URL:https://forbytes.com/blog/what-is-srs/ (дата звернення: 20.10.2022)
5. OOAD - Object Oriented Analysis URL:https://www.tutorialspoint.com/object_oriented_analysis_design/ooad_object_o riented_analysis.htm (дата звернення: 20.10.2022)
6. Object-Oriented Modeling (OOM) URL:https://www.techopedia.com/definition/28584/object-oriented-modeling-oom (дата звернення: 20.10.2022)
7. Dynamic modelling in object oriented analysis and design URL:https://www.geeksforgeeks.org/dynamic-modelling-in-object-oriented-analysisand-design\ (дата звернення: 21.10.2022)
8. OOAD - Functional Modeling URL:https://www.tutorialspoint.com/object_oriented_analysis_design/ooad_functional_modeling.htm (дата звернення: 20.10.2022)
9. Стандарт IEEE 830-1998. Методика составления спецификаций требований к программному обеспечению.

10. A Beginner's Guide to Back-End Development URL:https://www.upwork.com/resources/beginners-guide-back-end-development (дата звернення: 21.10.2022)
11. Язик програмування C# URL:https://timeweb.com/ru/community/articles/chto-takoe-csharp (дата звернення: 20.10.2022)
12. Three-Tier Architecture URL:https://www.ibm.com/cloud/learn/three-tierarchitecture (дата звернення: 24.10.2022)
13. Repository Design Pattern In ASP.NET MVC URL:https://www.c-sharpcorner.com/article/repository-design-pattern-in-asp-net-mvc/ (дата звернення: 24.10.2022)
14. Unit Of Work in Repository Pattern URL:https://dotnettutorials.net/lesson/unit-of-work-csharp-mvc (дата звернення: 26.10.2022)
15. Business Logic Layer And Why It Matters To Understand Blockchain Business Models URL:https://fourweekmba.com/business-logic-layer/ (дата звернення: 26.10.2022)
16. Data Mapper URL:https://martinfowler.com/eaaCatalog/dataMapper.html (дата звернення: 29.10.2022)
17. Swagger URL:https://highload.today/swagger-api/ (дата звернення: 30.10.2022)
18. What is Frontend? URL:https://www.frontend-gmbh.de/en/blog/what-isfrontend-what-is-backend/ (дата звернення: 30.10.2022)
19. Low Orbit Ion Cannon (LOIC) URL:https://www.imperva.com/learn/ddos/low-orbit-ion-cannon/ (дата звернення: 1.11.2022)