МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп’ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління

ДЯКІВ Ростислав Ігорович

Метод моніторингу системних ресурсів операційних систем / Method of monitoring system resources of operating systems

спеціальність: 122 - Комп'ютерні науки
освітньо-професійна програма - Комп’ютерні науки

Кваліфікаційна робота

Виконав студент групи
КНм-21
Р.І. Дяків
_____

Науковий керівник:
к.т.н., доцент І.В. Турченко
_____

Кваліфікаційну роботу
допущено до захисту:
«___» _____ 20___ р.
Завідувач кафедри
_____ М.П. Комар

ТЕРНОПІЛЬ - 2021

ЗМІСТ

ВСТУП

Актуальність: сьогодні тема моніторингу комп'ютерів є дуже актуальною, бо працівники, що відповідають за належний стан IT-інфраструктури компанії хочуть моніторити стан, продуктивність та завантаженість комп'ютерів, що входять до їх мережі. Тому такі системи швидко набирають популярність на ринку сучасних IT-технологій. Всі прагнуть розробити системи, які б відповідали сучасним потребам IT-адміністраторів, але втілити це не завжди вдається через складність імплементації і ряд інших причин пов'язаних з комплексними комп'ютерними системами сьогодення.

Комп'ютерні системи компаній повинні бути доступні в будь-якій ситуації, часто в будь-який час доби та працювати безвідмовно, особливо це стало актуальним при дистанційній роботі [20].

Мета та завдання дослідження: метою є вдосконалення методу моніторингу системних ресурсів операційних систем, який знизить витрати коштів та часу на обслуговування, підтримку і використання комп'ютерного забезпечення, серверів підприємства, ремонт та попередження, або ж відновлення комп'ютерних систем підприємства.

Об'єкт дослідження: системні ресурси операційних систем.

Предмет дослідження: метод моніторингу системних ресурсів операційних систем.

Методи дослідження: системний аналіз, методи збору даних, математична статистика, методи розробки мікросервісної системи.

Наукова новизна одержаних результатів: запропоновано метод моніторингу системних ресурсів операційних систем, який на відміну від інших використовує протокол передачі даних SignalR, завдяки методу агрегації зібраної за певний проміжок часу інформації дозволяє краще візуалізувати дані.

Практичне значення одержаних результатів: результати роботи можуть бути використані підприємствами для моніторингу системних ресурсів операційних систем комп'ютерів що є частиною їх IT-інфраструктури.

Публікації та апробація КР.

Результати дослідження опубліковано та апробовано в матеріалах міжнародної науково-практичної інтернет-конференції (6-7 грудня 2021 р., м. Дніпро) та у матеріалах міжнародної наукової інтернет-конференції “Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення” (10 грудня 2021 р., м. Тернопіль) (Додаток В).

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВІДОМИХ РІШЕНЬ У СФЕРІ МОНІТОРИНГУ СИСТЕМНИХ РЕСУРСІВ ОПЕРАЦІЙНИХ СИСТЕМ

1.1 Моніторинг та аналіз комп'ютерних систем

Комп'ютерна система — це базове, повне та функціональне обладнання та програмне забезпечення з усім необхідним для реалізації обчислювальної продуктивності [20].

Моніторинг та аналіз комп'ютерних систем поширений практично в усіх галузях інформаційних технологій. Цей напрям потребує поглибленого теоретичного і практичного аналізу досліджуваних процесів та створення спеціальних математичних засобів для прогнозування показників роботи комп'ютера [20].

Комп'ютери, які використовуються в різних організаціях чи фірмах, вирішують різнопланові завдання і важливо їх підтримувати в хорошому стані задля того, щоб мінімізувати збої у їх роботі, уникнути помилок при обробці даних, запровадити методи прогнозування виходу з робочого стану та запобігти повній або частковій заміні компонентів [20].

Найкращим підходом для вирішення цієї проблеми є моніторинг системних ресурсів операційної системи комп'ютерів у режимі реального часу. Це забезпечить стабільний контроль над комп'ютерами у весь робочий день та допоможе зекономити час для адміністратора і кошти для підприємства. В такому випадку добре мати застосунок, який матиме інтерфейс взаємодії для адміністратора та сервіс для комп'ютерів, стан яких потрібно відслідковувати. В якості такого застосунка зміг би виступити комплекс програмного забезпечення, який займатиметься збором і надсиланням різних показників системи в даний момент часу на сервер для подальшого зберігання, аналізу цих даних і відображення їх у зручному для користувача вигляді чи надсилання текстових нотифікацій у разі потреби.

Дане програмне забезпечення буде корисне для підприємств різного розміру, але особливо тих у яких є багато комп'ютерів стан яких фізично важко

відслідкувати системному адміністратору. Хоча основною сферою застосування такого програмного забезпечення є фізичні сервери чи віртуальні машини, які запущені для виконання якоїсь роботи і за якими не сидять користувачі. Зазвичай такого роду комп’ютери повинні працювати у безвідмовному режимі досить великі проміжки часу, це можуть бути навіть роки. Кожна поломка в операційній системі цих комп’ютерів може дорого коштувати для підприємства. В цьому випадку дане програмне забезпечення допоможе вчасно виявити незвичну поведінку системи, або в разі поломки провести невідкладні роботи для негайного її вирішення.

Основні потужності більшості сучасних IT компаній базується на комп'ютерних системах, тому їх моніторинг став основним завданням управління всією IT-інфраструктурою компанії, виконання якого дозволяє досягти наступні цілі:

– отримати максимум переваг використання ресурсів компанії;

– запобігти та передчасно виявити проблеми;

– сповіщати про виявлені проблеми відповідальних осіб.

Загалом ці цілі можна об’єднати в одну мету, досягнення якої призводить до скорочення витрат на обслуговування комп’ютерів, зменшення кількості інцидентів недоступності IT-інфраструктури компанії, скорочення часу на відновлення комп’ютерних потужностей та, як результат, більшого рівня задоволення потреб клієнтів [20].

Моніторинг повинен включати максимальну кількість показників системи задля створення повної картини роботи комп’ютера. Найважливіші з них: використання CPU і RAM, використання диску і мережі [20].

Засоби моніторингу системи повинні бути зосереджені на процесах, пам’яті, сховищі та мережевих з’єднаннях системи, для якої проводиться моніторинг. Базуючись на меті моніторингу комп’ютерних систем, можна виділити наступні переваги, які надає даний підхід:

– можливість налаштовувати події та сигнали тривоги, пов’язані з ними. Деякі приклади тривожних сигналів можна застосувати, щоб попередити про заповнені жорсткі диски, завантаженість оперативної пам’яті понад 80%, надмірний доступ до дисків з дозволом на запис, занадто багато відкритих потоків

в одній системі, тощо;

– можливість отримати доступ до інформації про стан налаштувань та перевірити найважливіші технічні ресурси;

– наявність доступу до статусу комп’ютерної системи в режимі реального часу;

– підвищення ефективності та продуктивності виконання завдань з обслуговування, що виконуються системою;

– створення системних інвентарів (карти, списки);

– планування зростання чи спаду використання комп’ютерних систем на основі їх навантаженості;

– можливість отримання інформації про те, коли може знадобитися більше місця для зберігання даних, новий сервер чи оновлення пам’яті;

– зниження витрат на підтримку IT-інфраструктури компанії.

Системний Монітор - це апаратний або програмний компонент, який використовується для контролю системних ресурсів та продуктивності в комп'ютерній системі. Серед проблем управління щодо використання засобів моніторингу системи виділяються дві основні: використання ресурсів та конфіденційність. Засоби моніторингу програмного забезпечення функціонують всередині пристрою, який вони моніторять [21].

Копія екрану компонента Системного Монітору (System Monitor), що показує використання ресурсів системи та їх стан представлена на рисунку 1.1.

Рисунок 1.1 – Системний монітор Windows

Монітори програмного забезпечення зустрічаються частіше, ніж апаратні та іноді як частина движка віджетів. Вони часто використовуються для відстеження

системних ресурсів, таких як використання та частота процесора або кількість вільної оперативної пам’яті. Ці системи моніторингу також використовуються для відображення таких елементів як вільний простір на одному або декількох жорстких дисках, температура процесора та інших важливих компонентів. Крім цього збирається мережева інформація, включаючи IP-адресу системи та поточну швидкість завантаження та надсилання. Інші дисплеї системного монітору можуть включати дату та час, системний час, ім'я комп'ютера, ім'я користувача, S.M.A.R.T. дані жорсткого диску, швидкість обертання охолоджуючого вентилятора та значення напруги, яка забезпечується джерелом живлення [21].

Рідше зустрічаються апаратні системи, що контролюють інформацію подібну до тої, що зчитують програмні монітори. Зазвичай вони займають один або декілька відсіків накопичувача на передній панелі корпусу комп'ютера та з'єднуються безпосередньо із системним обладнанням або підключаються до програмної системи збору даних через USB. В обох підходах до збору даних, система моніторингу відображає інформацію на одному або кількох аналогових чи світлодіодних цифрових дисплеях. Апаратні монітори також дозволяють безпосередньо керувати швидкістю вентиляторів, дозволяючи користувачу швидко налаштувати охолодження в системі [21].

Існують кілька спеціальних високоточних апаратних моделей моніторингу системи, що розроблені конкретно для взаємодії лише з певною моделлю материнської плати. Ці моделі безпосередньо використовують датчики, вбудовані в комп’ютер, надаючи більш детальну та точну інформацію, ніж їх менш точні дешевші аналоги [21].

Коли окремий користувач вимірює продуктивність системи, якою користується лише одна людина, тобто окремий комп’ютер чи віртуальна машина в багатокористувацькій системі, доступ не порушує конфіденційності інших користувачів. Конфіденційність стає проблемою, коли хтось, крім кінцевого користувача, наприклад системний адміністратор, має легітимну потребу отримати доступ до даних про інших користувачів [21].

Коли події відбуваються швидше, ніж монітор може записати їх, вирішенням є заміна запису подій простим підрахунком. При цьому потрібно зважати на те, щоб

підрахунок не мав значного впливу на процесор, сховище даних та споживання оперативної пам'яті і не перешкоджав продуктивній роботі системи.

За допомогою компоненту Ресурсний монітор (Resource Monitor) можна вимірювати продуктивність одного комп'ютера або декількох комп'ютерів у мережі [21].

Копія екрану компонента Ресурсного монітору (Resource Monitor) в операційній системі Windows 10 представлена на рисунку 1.2.

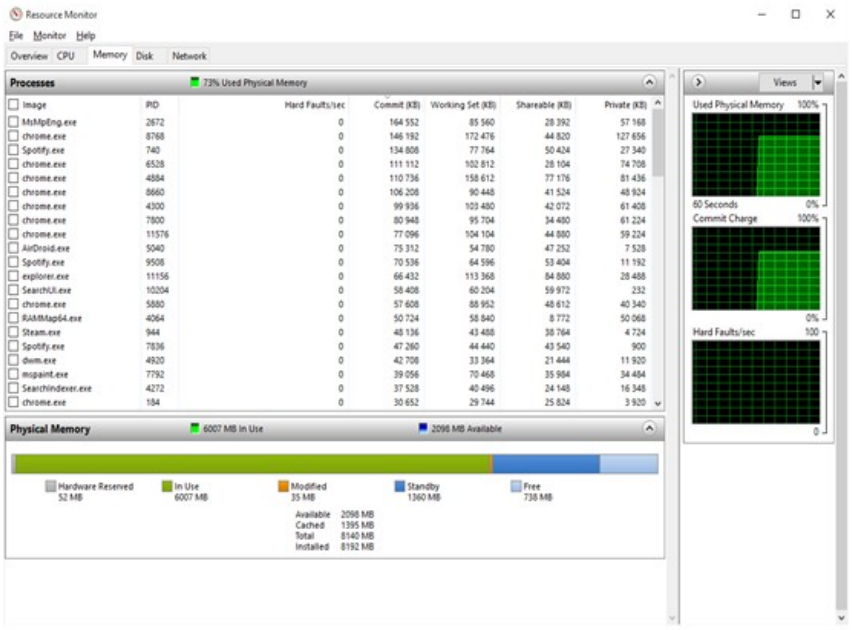

Рисунок 1.2 – Копія екрану компонента Системного монітору Windows

Хоча апаратний монітор, як правило, матиме менший вплив на продуктивність системи, ніж програмний монітор, є такі елементи даних, як деяка описова інформація, наприклад, назва програм, що повинна включати програмне забезпечення. Тут також є ризик того, що помилка в цій предметній області може мати серйозний вплив. У найгіршому випадку це може спричинити збій операційної системи на пристрої, який знаходиться під моніторингом [21].

Моніторинг мережі - це використання системи, яка постійно контролює комп'ютерну мережу для повільних або несправних компонентів і повідомляє адміністратора мережі електронною поштою, SMS або іншими засобами у разі відключення, затримки або інших проблем. Моніторинг мережі є частиною

управління мережею.

Моніторинг веб-сервера означає, що власник сервера завжди знає коли, як та чому один чи кілька його сервісів не працюють чи не є доступними для кінцевих користувачів. Моніторинг сервера може бути внутрішнім, тобто програмне забезпечення веб-сервера перевіряє його стан і повідомляє власника про те, що деякі сервіси не працюють. Також є зовнішнє, тобто деякі компанії, що займаються моніторингом веб-серверів перевіряють стан послуг з певною частотою і дають зворотній зв’язок щодо них власнику. Моніторинг сервера може охоплювати перевірку системних показників, таких як використання процесора, використання пам'яті, продуктивність мережі та дисковий простір. Він також може включати моніторинг додатків, наприклад перевірку процесів таких програм, як Apache, MySQL, Nginx, Postgres, MondoDB, IIS, Kassanda, Kafka, Kestrel та інші.

Зовнішній моніторинг є більш надійним, оскільки він продовжує працювати, коли сервер, що моніториться повністю виходить з ладу. Хороші інструменти моніторингу сервера також мають показники продуктивності, можливості оповіщення та можливість пов’язувати певні пороги витрат з автоматизованими завданнями сервера, такими як забезпечення більшої кількості пам'яті або проведення резервного копіювання.

Служби моніторингу мережі зазвичай мають ряд серверів по всьому світу: в Америці, Європі, Азії, Австралії, Африці, та інших місцях. Маючи кілька серверів у різних географічних місцях, служба моніторингу може визначити чи веб-сервер доступний у різних мережах по всьому світу. Чим більше місць розташування використовується, тим повнішою є картинка про доступність системи.

Під час моніторингу системи на предмет можливих проблем, зовнішня служба моніторингу веб-серверів перевіряє ряд параметрів. Перш за все, вона контролює правильний код відповіді HTTP. За специфікаціями HTTP RFC 2616 будь-який веб-сервер повертає кілька кодів HTTP, таких як 200, 404, 501 та інші. Аналіз HTTP-кодів - це найшвидший спосіб визначення поточного стану веб-сервера, що моніториться. Інструменти моніторингу продуктивності сторонніх програм надають додаткові можливості моніторингу, оповіщення та звітування веб-серверів.

Оскільки інформація, що надається службами моніторингу веб-серверів, в більшості випадків є нагальною і має вирішальне значення, можуть використовуватися різні способи сповіщення: електронна пошта, мобільні телефони, месенджери, SMS, факс, пейджери тощо, аби забезпечити безперервну обізнаність власника серверу про його теперішній стан.

Агенти моніторингу та спостереження, також відомі як агенти прогнозування, - це тип програмного забезпечення інтелектуального агента, який збирає дані та звітує про стан та характеристики комп'ютерної системи. Агенти моніторингу та спостереження часто використовуються для моніторингу складних комп'ютерних мереж, щоб передбачити, коли може статися збій чи якийсь інший дефект. Інший тип агента моніторингу та спостереження працює в комп'ютерних мережах, відслідковуючи конфігурацію кожного комп'ютера, підключеного до мережі. Він відстежує та оновлює центральну базу даних конфігурації, коли щось на будь-якому комп'ютері змінюється, наприклад, кількість або тип дискових накопичувачів. Важливим завданням управління мережами є пріоритетність трафіку та формування пропускної здатності.

Прикладом агентіа моніторингу та спостереження є агент, що має лабораторія реактивного руху NASA, що здійснює контроль за інвентаризацією та плануванням обладнання для того, щоб зменшити витрати. Allstate Insurance має мережу з тисячами комп'ютерів. Компанія використовує агент з моніторингу мережі від Computer Associates International під назвою Neugent, який спостерігає за своїми величезними мережами 24 години на добу. Кожні п’ять секунд агент вимірює 1200 точок даних і може передбачити збій системи за 45 хвилин до того, як це станеться.

Агенти які використовують вищезгадані компанії дозволяють рівень питання безпеки даних в приватних хмарах, оскільки питання зберігання даних має найбільший пріоритет.

1.2 Аналіз відомих рішень моніторингу системних ресурсів

Найбільш популярними рішеннями для моніторингу завантаженості комп'ютерів на даний час є:

– New Relic;

– Datadog;

– Microsoft Azure.

New Relic забезпечує збір даних, візуалізацію та аналіз метрик, подій, журналів в одному місці, надаючи інформацію про стан багатохмарних серверів.

За допомогою їхньої розробки New Relic APM можна контролювати програми, що працюють у контейнерах. За допомогою нового розподіленого відстеження New Relic можна простежити шлях запиту під час проходження по стеку Kubernetes, виявити затримку компонентів на цьому шляху та визначити, який компонент на шляху створює проблеми з ефективністю.

Такі рішення дозволяють відслідковувати конкретне встановлене програмне забезпечення, а не всю систему загалом.

На рисунку 1.3 можна ознайомитись з користувацьким інтерфейсом головної сторінки New Relic. Проаналізувавши даний інтерфейс, що призначений для вибору та налаштування певної системи моніторингу, були встановлены деякі недоліки та переваги такого інтерфейсу.

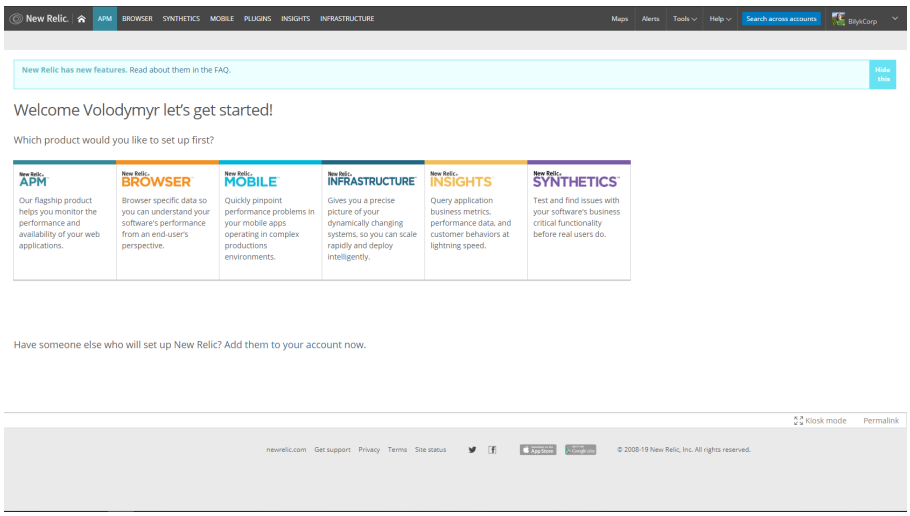

Рисунок 1.3 – Web-інтерфейс стартової сторінки порталу New Relic

Перевагами програми New Relic є простий не перевантажений анімаціями чи іншими зображеннями інтерфейс, що допомагає концентруватися на потрібній задачі.

Недоліки:

– такий інтерфейс до кінця не розкриває всі можливості кожної системи, яка користується сервісами New Relic;

– недоцільне використання вільного простору на web-ресурсі (можна побачити багато вільного місця справа, що ніяк не використовується, даних про систему в цей же час є недостатньо).

Ще одним недоліком даної системи і чому вона не підходить для моніторингу комп'ютерних систем це те, що New Relic не може відслідковувати завантаженість комп'ютера, натомість вона дає можливість моніторити швидкодію різних аплікацій, що виконуються на комп'ютері.

Як можна побачити на рисунку 1.4, графіки загруженості комп'ютера виглядають доволі непогано і показують дійсно релевантні дані для аналізу.

Проаналізувавши інтерфейс та діаграми, що наявні на екрані, було виділено кілька переваг:

– велика кількість потрібних графіків;

– можливість одночасно відслідковувати значення різноманітних параметрів комп'ютера;

– можливість моніторингу максимальних та мінімальних показників характеристик системи.

Недоліки:

– перевантаженість сторінки різними елементами, не дає можливості швидко зорієнтуватися в інтерфейсі.

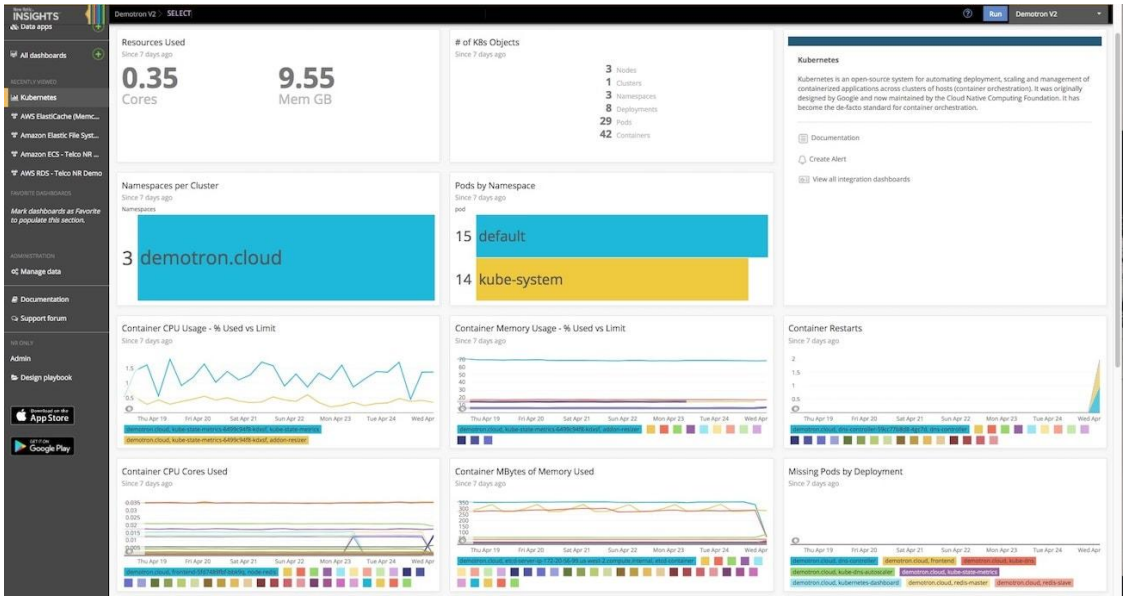

Рисунок 1.4 – Web-інтерфейс додатку New Relic для відслідковування параметрів системи

На рисунку 1.5 зображений інтерфейс налаштування та вибору ОС комп’ютера, на якому проводитиметься моніторинг.

Зробивши аналіз інтерфейсу, можна зробити певні висновки.

Переваги:

– доступний широкий вибір систем, для моніторингу:

1) Linux;
2) Ubuntu;
3) Kali;
4) Redhat;
5) Fedora;
6) Mac OS;
7) Windows;

– перелік доступних ОС займає усю ліву частину меню – це дає можливість швидко знайти підходящу систему та розпочати роботу;

– показаний приклад графіку моніторингу.

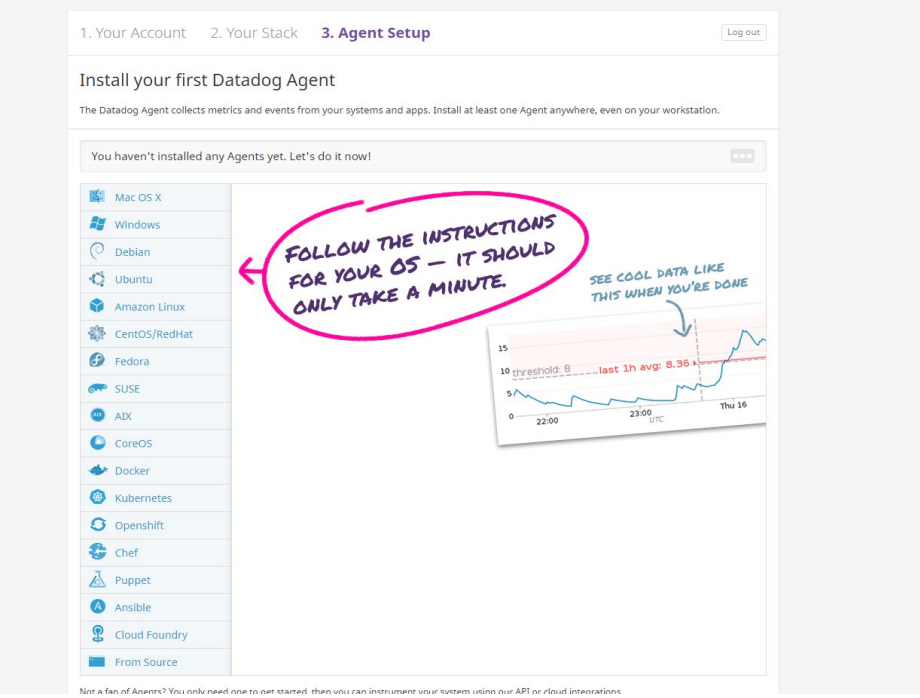

Рисунок 1.5 – Web-інтерфейс системи Datadog для початкового налаштування

Інтерфейс, на рисунку 1.6, надає користувачу інструкцію для встановлення програмного забезпечення на комп'ютер. Для операційної системи Windows є наступні можливі інсталяції аплікації:

– використовуючи інсталятор додатків під Windows;

– використовуючи командну стрічку.

Це хороший бонус до зручності в користуванні продуктом, так як це дає користувачу обрати зручний для нього інструмент. Одразу після того як користуєшся додатками такого типу, можна зрозуміти, що інтерфейс користувача відіграє надзвичайно велику роль у розробці програмного забезпечення різного роду.

На рисунку 1.7 зображений інтерфейс робочого столу з усіма графіками завантаженості системи, а саме:

– загруженість CPU;

– загруженість RAM;

– кількість вільної постійної пам'яті.

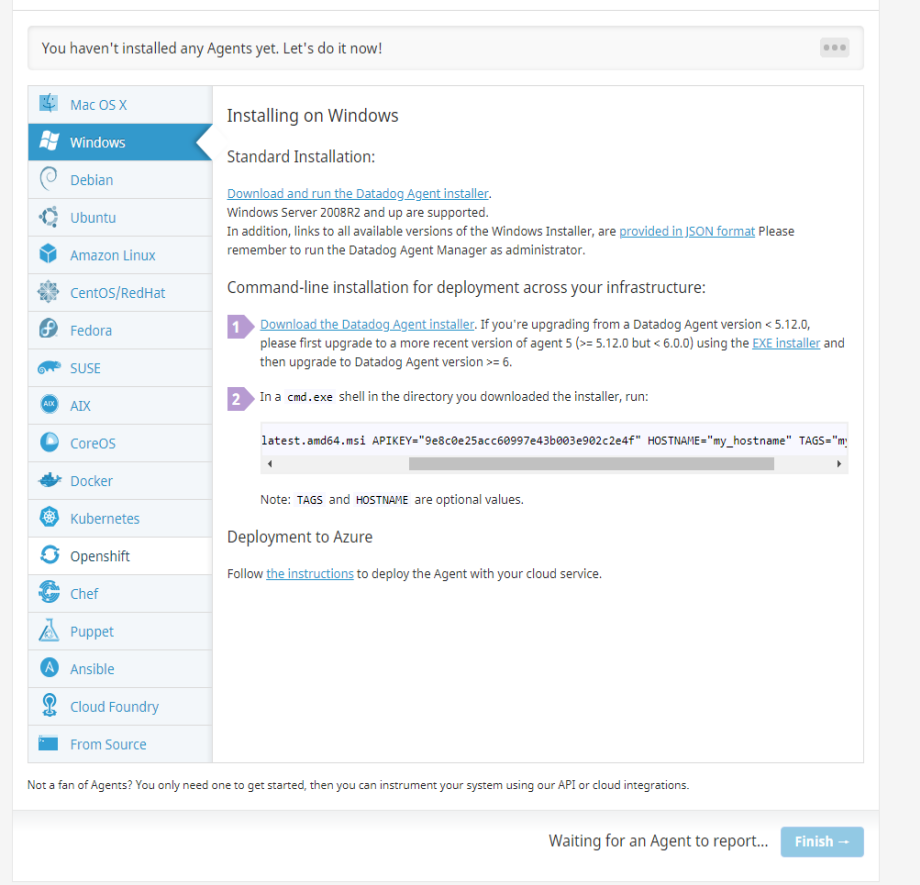

Рисунок 1.6 – Web-інтерфейс системи Datadog для вибору операційної системи та конфігурації програми моніторингу

Варто визначити так звані “Pros & Cons” наступного web-інтерфейсу.

Переваги:

– є дані про стан програм, що виконуються на комп’ютері;

– моніторинг в реальному часі;

– відображені графіки, по яких можна легко віднайти максимальні точки завантаженості того чи іншого компоненту;

– можливість відслідковувати метрики процесів.

Недоліки:

– перевантажений інтерфейс багатьма елементами, що не дає швидко зорієнтуватися на потрібному;

– не можна гнучко налаштувати інтерфейс (змінити місця положення графіків).

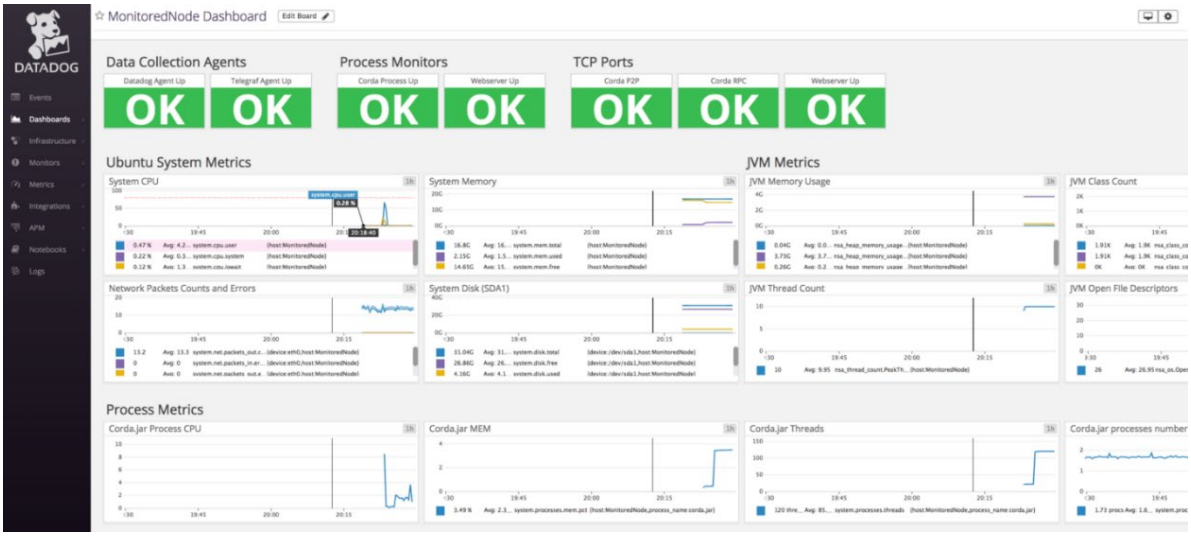

Рисунок 1.7 – Web-інтерфейс Datadog для моніторингу характеристик системи

Microsoft Azure - це хмарна платформа з великим набором доступних функцій, які дозволяють без особливих зусиль розробляти, розгортати та управляти своїми послугами без особливих зусиль. Microsoft Azure має дві моделі - Платформа як сервіс (PaaS) та Інфраструктура як сервіс (IaaS) та розповсюджується за принципом, що дозволяє нам повністю контролювати наші витрати ресурсів. Але Microsoft Azure не звертає всієї уваги лише на хмарних ЦОД (центрах обробки даних), він також підтримує гібридний інфраструктурний формат, який надає користувачеві інструменти для збільшення можливостей зберігання, архівації та відновлення найбільш ефективним та економічно ефективним способом.

Функціоність платформи безперервно розвиваються, щоб охопити більшість завдань і сервісів, якими користувач зазвичай користується. Жоден постачальник цих послуг не пропонує в своїх ЦОД стільки послуг, які в свою чергу тісно пов'язані один з одним і є повноцінним комплексом для вирішення будь-якої проблеми. Це робить платформу Azure найпопулярнішою серед всіх наявних. На сьогоднішній день її послуги доступні більше ніж у 20 регіонах планети, і Microsoft не зупиняється на досягнутому, постійно розширюючись, надаючи свої послуги у все більше і більше регіонах, щоб забезпечити найбільшу продуктивність, конфіденційність та збереження даних. З всіх наявних служб та послуг є основні, які беруть участь майже у всіх випадках вирішення проблем:

– віртуальна машина Azure - дозволяє розробляти та використовувати віртуальні машини в хмарі та забезпечує багато гнучких можливостей віртуалізації без необхідності придбання та підтримання фізичної машини. Хоча користувачеві доводиться підтримувати віртуальну машину - налаштовувати її, встановлювати патчі та підтримувати програмне забезпечення, що працює на віртуальній машині, це робить набагато легшим процес керування обчислювальною потужністю і дозволяє керувати витрати на утримання. Підхід IaaS, застосовуваний у технології віртуальної машини, надає можливість користуватися ним за допомогою багатьох способів;

– служба зберігання даних Azure - хмарне вирішення для зберігання модерних додатків, що надає високу доступність та масштабованість для виконання поточних вимог користувача. Служба зберігання даних Azure дуже масштабована, що дозволяє зберігати та обробляти декілька сотень терабайтів даних для підтримки масштабних даних, необхідних для досліджень, аналітичних, фінансових та мультимедійних додатків. Ви також можете зберігати велику кількість інформації. Якщо виникає потреба щось змінити, стягуються лише інформація, яка була збережена. В даний час MA зберігає декілька десятків трильйонів неповторних клієнтських об'єктів і обробляє кілька мільйонів запитів в секунду;

– віртуальна мережа MA (VNet) - це хмарна мережа яка повністю керується користувачем. Ви можете керувати блоками IP-адрес, налаштуваннями DNS, політикою безпеки та таблицями маршрутизації в цій мережі. Також, можна розбити віртуальну мережу на менші підмережі та запустити віртуальні машини MA IaaS та/або хмарні сервіси (екземпляри ролі PaaS). Можливе додання віртуальної мережі до локальної мережі. Це дозволить збільшити локальну мережу в MA з повним контролем над блоками IP-адрес та перевагами програм для корпоративного рівня, що надаються MA;

– відновлення сайту MA - надає можливість втілити стратегію BCDR. ASR керує процесами реплікації локальних фізичних серверів і віртуальних машин у хмарі (Azure) або на додатковому сайті. При виникненні проблеми з основним розташуванням перевіряється сценарій переходу на додатковий сайт та

забезпечується наявність програм та робочих навантажень. При поверненні до нормального режиму роботи основне місце переключення на нього.

На рисунку 1.8 ми можемо бачити web-інтерфейс моніторингу віртуальних машин MS Azure. Варто відмітити наступні переваги:

– наявність усіх основних графіків для моніторингу комп’ютера;
– корисні та ілюстративні діаграми;
– інформація розподілена по різних вкладках;
– можливість гнучкого налаштування інтерфейсу;
– детальний опис про всі параметри комп’ютера.

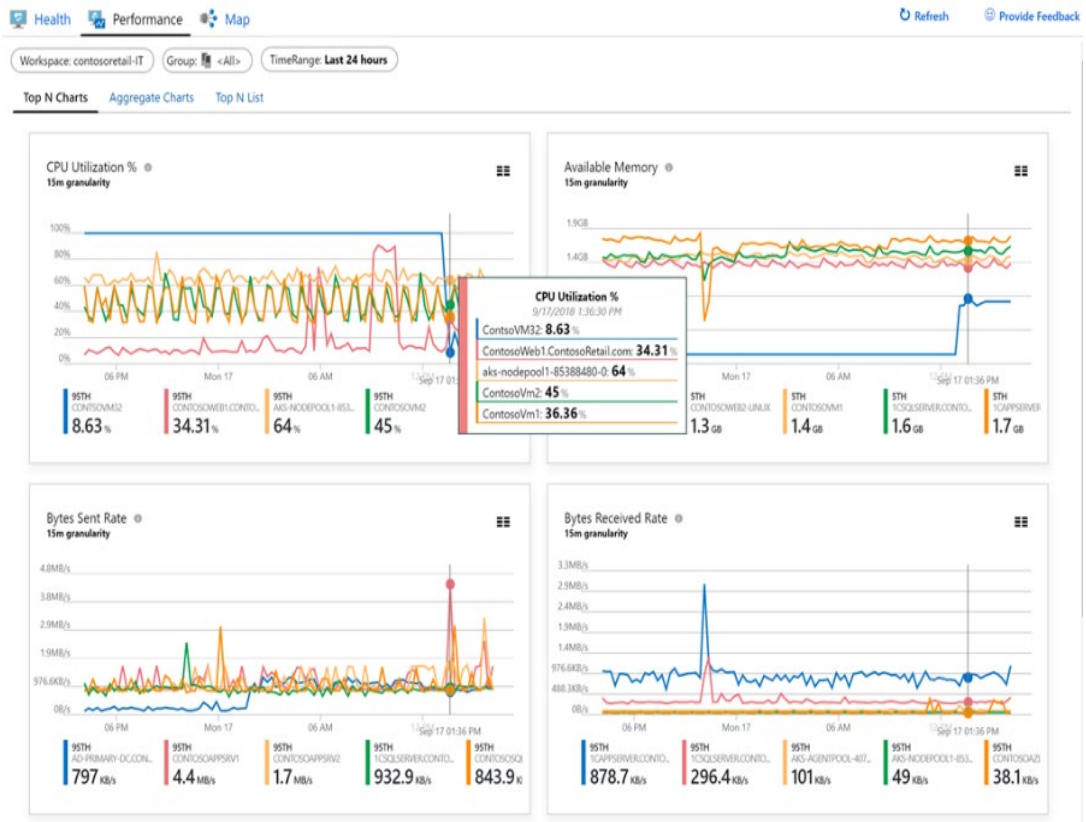

Рисунок 1.8 – Web-інтерфейс MS Azure для моніторингу характеристик системи

1.3 Формування завдань дослідження

Проведений у попередніх підрозділах аналіз показує, що моніторинг системних ресурсів операційних систем є важливою роботою будь-якої компанієї. Основним фактором успіху дослідження є такий метод моніторингу, при якому програмний продукт надає всі задані можливості та є зручним для користування. Але важливими також є і чітко сформульована мета роботи та спроектовані етапи роботи, доступність теоретичних даних та опису використання подібних методів.

Вихідні дані до розробки програмної реалізації методу є :

– фреймворки для роботи з даними, а саме збір, обробка та візуалізація;

– інструменти для збору даних про комп'ютерних системах на основі Windows, Linux, Unix та MacOS;

– розробка веб-сервісів та веб-додатку.

Моніторинг комп'ютерів є надзвичайно об'ємною за функціоналом програмою. Імплементація багатого функціоналу забезпечить конкурентоспроможність системи на ринку. Зараз можна відмітити кілька властивостей, які будуть життєвонеобхідними для продукту подібного типу:

– безвідмовність;

– інтуїтивно зрозумілий та простий користувацький інтерфейс;

– забезпечення відображення інформації в режимі реального часу.

Мінімальний необхідний функціонал системи моніторингу:

– авторизація та автентифікація користувача;

– управління організацією:

1) видалити;

2) редагувати інформацію;

3) запрошення(створення) нових користувачів;

– керування порталом системи (адміністрування порталу):

1) оновлення версій програми моніторингу, яке буде доступне користувачам для завантаження;

2) управління користувачами додатку (активувати, деактивувати, редагувати інформацію);

3) управління організаціями додатку (активувати, деактивувати, редагувати інформацію);

4) перегляд і відповіді на відгуки користувачів;

– можливість спілкування в режимі реального часу:

1) спілкування між учасниками однієї організації (групові чати);

2) спілкування між учасниками один на один;

– управління даними свого профілю:

1) можливість редагувати свої дані;

2) налаштування сповіщень;

3) зміна тем оформлення;

– управління під’єднаних комп’ютерів до користувача на порталі:

1) створення;

2) видалення;

3) редагування;

4) завантажити програму збору даних на комп’ютер;

5) переглянути звіти активності комп’ютера за різними параметрами;

6) перегляд діючих програм комп’ютера;

– налаштування робочого столу з параметрами моніторингу:

1) додавання графіку з параметрами;

2) редагування параметрів графіку;

Висновки до розділу 1

1. Зроблено огляд предметної області моніторингу системних ресурсів комп’ютера.

2. Проаналізовано відомі рішення системного моніторингу.

3. Сформовано завдання дослідження, в результаті якого буде розроблено метод моніторингу операційних систем.

2 АЛГОРИТМ МЕТОДУ ТА ЗАГАЛЬНА СТРУКТУРА СИСТЕМИ МОНІТОРИНГУ

2.1 Алгоритм методу моніторингу системних ресурсів операційних систем

Проаналізувавши відомі рішення моніторингу системних ресурсів операційних систем, було прийнято до уваги переваги та недоліки кожного з методів.

Задля створення найоптимальнішого алгоритму для початку потрібно зібрати системні дані операційної системи, які далі будуть відправлені на сервер. Веб-сервіс займається їх обробкою, по завершенню цього процесу, отримані результати зберігаються в БД. Найоптимальніше для цього використовувати MongoDB, адже вона є одним із найкращих варіантів для збереження великої кількості неструктурованих даних.

Далі можливі два варіанти: є активні клієнти для надсилання даних та їх відсутність. В другому випадку алгоритм доходить до свого логічного кінця. В першому переходить до наступного кроку – надсилання зібраних даних на веб-клієнт. Передача даних відбувається за допомогою SignalR, щоб забезпечити відображення отриманих результатів у реальному часі. без необхідності оновлення сторінки. Після того, якщо на екрані немає панелі приладів відображення даних, тоді веб-клієнт, зберігає отримані дані в кеші, щоб при можливосіт відобразити актуальні дані, в іншому випадку відобрає їх відразу.

На рисунку 2.1 представлено схему алгоритму методу моніторингу системних ресурсів операційних систем.

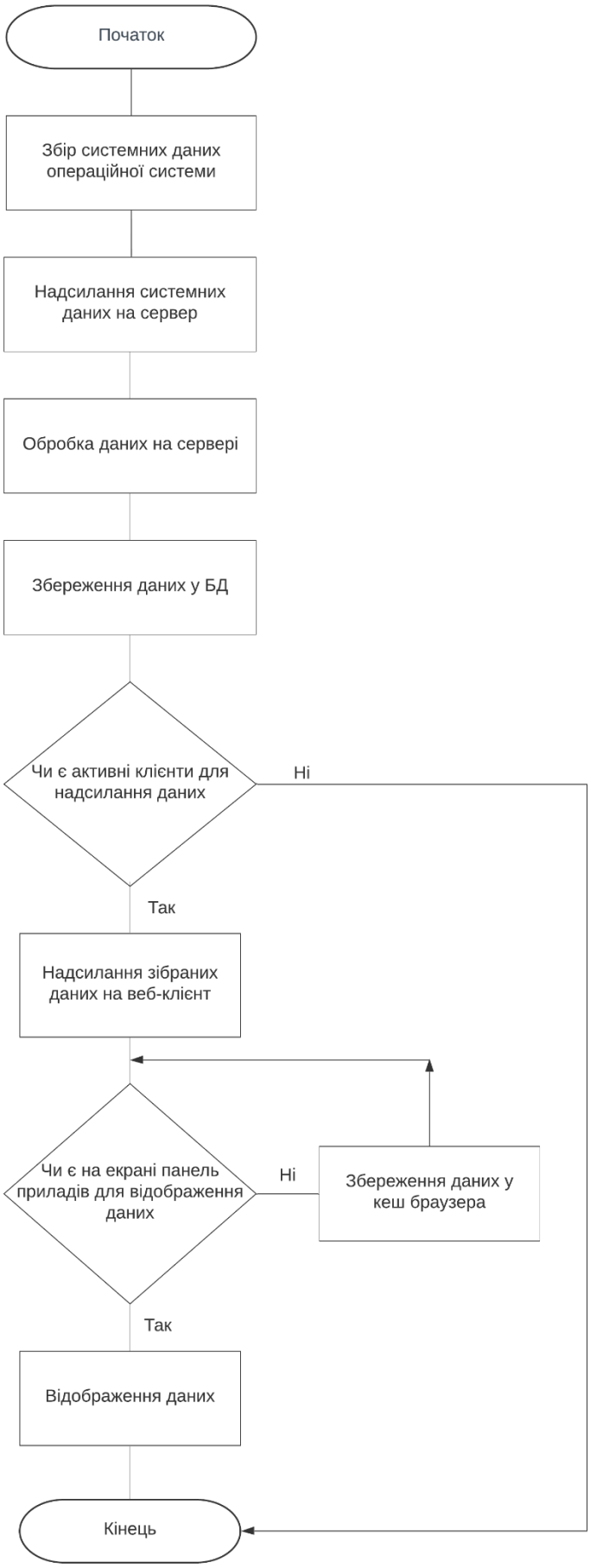

Рисунок 2.1 – Схема алгоритму методу моніторингу системних ресурсів операційних систем

2.2 Структура проектованії системи моніторингу

Програмне забезпечення на мікросервісній архітектурі клієнт-серверного типу. Тобто, заплановано два незалежні сервіса, кожен з яких буде мати свою базу даних і виконуватиме бізнес логіку над інформацією із доступними йому даними.

Ці сервіси матимуть програмний веб інтерфейс доступ до якого надається при наявності потрібного токена доступу. Також є веб-клієнт для взаємодії з системною інформацією у графічному вигляді і десктоп-клієнт для збору даних.

Головним ядром системи є веб-сервіс «Core» який надає програмний веб інтерфейс для роботи з бізнес логікою і основними даними системи в цілому. Цей сервіс дозволяє виконувати різного роду запити для створення, оновлення, фільтрації, видалення і багато інших дій над основними сутностями бази даних з віддалених клієнтів. Наразі розроблено лише веб-клієнт, але якщо в майбутнього буде потреба у клієнті для мобільного телефону чи настільного комп'ютера при чому не важливо на якій операційній системі він працює, то можна легко використати даний сервіс для роботи з бізнес логікою. Ніяких додаткових дій для адаптації серверу не потрібно проводити, адже він є клієнто незалежним. В цьому і полягає основна перевага клієнт-серверної архітектури.

Комунікація з основним сервісом виконується шляхом надсилань HTTP запитів для виконання різного роду операцій. Для встановлення постійного каналу зв'язку використовується WebSocket протокол, це особливо корисно для функціонування таких частин клієнт, як повідомлення чату, системні сповіщення чи оновлення графіків із збираною інформацією. А також для комунікації з іншими веб-сервісами виконується завдяки черг повідомлень.

В даному сервісі можна виділити такі кінцеві контролери:

– TokensController – кінцева точка, яка дозволяє реєструвати нових користувачів, проводити авторизацію та аутентифікацію користувача, отримувати дані про користувача;

– UsersController – кінцева точка, яка дозволяє отримувати та змінювати дані користувача. Отримувати відфільтровані списки користувачів системи і конкретної організації, наприклад задля адміністрування;

– RolesController – кінцева точка, яка дозволяє керувати ролями в системі;

– OrganiztionController – кінцева точка, яка дозволяє керувати організаціями в системі;

– OrganiztionInviteController – ендпоінт, який дозволяє керувати запрошеннями до організацій в системі;

– InstancesController – кінцева точка, яка дозволяє керувати електронними екземплярами комп'ютерів, запрошувати статистичну інформацію, генерувати звіти;

– DashboardsController – кінцева точка, яка дозволяє керувати панелями приладів, тобто панеллю показників;

– ChartsController – кінцева точка, яка дозволяє керувати графіками.

Ще одним веб-сервісом є сервіс «Акумулятор даних». Він служить для отримання системних метрик від десктоп-клієнта для запису їх у документовану базу даних. При отримані цього типу даних сервіс перевіряє граничні значення виставлені користувачем для аналізуючого клієнта, якщо якийсь із параметрів перевищує виставлену межу, то надсилається повідомлення через чергу повідомлень на головний сервіс. В свою чергу головний сервіс перевіряє користувацькі налаштування для всіх під'єднаних користувачів до цього комп'ютера, в разі потреби відправляє сповіщення на веб-клієнт, якщо користувач в системі і лист на електронну пошту з детальним описом проблеми.

Також кожного разу при отримані метрик з комп'ютера ці дані окрім обробки і збереження до бази даних відправляються на веб-клієнт всім користувачам, які мають доступ до відслідкованого комп'ютера, задля оновлення графіків в режимі реального часу.

Варто зазначити, що сервіс «Акумулятор даних» немає прямого зв'язку із будь-яким іншим вузлом системи. Вся комунікація із іншими частинами системи виконується через чергу повідомлень, яка є по-суті незалежним сервером із каналами, які працюють як буфери даних куди можна записувати або читати інформацію різного типу. Перевагами такого підходу комунікації є те, що навіть якщо один із сервісів перестав працювати інший і надалі працюватиме, адже черга залишається доступною завжди. Також це покращує масштабованість системи.

Ще однією важливою функцією даного серверу є агрегація даних. Оскільки під час довгого періоду збирання даних в базу даних записується велика кількість об’єктів і з часом потужності на отримання такої будуть тільки зростати, а старі дані в такій точності не несуть великої користі, то вирішено реалізувати агрегацію даних. Тобто в залежності від налаштувань виконуватиметься агрегація даних по зібраних даних за деякий період часу і осереднюватиметься. Наприклад це можна настроїти наступним чином: всі дані старші 24 годин, будуть агрегуватися по проміжках в 5 хвилин. Для кращої гнучкості агрегації, користувачу надано можливість самому настроювати це у графічному вигляді, в налаштуваннях до інстансу віртуальної машини.

Для захоплення якомога більше користувачів графічний інтерфейс вирішено створити у вигляді веб-порталу. Адже це дозволить бути йому платформо-незалежним і доступним із будь-якого пристрою в якого є доступ до мережі інтернету.

В якості основи веб-клієнта вибрано архітектуру односторінкової програми, яка буде побудована на таких технологіях як HTML і CSS для створення розмітки і її стилізації та JavaScript фреймворком Angular, для програмування клієнтської логіки. Даний клієнт повністю виконуватиметься у браузері користувача, що допоможе розгрузити навантаження із хостингових серверів і комунікуватиме із сервером шляхом здійснення запитів по HTTP або WebSocket протоколах.

Веб-клієнт повинен бути багатокористувацьким, тобто потрібно добавити авторизацію і аутентифікацію користувачів і надавати кожному з ним доступ до інформації, яка належить конкретно їм, або якою інші користувачі поділилися з ними. Також важливим є добавлення розширеної системи налаштувань над різними сутностями системи. Це можуть бути такі дані, як налаштування користувача, організації, кастомізація палені зібраних даних, виставлення граничних рівнів відслідковуваних комп’ютерів. Після добавлення всієї цієї функціональності сайт повинен залишатись простим у користуванні і не мати перевантажений інтерфейс, якщо якісь місця порталу є незрозумілими, потрібно добавити підказки користувача.

Іншим розробленим клієнтом є десктоп-клієнт «Збирач даних». Це застосунок-інсталятор, який доступний для завантаження на вікні екземпляру комп'ютера на порталі. При встановленні через інсталятор користувач повинен ввести ідентифікатор інстансу вказаний на порталі. Після виконання всіх цих кроків програма працюватиме у фоновому режимі на комп'ютері і відправлятиме кожні 5 секунд дані на спеціальний сервер «Акумулятор даних». В разі якщо немає зв'язку із сервером, або машина знаходиться в режимі офлайн, то зібрані дані зберігатимуться локально на машині до того моменту, допоки не буде встановлено зв'язок із сервером. Також програма запускається автоматично із запуском комп'ютера і не потребує ніяких додаткових дій для початку роботи. Даний клієнт розроблений на платформі .NET Core, що дозволяє йому бути мультиплатформним і виконуватись на таких системах як Microsoft Windows, Apple MacOS і на всіх популярних дистриб'ютивах Linux.

Програмне забезпечення розраховано на велику кількість користувачів, які повинні мати доступ до основного керуючого ядра. Тому вважаю, що в такому плані краще всього зарекомендує себе клієнт-серверна архітектура. А саме багаторівнева, або ще відома як трирівнева архітектура.

Трирівнева архітектура складається з таких основних вузлів:

- рівень презентації;
- програмний рівень;
- рівень даних;

Рівень презентації - кінцевий у трирівневій системі та складається з користувальницького інтерфейсу. Цей інтерфейс користувача часто є графічним, доступним через веб-браузер або веб-додаток, який відображає вміст та інформацію, корисні кінцевому користувачеві. Цей рівень часто будується на веб-технологіях, таких як HTML5, JavaScript, CSS або через інші популярні рамки веб-розробки, і спілкується з іншими рівнями за допомогою API-викликів.

Програмний рівень - це рівень, на якому знаходиться вся бізнес-логіка програми. Тут знаходиться код, який містить всю бізнес логіку, роботу з різними сторонніми та внутрішніми API. Application Layer це по суті посередник, що є проміжним між веб-рівнем(презентаційним рівнем), та рівнем бази даних.

Рівень даних складається з бази даних та або системи зберігання даних та рівня доступу до даних. Прикладами таких систем є MySQL, Oracle, PostgreSQL, Microsoft SQL Server, MongoDB тощо. Доступ до даних здійснюється на рівні програми через виклики API.

На рисунку 2.2 зображено приклад структури трирівневої архітектури.

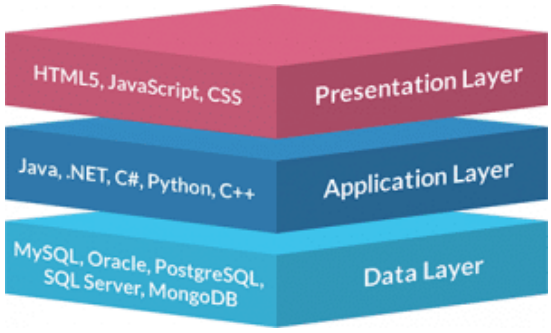

Рисунок 2.2 – Структура трирівневої архітектури

Типова структура для трьохрівневого розгортання архітектури повинна мати рівень презентації, розміщений на десктопі, ноутбуці, планшеті або мобільному пристрої через веб-браузер або веб-додаток, що використовує веб-сервер.

Базовий рівень програми зазвичай розміщується на одному або декількох серверах прикладних програм, але також може розміщуватися у хмарі або на спеціальній робочій станції залежно від складності та потужності обробки, необхідної додатку. Рівень даних зазвичай складається з однієї або декількох реляційних баз даних, великих джерел даних або інших типів систем баз даних, розміщених або на локальному рівні, або в хмарі.

Також я б хотів добавити декілька слів про мікросервісну архітектуру. Вона також частково використовуватиметься в спроектованому додатку. По структурі її можна назвати як сукупність незалежних багаторівневих архітектур.

Мікросервіси – незалежні програмні служби, які забезпечують певну бізнес-функціональність в програмному забезпеченні. Ці служби можуть обслуговуватися, контролюватися і розгортатися незалежно. Мікросервіси побудовані на основі сервіс-орієнтованої архітектури. Мікросервісну архітектуру зображено на рисунку 2.3.

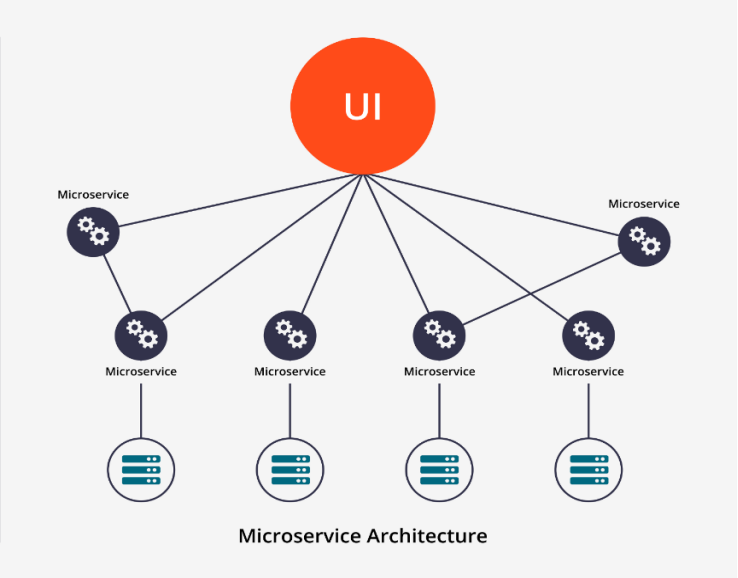

Рисунок 2.3 – Мікросервісна архітектура

Опираючись на найкращі практики, які сьогодні використовуються при розробці високонавантажених систем, було спроектовано архітектуру програмного забезпечення для моніторингу комп’ютерних систем, що показана на рисунку 2.4.

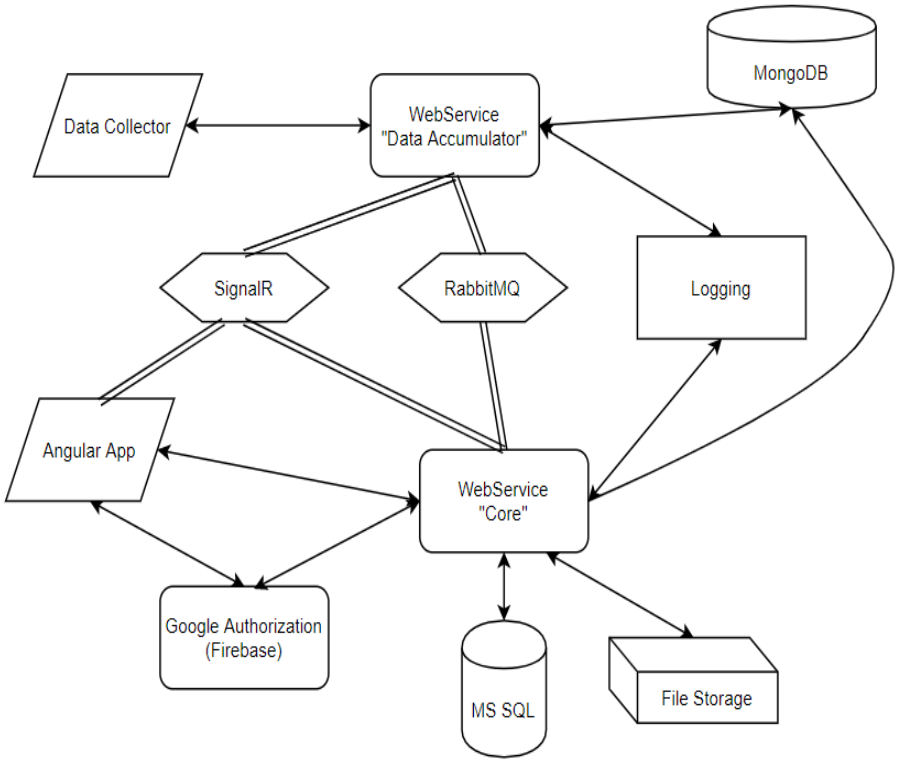

Рисунок 2.4 – Архітектура ПЗ для моніторингу комп’ютерних систем

На рисунку 2.5 показано список дій/викликів, які здійснює проектована система протягом усього процесу збору даних системних ресурсів.

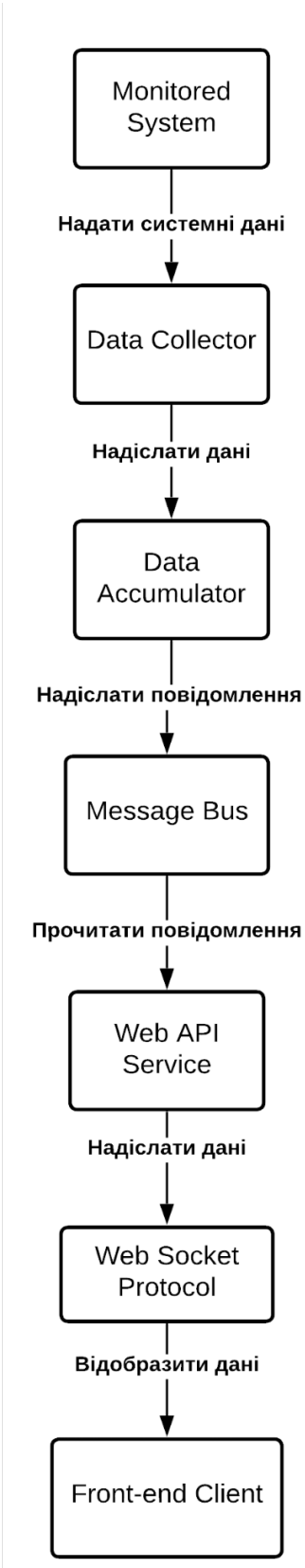

Рисунок 2.5 – Список дій/викликів, які здійснює проектована система

2.3 Визначення та опис сценаріїв використання системи

Клієнтам системи потрібно дати можливість створювати віртуальні екземпляри комп'ютерів для того, щоб збирати дані про роботу їхньої системи, при чому такі дані повинні бути приватними і доступні лише для авторизованих користувачів з потрібним рівнем доступу. Тому потрібно використати такий підхід як авторизація та аутентифікація.

Аутентифікація – це процес перевірки особи чи пристрою. Поширений приклад – це введення імені користувача та пароля під час входу на веб-сайт.

Введення правильної інформації для входу дозволяє веб-сайту знати хто ви для визначення доступу до певного ресурсу. Фактори автентифікації беруть до уваги велику кількість різноманітних критеріїв, що використовуються системою для перевірки особистості перед наданням індивідуального дозволу на доступ для того, щоб читати, редагувати, створювати чи видаляти певні ресурси. Особистість людини повинна визначатися тим, що вона знає, і коли мова йде про безпеку, необхідно перевірити як мінімум два або всі три фактори аутентифікації, щоб дати комусь дозвіл на доступ до системи. Залежно від рівня безпеки чинники автентифікації можуть відрізнятися від одного з наступних:

– однофакторна аутентифікація – це найбільш простий та швидкий шлях, задля визначення свого місця в системі. Щоб автентифікуватись на певному веб-ресурсі за допомогою цього методу потрібен тільки логін і секрет(пароль). Щоб отримати доступ у систему, персона має використати тільки один із облікових записів, щоб підтвердити свою особу;

– двофакторна аутентифікація – вона вимагає правильного входу, а також іншої перевірки. Наприклад, якщо ви ввімкнули 2FA для свого банківського рахунку в Інтернеті, вам може знадобитися ввести тимчасовий код, надісланий на ваш телефон або електронну адресу, щоб завершити процес входу. Це гарантує, що тільки ви або хтось із хто має доступ до вашого телефону чи облікового запису електронної пошти можете отримати права на доступ до свого облікового запису навіть за умови введення правильної інформації для входу;

– багатофакторна аутентифікація – це найбільш просунутий з усіх методів автентифікації, що вимагає два чи більше рівні захисту.

У додатку використано аутентифікацію та авторизацію, задля забезпечення захисту даних всіх користувачів та організацій.

В таблицях 2.1 - 2.9 показано опис структури сутностей програмної системи.

Таблиця 2.1 – Модель користувача

Назва поля	Тип даних	Детальний опис
Id	String	Ідентифікатор
FirstName	String	Ім’я
LastName	String	Прізвище
DisplayName	String	Ім’я що буде відображатись
Email	String	Електронна пошта користувача
PhotoURL	String	Лінк на фото користувача
Bio	String	Коротка інформація про користувача
RoleId	Int32	Ід ролі користувача
NotificationSettings	Array<NotificationSetting>	Налаштування сповіщень
Feedback	Array<Feedback>	Відгуки користувача
UserChats	Array<UserChat>	Чати користувача
OrganizationInvites	Array<InviteToOrganization>	Запрошення в організацію, які створив користувач

Таблиця 2.2 – Модель ролі

Назва поля	Тип даних	Детальний опис
Id	String	Ідентифікатор
Name	String	Назва
AccessLevel	Int32	Рівень доступності

Користувач авторизується через зовнішні сервіси (Google, Facebook, Github), що надає змогу використати багатофакторну аутентифікацію. Для цього було використано сервіс Google Firebase. Частина необхідної інформації, наприклад: ім’я, прізвище, фото, емейл надсилається з Google, Facebook чи Github використовуючи публічний API використаного сервісу. Вибрати сервіс та заповнити назву організації – це все що треба зробити новому юзеру для авторизації при реєстрації нового облікового запису на порталі. Цей підхід сильно полегшує та пришвидшує вхід на веб-сайт, так як користувачу не доводиться витрачати час на заповнення всієї інформації про себе, підтвердження електронної пошти. І що немало важливо, користувач може бути впевнений, що при такому виді аутентифікації приватні дані, такі як логін і пароль не зберігається на кінцевому порталі. На рисунку 2.4 зображена саме ця функціональність.

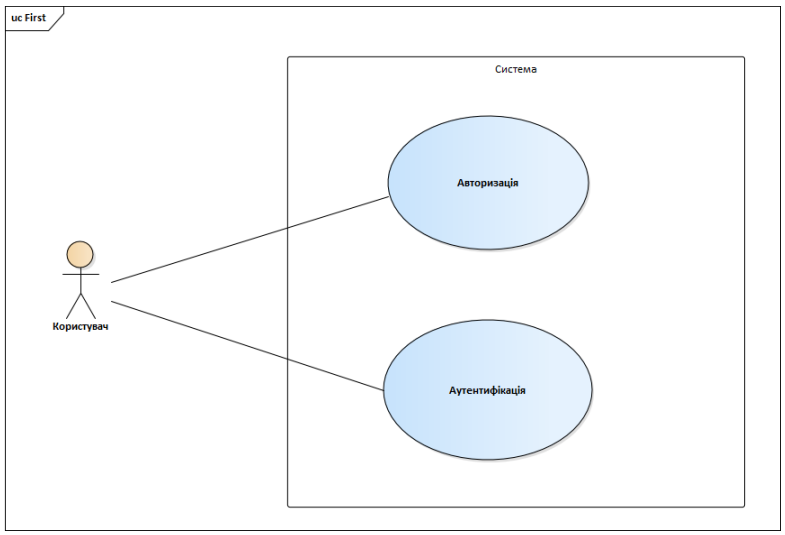

Рисунок 2.6 – Прецедент «Аутентифікація та авторизація користувача»

Система має працювати з великою кількістю користувачів та витримувати високі навантаження. Тому, для розподілу всіх користувачів по групам система має мати функціонал для управління організаціями, користувачами у ній та правами доступу до ресурсів. Залучення такої сутності як організація набагато полегшить задачу по керування багаться користувачами та їхніми комп’ютерами. Можна буде розподілити користувачів по підрозділах та дати їм права адміністраторів. Це

дозволить зробити систему підтримки великих підприємств добре масштабованою та організованою.

Таблиця 2.3 – Модель організації

Назва поля	Тип даних	Детальний опис
Id	Int32	Ідентифікатор
Name	String	Назва
Description	String	Опис організації
Email	String	Електронна пошта
ContactNumber	String	Телефонний номер
ImageURL	String	Посилання на фото
CreatedByUserId	String	Ідентифікатор юзера, який створив організацію
ThemeId	Int32	Ідентифікатор теми, яка застосована
Instances	Array<Instance>	Екземпляри комп’ютерів, які знаходяться в організації
OrganizationInvites	Array<InviteToOrganization>	Запрошення до організації

Таблиця 2.4 – Модель запрошення в організацію (InviteToOrganization)

Назва поля	Тип даних	Детальний опис
Id	String	Ідентифікатор
Link	String	Посилання на реєстрацію
OrganizationId	Int32	Ід організації, для якої належить запрошення
InviteEmail	String	Електронна пошта куди відправили запрошення
InvitedUserId	String	Ід запрошеного користувача
CreatedByUserId	String	Ід користувача, що запросив

Адміністратори порталу повинні мати права на керування цими організаціями, а саме, мати доступ до їх створення, видалення, редагування та налаштування.

Для того щоб додавати нових користувачів у систему чудовим рішення буде просто їх запросити до організації. Це полегшить реєстрацію для користувачів, так як їм потрібно буде лише прийняти запрошення та дозаповнити недостаючу інформацію про нього. Цей процес є найоптимальнішим та найменш ресурсозатратним. На рисунку 2.5 зображено вищеописаний прецедент «Керування організацією».

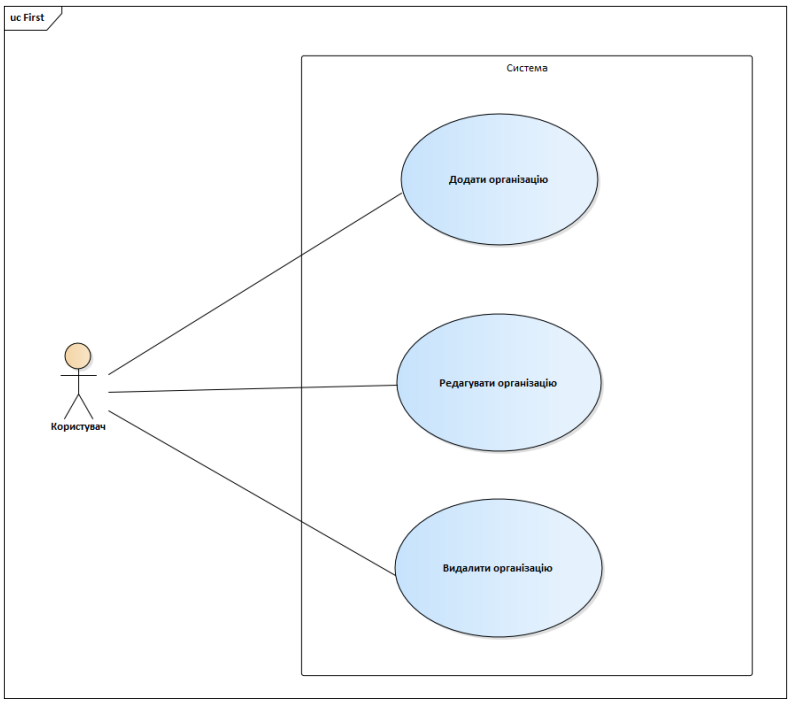

Рисунок 2.7 – Прецедент «Керування організацією»

Керування порталом – важка справа, адже в системі можуть бути багато працівників однієї компанії. А також це досить відповідальна справа, адже від цього залежить якість системи в цілому. Частина системи для адміністраторів має мати функціонал, який дозволить керувати всіма процесами, які відбуваються у системі. Зокрема, оновлення версії програмного забезпечення, яке призначене для збирання інформації з комп’ютера. Така функція дозволить швидко замінити програму. Наприклад, якщо новіша версія програми виявилася з помилками і так

званими “багами” та не дає коректно провести моніторинг комп’ютера, то адміністрація порталу повинна негайно замінити цю версію на попередню, з якою не було проблем. Таким чином можна гарантувати те, що додаток буде стабільно та безвідмовно працювати. Діаграму цього прецеденту зображено на рисунку 2.6.

Таблиця 2.5 – Модель завантаженої програми моніторингу

Назва поля	Тип даних	Детальний опис
Id	Int32	Ідентифікатор
ExeLink	String	Посилання на програму для Windows
CreatedAt	DateTime	Дата, завантаження нової версії додатку
TgzLink	String	Посилання для всіх інші Linux системи
DebLink	String	Посилання на програму для системи Debian
Version	String	Версія

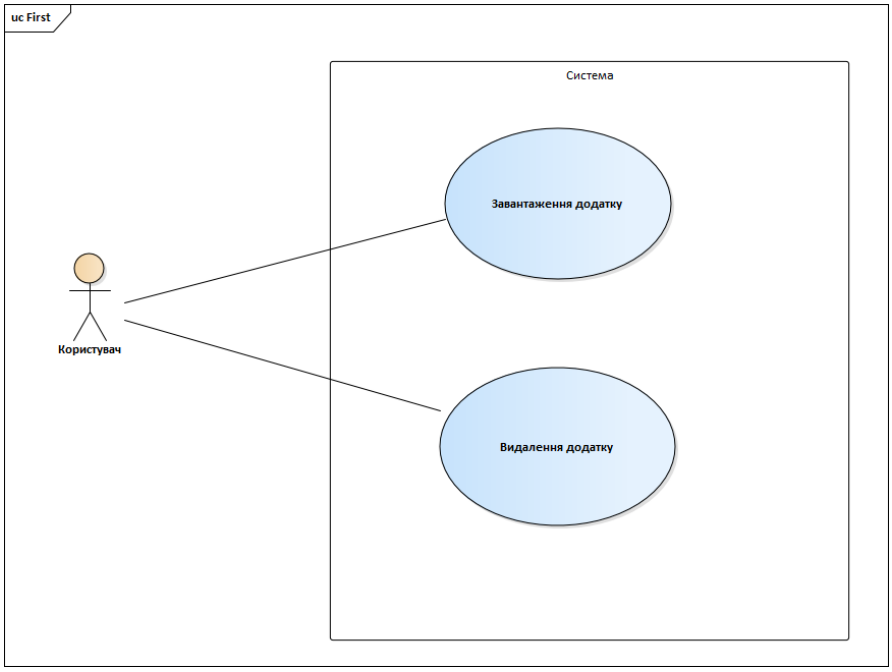

Рисунок 2.8. Прецедент «Завантаження нові версії програми моніторингу»

Управління організаціями, що зображене на рисунку 2.7 дозволяє слідкувати та змінювати стан організацій. У разі розформування якогось підрозділу, потрібно видалити організацію, тому що вона вже не потрібна.

Якщо в організації змінилися якісь контактні дані, то нам потрібно їх оновити для того, щоб інформація була завжди актуальною.

Управління користувачами дозволить тримати в системі лише активних користувачів. У разі звільнення чи відмови від персонального комп'ютера, адміністратор організації має деактивувати юзера. Варто замітити, що при видалені сутностей з бази даних, вони не видаляються фізично, а просто позначаються неактивними. Такий підхід називається – м'яким видаленням і служить для того щоб в разі потреби раніше видалені дані можна було повернути. Редагування інформації користувача також допоможе підтримувати актуальні дані в системі.

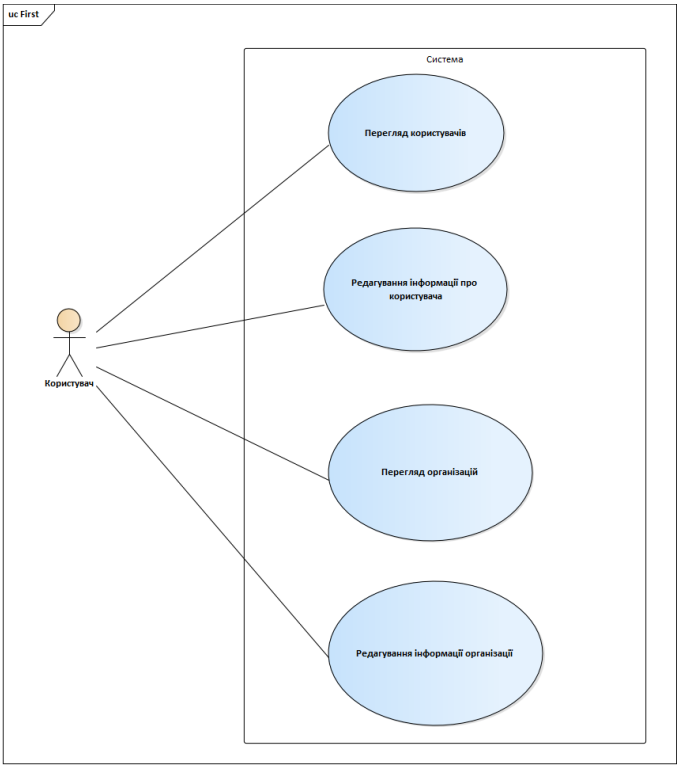

Рисунок 2.9 – Прецедент «Управління організаціями та користувачами»

Можливість переглядати користувацькі відгуки надасть змогу отримувати фідбек про систему, її помилки та недоліки. Це дозволить швидко та якісно виправляти баги та змінювати функціонал таким чином, щоб користувачі програми постійно були нею задоволені. Клієнтам дуже подобається такий підхід до

розробки додатків, тому що для них надзвичайно важливо зручно почуватись у додатку та з легкістю ним користуватися.

Комунікація користувачів на порталі це необхідність, яку потрібно надати. Звичайно, використання окремих месенджерів, емейлів, телефонів це не вихід, бо призведе до незручностей коли люди почнуть використовувати все зразу. Наприклад, для використання електронної пошти потрібно створити обліковий запис, та надати потрібну для спілкування інформацію для співробітників. Окремі програми для спілкування вимагають їх інсталяції на комп'ютер чи мобільний телефон та реєстрації.

Тому найбільш прийнятим вирішенням цієї проблеми буде інтеграція свого шляху комунікації, що призведе до швидкого та легкого спілкування користувачів порталу. Тому у разі виникнення серйозної проблеми, працівники швидко дізнаються про це через систему комунікації, що також підвищує надійність та безвідмовність всього додатку.

Є два види комунікації:

– між юзерами один на один (приватні повідомлення);

– усередині організації (група людей що входить до організації).

Таким чином можна легко розділити інформацію, якою можна ділитися. Інформація для членів певної організації повинна нести певну важливий меседж, що вимагає негайної реакції. Для прикладу, сповістити що будуть проведені ремонт всієї комп'ютерної системи, або її частини. Особисті повідомлення будуть здебільшого використовуватися для переписок між учасниками організацій. Також у них буде набагато детальніша інформація, з урахуванням користувачів та їх ролей у компанії.

Інформація про користувача зазвичай часто змінюється, наприклад: електронна адреса, фото, номер телефону і так далі. Тому необхідно дати можливість користувачу швидко та просто змінювати ці дані.

Роль сповіщень у системах важко переоцінити, адже за їх допомогою можна швидко проінформувати користувачів про будь які критичні зміни й надзвичайні ситуації. Отож це буде хорошою можливістю для користувача, щоб він обирав пріоритет тих чи інших сповіщень та налаштував спосіб, яким система

інформуватиме його про нове сповіщення. На рисунку 2.8 зображена функціональність чату.

Таблиця 2.6 – Модель чату

Назва поля	Тип даних	Детальний опис
Id	Int32	Ідентифікатор
Name	String	Назва
Type	ChatType	Тип
CreatedById	String	Ідентифікатор користувача, що створив чат
Messages	Array<Message>	Список повідомлень
UsersSettings	Array<NotificationSetting>	Налаштування сповіщень у чаті
UserChats	Array <UserChat>	Список користувачів, які відносяться до чату

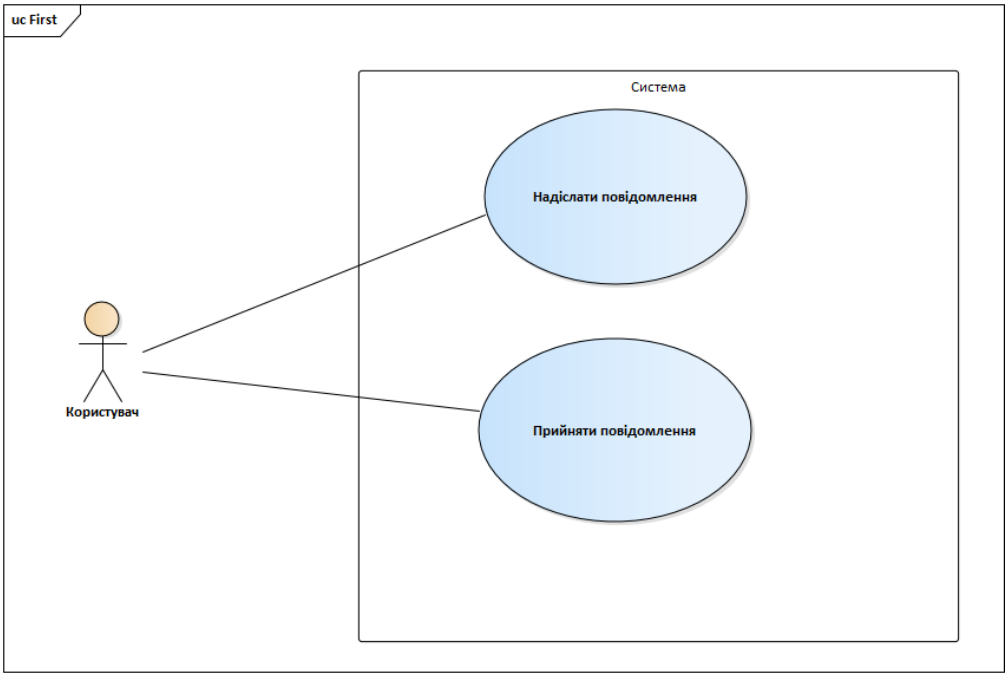

Рисунок 2.10 – Прецедент «Спілкування у реальному часі»

Екземпляр комп’ютера – це прототип реального комп’ютера, який відслідковується користувачем. Користувачі можуть керувати електронним екземпляром комп’ютера, змінювати різного роду налаштування. Тут можна

виставляти граничні значення показників комп'ютера при досягненні яких користувач отримає сповіщення. Це можуть бути сповіщення як на порталі, а також вони можуть бути продубльовані на електронну пошту. На рисунку 2.9 показано можливості правління екземпляром комп’ютера на порталі.

Таблиця 2.7 – Модель екземпляра комп'ютера

Назва поля	Тип даних	Детальний опис
Id	Int32	Ідентифікатор
Title	String	Назва
Platform	String	Операційна система комп’ютера
AggregationForHour	Boolean	Чи агрегуватимутся дані щогодини
AggregationForDay	Boolean	Чи агрегуватимутся дані щодня
AggregationForWeek	Boolean	Чи агрегуватимутся дані щотижня
AggregationForMonth	Boolean	Чи агрегуватимутся дані щомісяця
CpuMaxPercent	Float	Максимум % завантаженості процесору
RamMaxPercent	Float	Максимум % завантаженості оперативної пам’яті
DiskMaxPercent	Float	Максимум % заповненості накопичувальної пам’яті
OrganizationId	Int32	Ід організації, до якої належить комп’ютер
Dashboards	Array<Dashboard>	Список панелей приладів

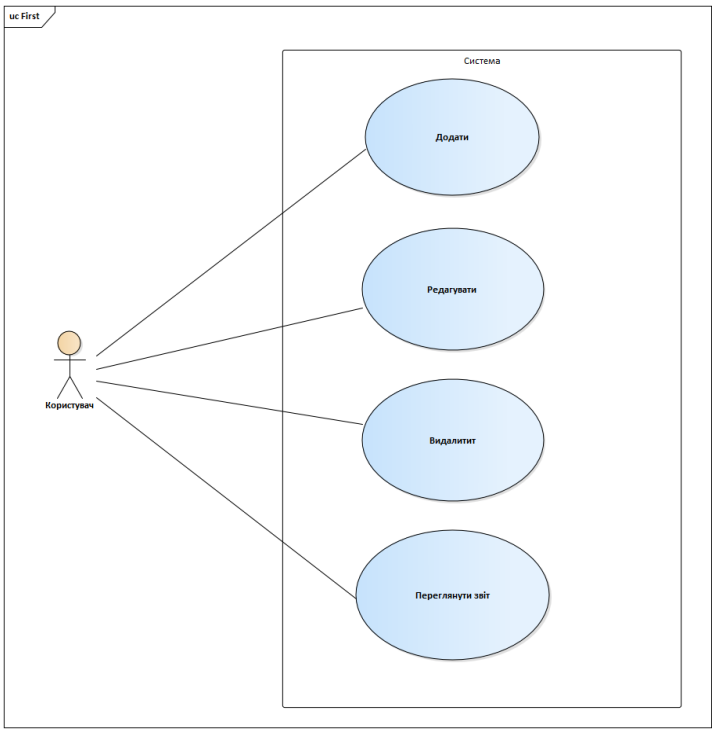

Рисунок 2.11 – Прецедент «Управління екземпляром комп’ютера на порталі»

Система має мати гнучкий інтерфейс для налаштування графіків, що моніторять систему. Це дасть змогу дозволить швидко та зручно знайти необхідну інформацію про систему та відстежувати зміну цих параметрів. Ці елементи користувацького інтерфейсу будуть основними інструментами, які клієнти використовуватимуть для того, щоб моніторити систему.

Панель приладів – це елемент для менеджменту інформації, що відстежує, відображає та аналізує ключові показники ефективності, метрики і ключові дані системи чи окремого процесу цієї системи. Ці інструменти повинні бути налаштовані в залежності від потреби певної системи та користувача.

Панель приладів є найефективнішим способом для того, щоб відстежувати кілька джерел, з яких надходять дані. Моніторинг в режимі реального часу скорочує час аналізу даних. Панель приладів складається з графіків і таблиць, вигляд яких можна налаштувати вручну. Для цього є багато налаштувань, все це дозволяє досить гнучко налаштовувати панель даних. Також слід дозволити створення декількох панелей приладів для екземпляра комп’ютера.

На рисунку 2.10 показано функціонал налаштування панелі приладів для моніторингу.

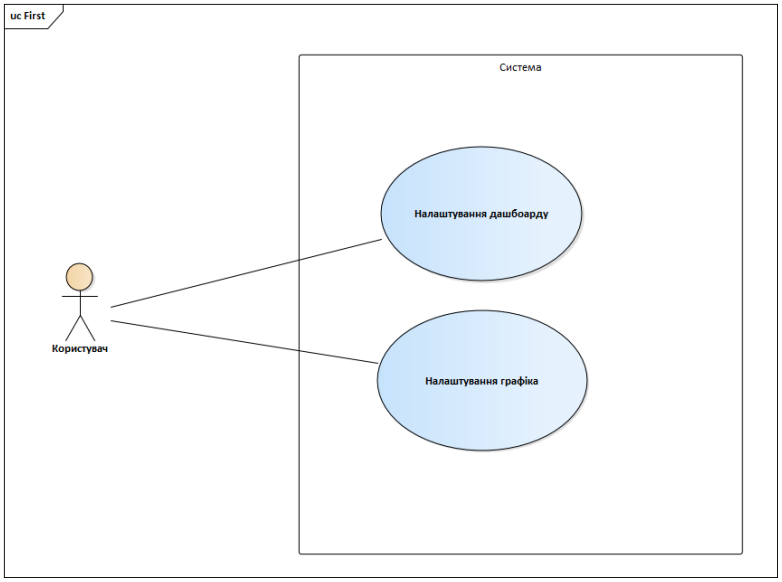

Рисунок 2.12 – Прецедент «Налаштування панелі приладів для моніторингу»

Таблиця 2.8 – Модель панелі приладів

Назва поля	Тип даних	Детальний опис
Id	Int32	Ідентифікатор
Title	String	Назва
CreatedAt	DateTime	Дата створення
InstanceId	Int32	Ід комп’ютеру, якому належить панель приладів
Charts	Array<Chart>	Графіки що належать панелі приладів

Таблиця 2.9 – Модель графіка

Назва поля	Тип даних	Детальний опис
Id	Int32	Ідентифікатор
DashboardId	Int32	Ід панелі приладів, якій належить графік
ShowLegend	Boolean	Показувати Легенду

Gradient	Boolean	Показує чи включений градієнт у графіку
ShowXAxis	Boolean	Показувати дані по осі X
ShowYAxis	Boolean	Показувати дані по осі Y
YAxisLabel	String	Позначка Y осі
XAxisLabel	String	Позначка X осі
ShowGridLines	Boolean	Показує чи відображаються лінії
RangeFillOpacity	Double	Прозорість графіку
RoundDomains	Boolean	Заокруглювати інтервали на графіку
IsTooltipDisabled	Boolean	Вимнкути текст підказок
Type	ChartType	Тип
Sources	String	Джерело даних

2.4 Інформаційне забезпечення системи моніторингу

Програмне забезпечення міститиме два типи даних. Перший тип – це зібрані метрики з комп’ютера користувача. Такі дані складатимуться з великої кількості об’єктів, а саме з основних показників системи в цілому і масиву значень для окремих виконуваних процесів. Структуру цих даних можна побачити в таблиці 2.1.

За налаштуваннями по замовчуванню ці дані збиратимуться з комп’ютера кожні 10 секунд і відправлятимуться на сервер для їх подальшої обробки і запису до бази даних. Оскільки в один момент часу можуть бути підключені сотні комп’ютерів дані яких потрібно обробляти, зберігати і відправляти на клієнтську частину, то потрібно максимально можливо оптимізувати цей сервер.

З цим завданням добре справиться документована база даних, а саме використано базу даних із назвою Mongo DB. При такому підході кожна порція даних записується до бази даних як документ без всяких зв’язків з іншими сутностями. Це дозволяє отримати великий приріст швидкості порівняно з реляційною базою даних при записі і читанні з бази даних.

Фрагменти програмного коду класів, що складають основний домен програми та використовуються для збору даних для операційних системи Linux та Windows представлені у Додатку А.

В таблицях 2.10 та 2.11 показано опис структури сутностей зібраних даних системи.

Таблиця 2.10 – Модель зібраних даних процесу

Назва поля	Тип даних	Детальний опис
Name	String	Назва процесу
RamMBytes	Float	Кількість використаної оперативної пам’яті в мегабайтах
PRam	Float	Кількість використаної оперативної пам’яті у процентах
PCpu	Float	Величина використаного ресурсу процесора у процентах

Другий тип даних – бізнес дані з глибокими зв’язками між собою. Для таких даних використана реляційна база даних MS SQL Server. Структуру якої можна поглянути в на схемі «Структура бази даних» в прикріплених додатках до роботи. Тут зображено всі зв’язки між різними сутностями, які можуть бути один до одного, один до багатьох і багато до багатьох, яка реалізується через проміжну таблицю.

Таблиця 2.11 – Модель зібраних даних системи

Назва поля	Тип даних	Детальний опис
Id	Int32	Ідентифікатор
ClientId	Int32	Ід машини, на якій збираються дані
InterruptsTimePercent	Float	Значення переривань процесора у процентах
ProcessesCount	Int32	Кількість виконуваних процесів
UsageRamMBytes	Float	Кількість використаної оперативної пам’яті МБ
TotalRamMBytes	Float	Всього доступної оперативної пам’яті у мегабайтах
RamUsagePercentage	Float	Кількість використаної оперативної пам’яті у процентах
LocalDiskUsageMBytes	Float	Кількість використаної пам’яті на накопичувачі у мегабайтах
LocalDiskUsagePercentage	Float	Кількість використаної пам’яті на накопичувачі у процентах
CpuUsagePercentage	Double	Величина використаних ресурсів процесора у відсотках
Time	DateTime	Час збору даних
Processes	List<ProcessData>	Список запущених процесів

На рисунку 2.13 показано UML діаграму класів зібраних даних процесу та системи.

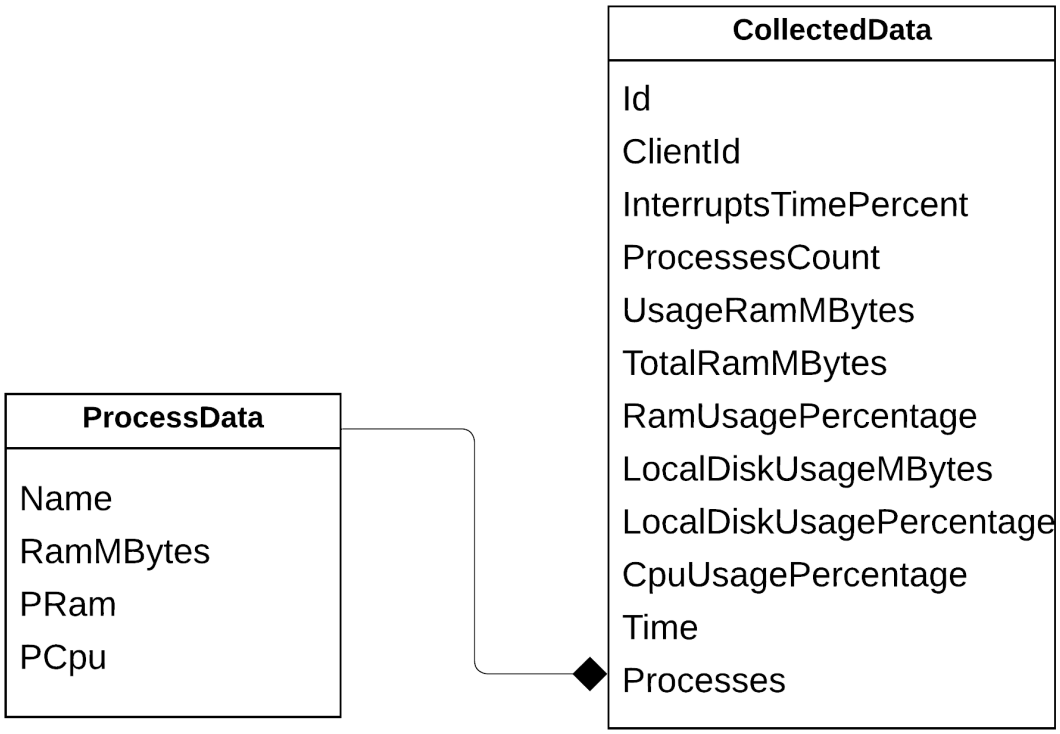

Рисунок 2.13 – UML діаграму класів ProcessData і CollectedData

Висновки до розділу 2

1. Розроблено алгоритм методу моніторингу системних ресурсів операційних систем.

2. Розроблено структуру проектованої системи.

3. Визначено та описано сценарії використання моніторингу системних ресурсів операційних систем.

4. Розроблено інформаційне забезпечення системи моніторингу.

3 РЕАЛІЗАЦІЯ ТА ДОСЛІДЖЕННЯ МЕТОДУ МОНІТОРИНГУ СИСТЕМНИХ РЕСУРСІВ

3.1 Платформа та інструменти реалізації

В якості платформи програмування було вибрано .NET Core, тому що в мене є достатній досвід програмування на даній платформі, а також тут присутні всі засоби для успішної реалізації поставленого завдання.

Microsoft .NET Core – багато-платформна програмна система. Вона має кілька ключових особливостей, таких як підтримка декількох мов програмування, асинхронні та паралельні моделі програмування та нативність, що дозволяє забезпечити широкий спектр сценаріїв на різних платформах. Однією з головних можливостей .NET є сумісність служб, написаних різними мовами програмування.

Платформа .NET Core містить багатомовне середовище виконання (середовище CLR) і бібліотеки класів. Що, серед іншого, визначає незалежність від мови виконання та інтероперабельність мови. Це означає, що ви вибираєте будь-яку мову .NET для створення додатків і служб у .NET. Microsoft активно розробляє та підтримує три мови .NET: C#, F# і Visual Basic (VB).

C# є простим, потужним, безпечним для типів та об'єктно-орієнтованим, зберігаючи виразність та вишуканість мов у стилі C. Усі, хто знайомий зі C та подібними мовами, знаходять мало проблем у адаптації до C#.

F# - це кросплатформна, функціональна мова програмування, яка також підтримує традиційне об'єктно-орієнтоване та імперативне програмування.

Visual Basic - це проста мова, що використовується для створення різноманітних програм, які працюють на .NET. Серед мов .NET синтаксис VB найбільш близький до звичайної людської мови, часто полегшений для людей, які не знайомі з розробкою програмного забезпечення.

Середовище виконання можна вважати агентом, який надає основні служби, такі як управління потоками, управління пам'яттю, керує кодом під час виконання, віддалену взаємодію. Середовищем контролюється суворі умови типізації та інші види перевірки точності коду, що забезпечують надійність і безпеку. Фактично

основним завданням середовища виконання є управління кодом. Код, який керується середовищем виконання, називається керованим кодом, а код, який виконується в обхід середовища виконання, називають некерованим кодом.

Платформа є строго типізованою, де об'єкт – це завжди екземпляр певного типу. Єдині операції, дозволені для даного об’єкта, - це його тип. Програма викликає лише методи, що належать даному типу. Усі інші виклики призводять до помилки часу компіляції або винятку під час виконання у разі використання динамічних функцій або об'єкта.

Але також і є можливість виконання небезпечного коду. Залежно від мовної підтримки, CLR дозволяє отримувати доступ до нативної пам'яті та виконувати арифметику вказівника через небезпечний код. Ці операції необхідні для певних алгоритмів та сумісності системи. Незважаючи на потужність, використання небезпечного коду не перешкоджає, якщо не потрібно взаємодіяти з системними API та впровадити найефективніший алгоритм. Небезпечний код може не виконуватись однаково в різних умовах, а також втрачає переваги сміттєзбірника та безпеку типу. Рекомендується максимально обмежити та централізувати небезпечний код і ретельно перевірити його [8].

Слід добавити, що .NET Core платформа має чудову підтримку асинхронності. Асинхронне програмування - це першокласна концепція в .NET з підтримкою асинхронності під час виконання, бібліотеками фреймворків та мовними конструкціями .NET. Внутрішньо вони засновані на об'єктах, таких як завданнях, які використовуються операційною системою для максимально ефективного виконання завдань.

В якості фреймворку для побудови клієнтської частини було обрано Angular. Це одне з найпопулярніших сучасних рішень для побудови клієнтського інтерфейсу. Його перевага полягає в тому, що всі графічні обчислення обчислюються на клієнтській стороні. В нашому випадку в браузері. Також це хороший вибір при використанні разом із .NET Core, тому що платформи близькі по структурі, якщо порівнювати такі мови програмування як C# і TypeScript. Така схожість полягає в тому, що обидві мови були розроблені компанією Microsoft.

Angular - це платформа для створення додатків на стороні клієнта використовуючи HTML і TypeScript. Компоненти визначають відображення, які є наборами елементів HTML, що фреймфорк може обирати і змінювати відповідно до логіки вашої програми. У компонентах застосовуються сервіси, що в свою чергу надають певний інкапсульований функціонал, що не пов'язаний безпосередньо з відображеннями. Для того щоб зробити програмний код повторно використовуваним, ефективним та модульним, можна запровадити зовнішні модулі як залежності до компонентів [7].

Для того щоб запускати та розгортати веб-сервіси чудово підійде технологія Docker. Docker — це інструментарій для управління ізольованими Linux і Windows контейнерами. Docker доповнює інструментарій LXC більш високорівневим API, що дозволяє керувати контейнерами на рівні ізоляції окремих процесів. Зокрема, Docker дозволяє не переймаючись вмістом контейнера запускати довільні процеси в режимі ізоляції і потім переносити і клонувати сформовані для даних процесів контейнери на інші сервери, беручи на себе всю роботу зі створення, обслуговування і підтримки контейнерів.

Docker допомагає:

- мінімально використовувати ресурси;
- зручно приховати фонові процеси;
- просто масштабовувати;
- зменшити час між написанням і запуском коду;
- швидше тестувати;
- швидко розгортати;
- швидше створювати додатки.

Docker робить це за допомогою легкої платформи контейнерної віртуалізації.

У своєму ядрі docker дозволяє запускати практично будь-який додаток, безпечно ізольований в контейнері. Безпечна ізоляція дозволяє запускати на одному хості багато контейнерів одночасно.

Докер характеризується досить простим синтаксисом. Тому він досить простий. Програмне забезпечення сумісне з усіма версіями операційних систем Linux і Windows, тому сфера застосування Docker практично не обмежена.

Компоненти і сервіси - це просто класи з декораторами, які відзначають їх тип і надають метадані, які повідомляють Angular, як їх використовують. Метадані для класу компонента пов'язують його з шаблоном, який визначає уявлення. Шаблон поєднує в собі звичайний HTML з директивами Angular і розміткою прив'язки, які дозволяють Angular змінювати HTML перед його відображенням.

Оскільки програмне забезпечення розроблено на платформі .NET Core 6.0 з використанням мови програмування C#, тому в якості інтегрованого середовища розробки вибраного «рідне» середовище Visual Studio 2021.

Visual Studio включає редактор коду, що підтримує IntelliSense (компонент доповнення коду), а також рефакторинг коду. Інтегрований налагоджувач працює як налагоджувач на рівні джерела, так і налагоджувач на рівні машини. Інші вбудовані інструменти включають кодовий профайлер, конструктор для побудови програм GUI, веб-дизайнер, дизайнер класів та конструктор схем бази даних.

Редактор коду приймає плагіни, які покращують функціональність майже на кожному рівні, включаючи додавання підтримки для систем керування джерелами (наприклад, Subversion і Git) та додавання нових наборів інструментів, таких як редактори та візуальні дизайнери для мов, що задаються доменом або набори інструментів для інших аспектів розробки програмного забезпечення життєвий цикл.

Вбудований відлагоджувач коду можна застосовувати як для відлагодження вихідного коду, так і в якості відлагоджувача машинного рівня.

Сучасна система налагодження Visual Studio виконує налагодження коду в емуляторі, наприклад для Windows Mobile чи Android пристроїв, на віддаленому пристрої чи локальному проекті. Можна покроково в один оператор переглядати код, перевіряючи значення змінних. Можна задати точки зупинок, що будуть спрацьовувати тільки при виконанні зазначеної умови. Також можна відстежувати значення змінних під час виконання коду і багато інших корисних для тестування

програмного забезпечення функцій. Все це можна контролювати в самому редакторі коду, не залишаючи вікно з кодом.

Visual Studio не підтримує жодної мови програмування, рішення чи інструменту внутрішньо. Натомість це дозволяє підключати функціональність, кодовану як VSPackage. Після встановлення функціональність доступна як Сервіс. IDE надає три послуги: SVsSolution, яка надає можливість перераховувати проекти та рішення; SVsUIShell, що забезпечує функцію вікон та інтерфейсу користувача (включаючи вкладки, панелі інструментів та вікна інструментів); та SVsShell, яка займається реєстрацією VSPackages. Крім того, IDE також відповідає за координацію та забезпечення зв'язку між службами [9].

Підтримка мов програмування додається за допомогою спеціального VSPackage, який називається Language Service. Мовна служба визначає різні інтерфейси, які може реалізувати реалізація VSPackage, щоб додати підтримку різних функціональних можливостей [9].

Функції, які можна додати таким чином, включають забарвлення синтаксису, завершення оператора, відповідність дужок, підказки щодо інформації про параметри, списки членів та маркери помилок для складання фону [9]. Якщо інтерфейс буде реалізований, функціонал буде доступний для мови. Реалізації можуть повторно використовувати код з аналізатора або компілятора для мови.

3.2 Функціонування системи

На рисунку 3.1 зображено головну сторінку порталу для всіх користувачів, які не увійшли у систему. Тут можна побачити знімки сторінок порталу, на яких зображено головний функціонал системи і короткий опис про можливості. Також тут можна прочитати інформацію про створення програмного забезпечення чи залишити відгук про систему.

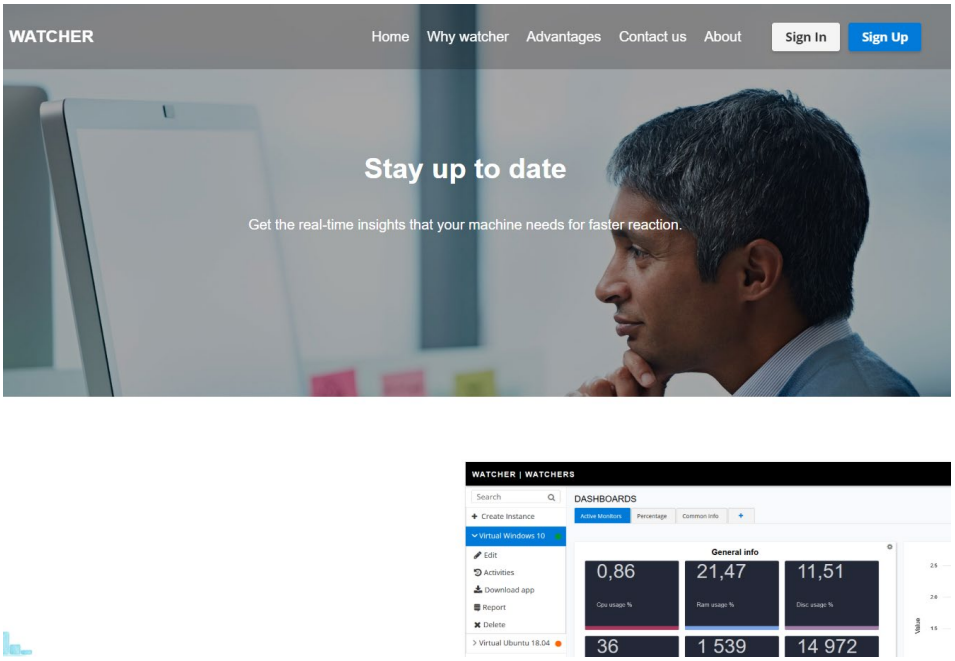

Рисунок 3.1 – Головна сторінка веб-порталу для незареєстрованих користувачів

Далі користувач може увійти в систему (Sign In) або ж зареєструватись (Sign Up) причому, якщо користувач із вже зареєстрованою поштою захоче зареєструватись ще раз то він просто увійде у систему із попередньо створеного аккаунта.

Для входження в систему потрібно скористуватись одним із трьох доступних методі, а саме за допомогою користувацького облікового запису із сервісу Google, GitHub або Facebook. Форма входу зображена на рисунку 3.2.

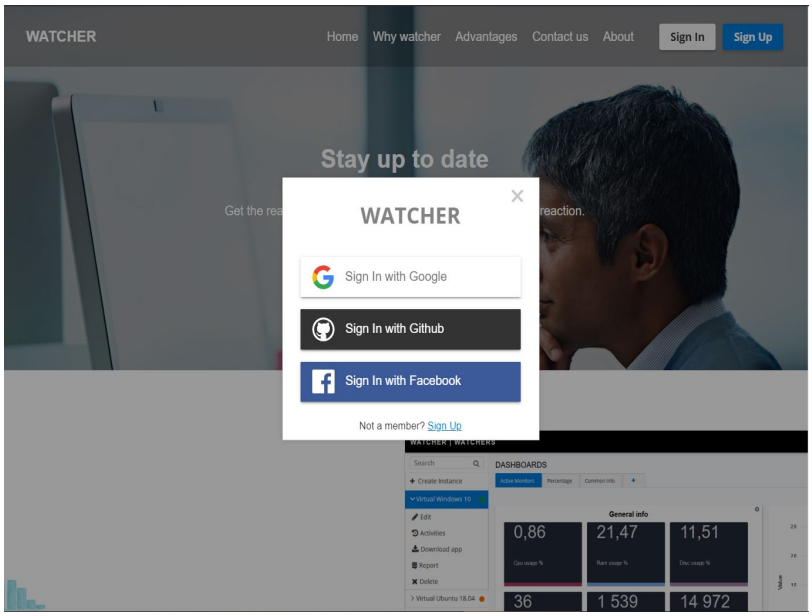

Рисунок 3.2 – Діалогове вікно із вибором сторонніх сервісів для входу у систему

Після успішного входу в один із сервісів портал реєструє користувача з інформацію наданою від стороннього сервісу, а саме це такі дані як ім’я, прізвище, електронна пошта, фото користувача. Також сайт покаже діалогове вікно в якому користувач може змінити дані, які були завантажені із стороннього сервісу, якщо раптом вони виявились неправильними, чи з якихось інших причин користувач захотів їх змінити і запропонує ввести назву організації за замовчування, як представлено на рисунку 3.3. Цей крок є необов’язковим, тому якщо користувач не хоче, що небудь змінювати він може пропустити це, натиснувши кнопку «Continue Later».

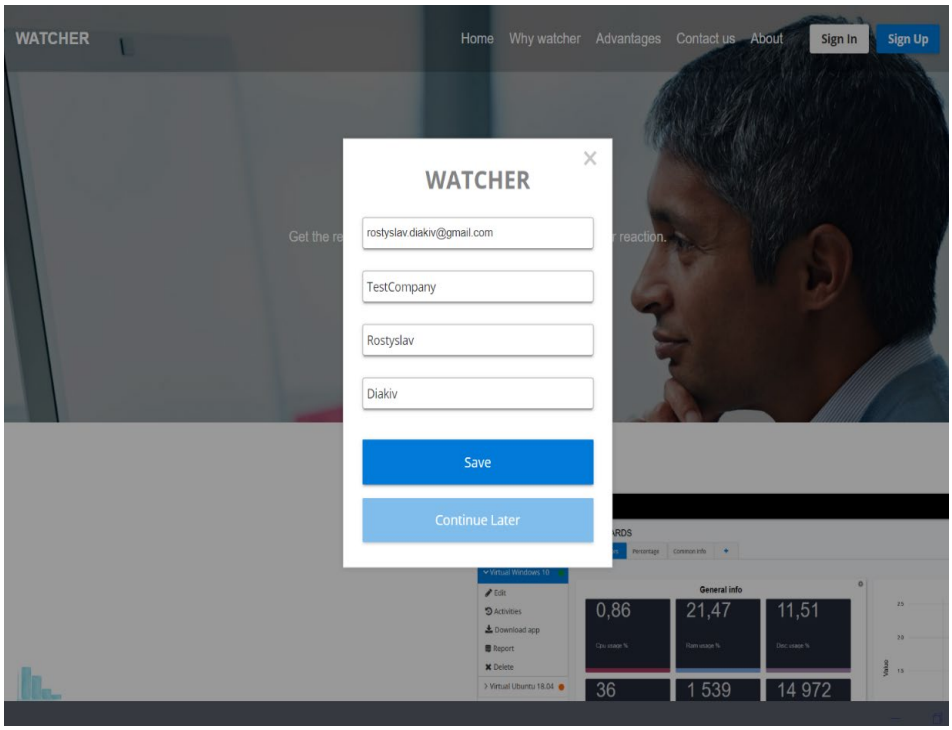

Рисунок 3.3 – Діалогове вікно із зміною автозаповнених даних користувача

Після успішного входу чи реєстрації користувач попадає на головну сторінку (рисунок 3.4). А саме це сторінка панелі приладів.

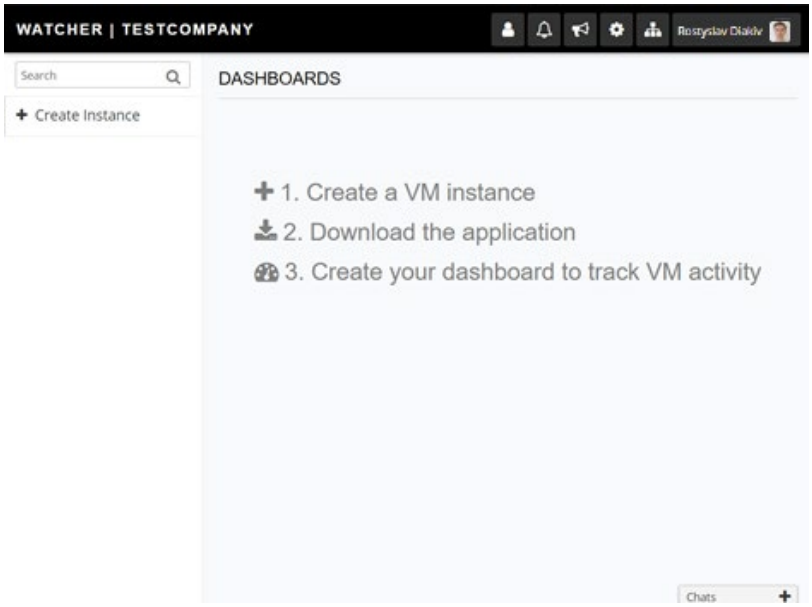

Рисунок 3.4 – Головна сторінка сайту для зареєстрованих користувачів

Для початку відслідковування стану комп'ютера користувач повинен створити екземпляр віртуального комп'ютера в системі. Для цього натискаємо на пункт меню «Create Instance» в лівому бічному меню. Сторінка створення нового екземпляра комп'ютера відображено на рисунку 3.5.

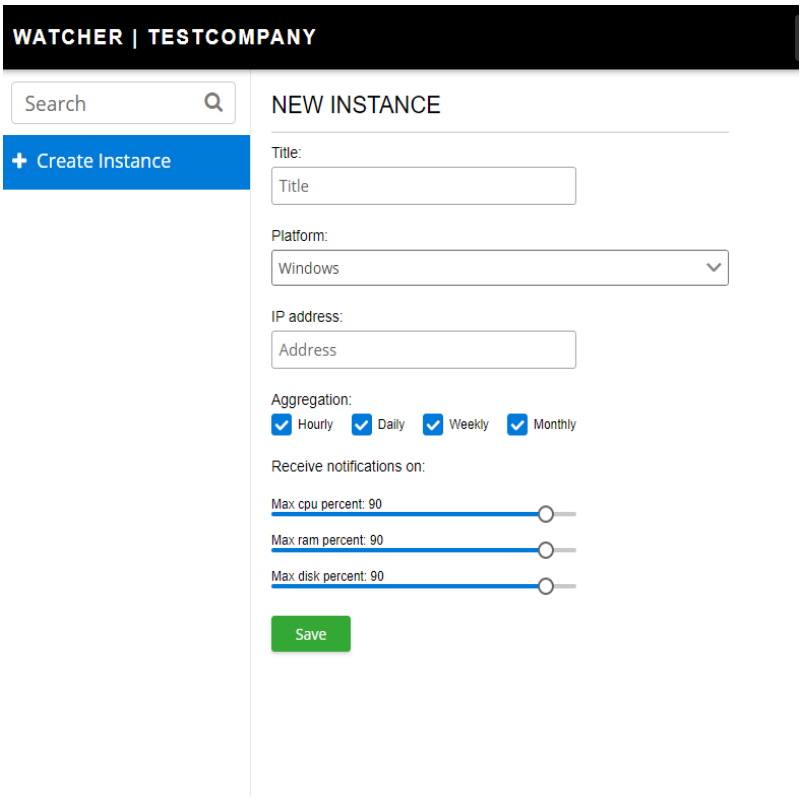

Рисунок 3.5 – Сторінка зі створенням нового екземпляра комп'ютера в системі

На сторінці налаштувань нового екземпляра комп’ютера можна задати йому назву, вибрати платформу на якій він працює Windows, Linux або MacOS. Також тут можна включити агрегацію даних за період часу як погодинно, щоденно, щотижнево і щомісячно. А також виставити максимальні значення показників системи у відсотках при перевищення яких буде повідомлено власника комп’ютера.

Після збереження екземпляра комп’ютера користувач попадає на сторінку (рисунок 3.6), де відкриваються такі функції як створення панелі приладів, яка є дошкою, де міститься набір різних графіків і таблиць. А також стають доступні такі функції по комп’ютеру як перегляд сповіщень активності комп’ютера у вкладці “Activities”, завантаження клієнта з посиланнями для різні платформ і вказаним ідентифікатором, який потрібно вказати при першому запуску програми збирання даних на комп’ютері (рисунок 3.7).

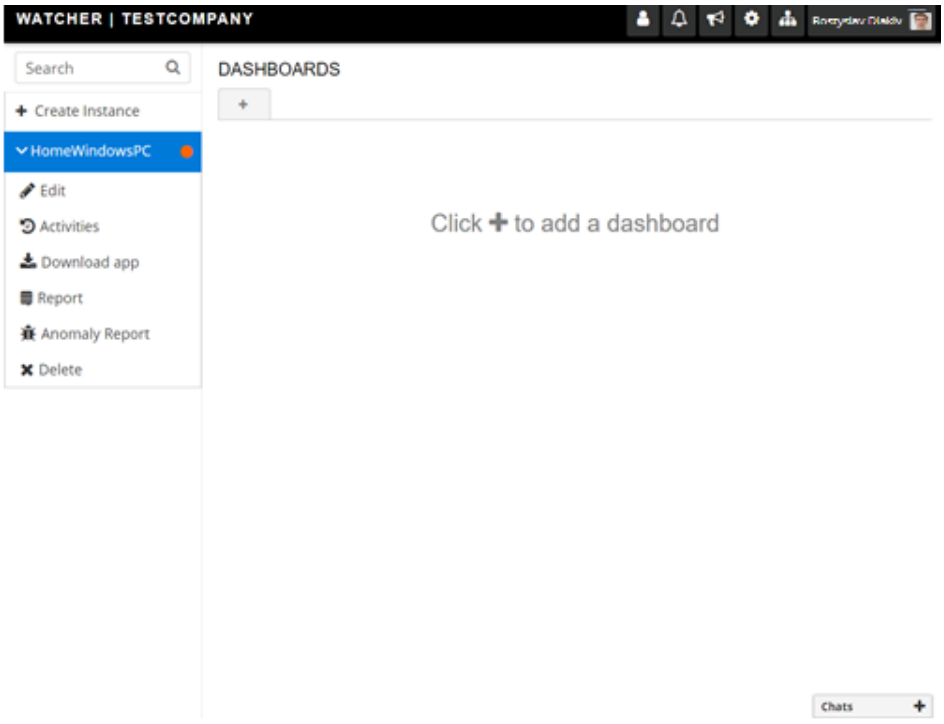

Рисунок 3.6 – Сторінка новоствореного екземпляра комп’ютера

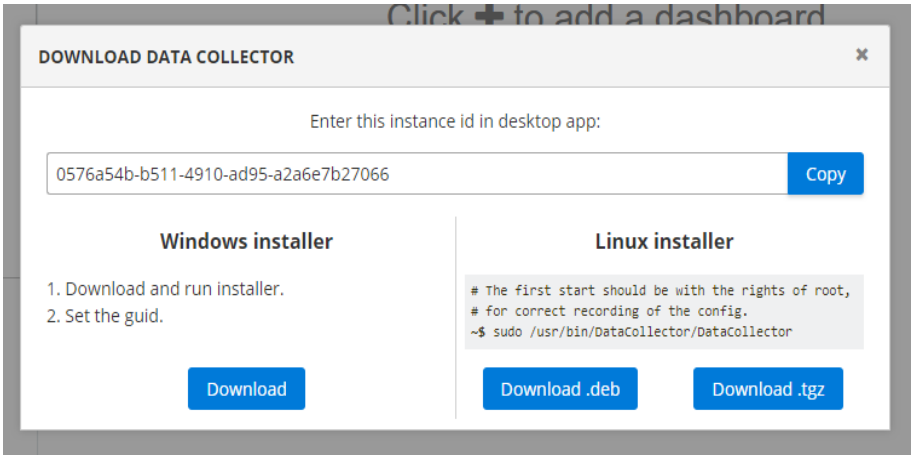

Рисунок 3.7 – Діалогове вікно з посиланнями на завантаження програми збирання даних

Після завантаження програми її потрібно встановити на комп'ютер, виконавши кроки вказані на рисунку 3.7. Запустивши програму після встановлення почнеться збір даних і відправка їх на віддалений сервер. Коротку інформацію про зібрані дані можна бачити у логах в консолі, що зображено на рисунку 3.8.

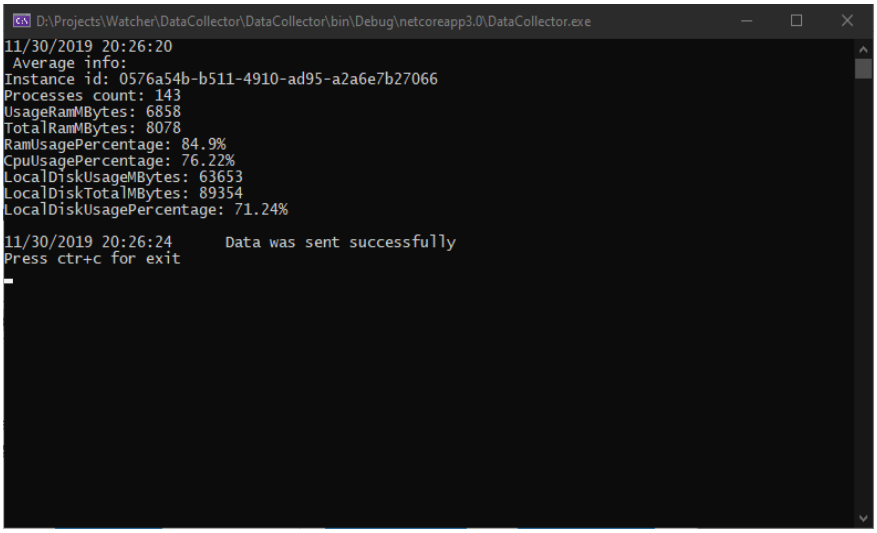

Рисунок 3.8 – Вікно програми збору даних

Далі користувачу скоріше за все цікаво побачити зібрану інформацію. Для цього потрібно створити панель приладів. Зробити це можна натиснувши на кнопку плюс біля напису «Dashboard», далі користувачу показано діалогове вікно із генерацією базової панелі приладів (рисунок 3.9). Його пізніше можна буде відредагувати, виділити старі або добавити нові графіки.

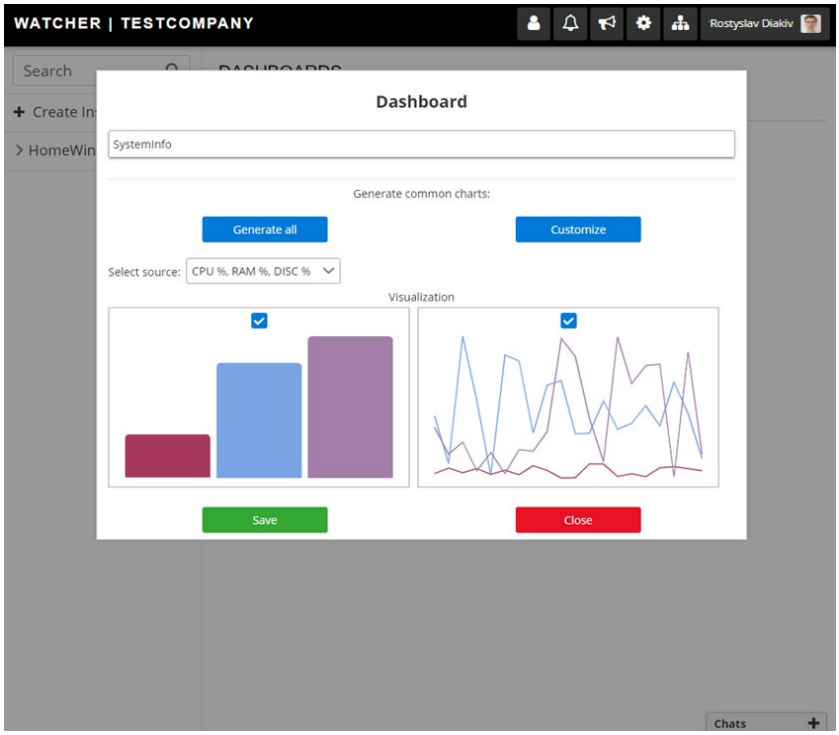

Рисунок 3.9 – Діалогове вікно створення нової панелі приладів

Після створення першої панелі приладів користувач отримує набір графіків (рисунок 3.10) на яких вже можна відслідкувати зібрані дані з комп'ютера, які оновлюються автоматично із заходженням їх від клієнта. За налаштуваннями по замовчуванні це відбувається кожних 10 секунд.

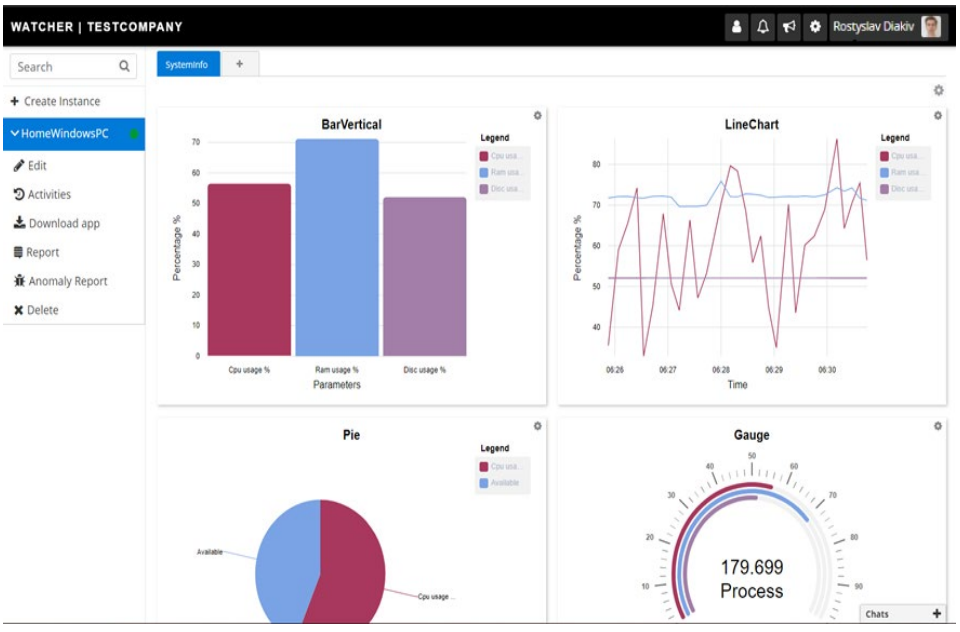

Рисунок 3.10 – Сторінка із створеною панеллю приладів

Панелі приладів можна редагувати, видаляти або створювати нові. Можна мати необмежену кількість панелей приладів. Це зручно коли потрібно згрупувати графіки, наприклад на одній панелі приладів можна вивести статистику по системі в цілому, а на іншому статистику по процесах. Також користувачу доступні налаштування самих графіків. Наприклад, можна поміняти часовий інтервал графіка «LineChart» з історією за 5 останніх хвилин до 60 хвилин (рисунок 3.11).

Також в налаштуваннях можна вибрати якого типу дані виводити.

Є два типи таких даних: загальні показники системи, що вибирається галочкою в пункті «Show total» або по набору процесів запущених в системі. При чому можна вибрати, яку кількість найбільш затратних ресурсів ми хочемо показувати.

Можна також міняти різні написи графіків, такі як заголовок графіка, заголовок історії, підпис горизонтальної чи вертикальної шкали, колірну схему позначень графіка. На рисунку 3.12 зображений доданий графік у вигляді інформаційних карток.

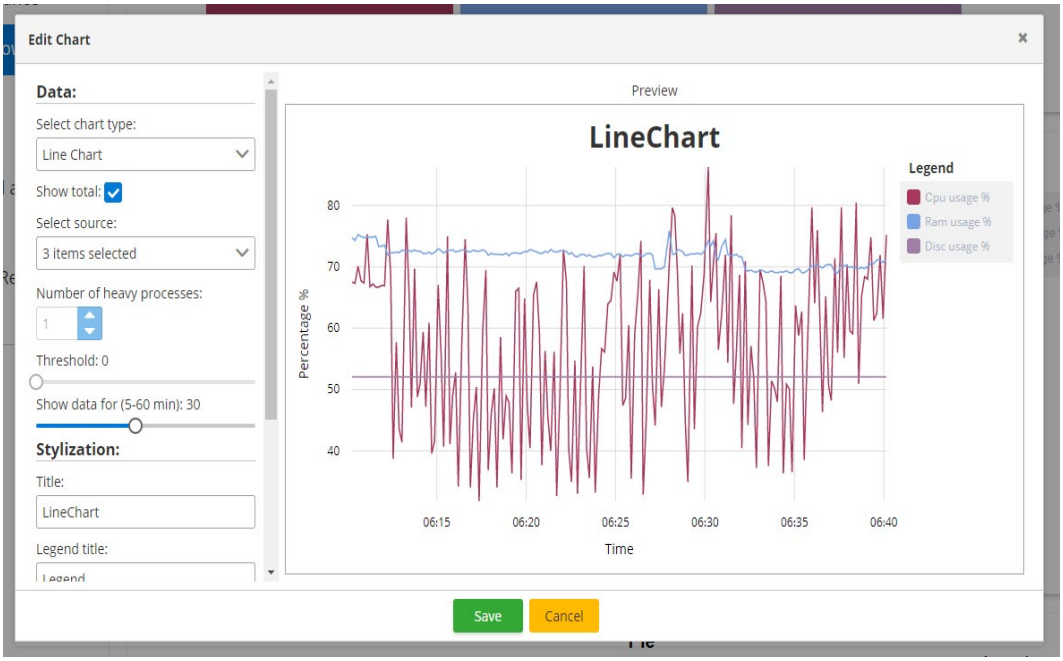

Рисунок 3.11 – Діалогове вікно з налаштуваннями графіка

Система була побудова з нахилом на гнучкість, щоб кожен користувач міг максимально ефективно використовувати простір веб-сайту. Тому було добавлено

розширений набір налаштувань графіків, сповіщень і інших все можливих значень.

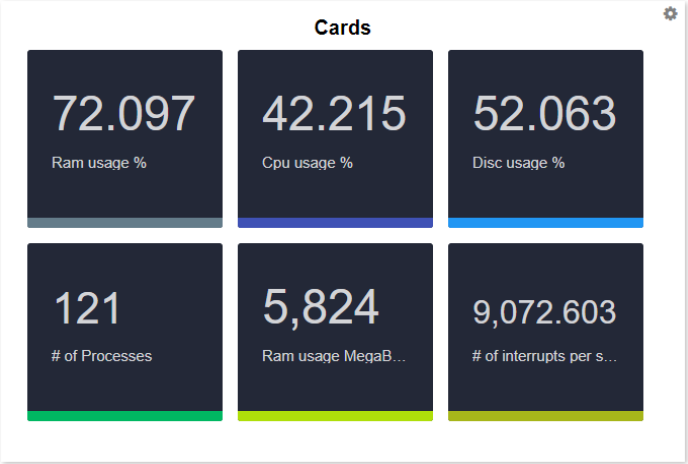

Рисунок 3.12 – Добавлений графік у вигляді інформаційних карток

Слід відзначити, що користувач має можливість використати 21 вид різних графіків (рисунок 3.13).

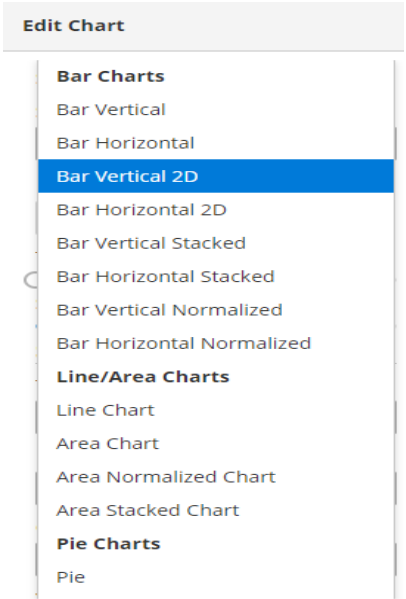

Рисунок 3.13 – Список доступних видів графіків

3.3 Дослідження роботи системи

Розроблене програмне забезпечення повинне повідомляти користувача, якщо показники системи перетнули критичні значення, це здійснюється двома шляхами.

Перший – це надсилання повідомлення на веб-сайт, а другим способом є дублювання сповіщення і на пошту користувача.

Для перевірки чи все працює як і заплановано були включені потрібні сповіщення на електронну пошту в налаштуваннях користувача (рисунок 3.14). А також виставлю мені критичні рівні показники для комп'ютера, а саме мінімальний рівень вільного ресурсу процесора у відсотках до 70 відсотків (рисунок 3.15).

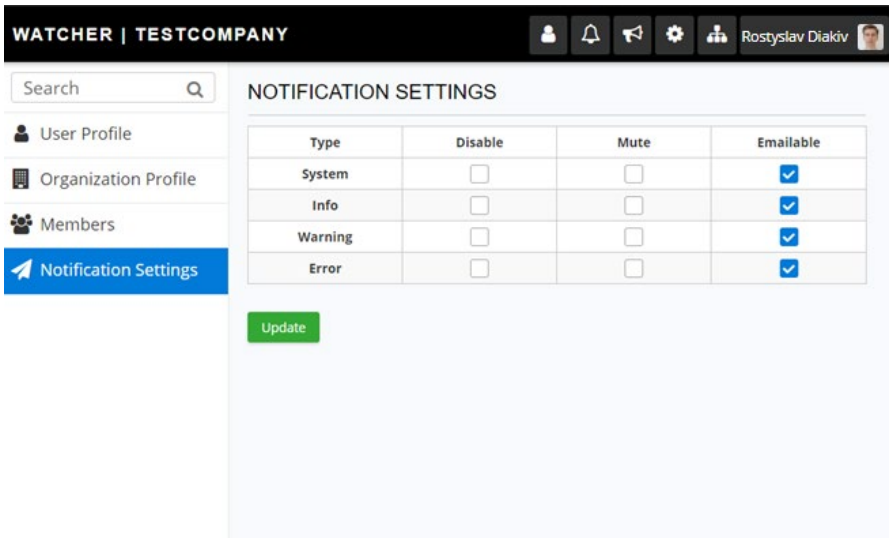

Рисунок 3.14 – Налаштування сповіщень користувача

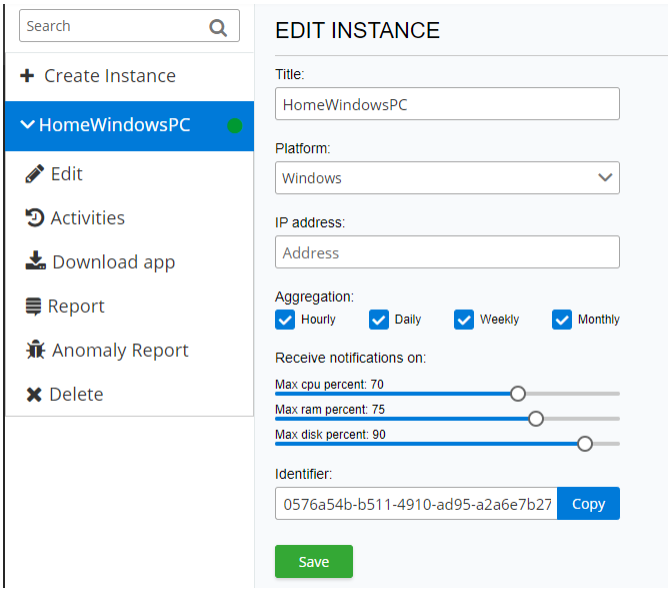

Рисунок 3.15 – Налаштування критичних рівнів показників комп'ютера

В результаті через деякий час, виконуючи повсякденні справи на комп'ютері в деякий момент система під навантаженням перетнула критичний показник. В

результаті чого можна побачити спеціальні сповіщення в заголовку сайту з описом проблеми (рисунок 3.16)

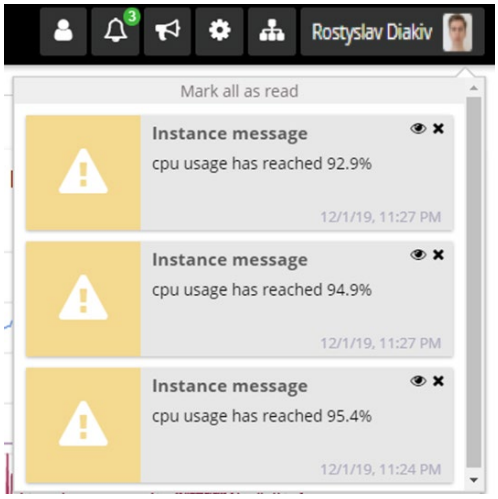

Рисунок 3.16 – Сповіщення на веб-сайті про порушення на комп'ютері, що відслідковується

Система повинна бути безвідмовною, тому потрібно провести тестування під навантаженням. Для цього треба запустити відслідковування на довгий період і зробити перевірку по зібрани

даних чи не було прогалин. По рисунку 3.17 можна побачити, що збір даних працює стабільно як мінімум останні 60 хвилин.

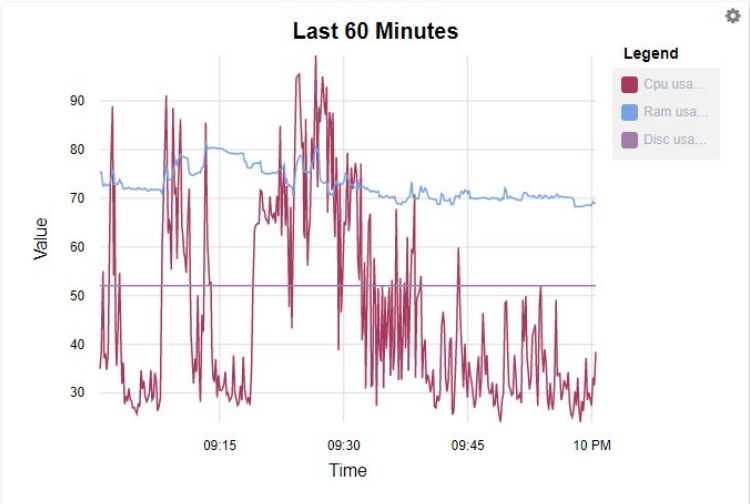

Рисунок 3.17 – Графік зібраних даних за годину часу

Можна вважати, що тест пройдено. Графік також цікавий тим, що на ньому видно деякі закономірності. Проаналізувавши його видно піки навантаження

системи і на основі них можна зробити висновки, що комп’ютер працював над досить затратними задачами і можливо його потужності недостатньо.

Висновки до розділу 3

1. Представлено платформу та інструменти реалізації.
2. Описано функціонування системи, яка реалізовує метод моніторингу системних ресурсів операційних системкомп’ютерів.
3. Проведено дослідження роботи системи.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було:

1. Зроблено огляд предметної області моніторингу системних ресурсів комп’ютера.
2. Проаналізовано відомі рішення системного моніторингу.
3. Сформовано завдання дослідження, в результаті якого буде розроблено метод моніторингу операційних систем.
4. Розроблено алгоритм методу моніторингу системних ресурсів операційних систем.
5. Розроблено структуру проектованої системи.
6. Визначено та описано сценарії використання моніторингу системних ресурсів операційних систем.
7. Розроблено інформаційне забезпечення системи моніторингу.
8. Представлено платформу та інструменти реалізації.
9. Описано функціонування системи, яка реалізовує метод моніторингу системних ресурсів операційних систем.
10. Проведено дослідження роботи системи.

Отже, запропонований метод реалізовано у вигляді системи. Розроблена система дозволяє слідкувати за станом комп’ютера шляхом графічної інтерпретації показників системи та отримання спеціальних сповіщень електронною поштою, генерування звіти на основі даних за певний проміжок часу, а також дозволяє адмініструвати веб-портал адміністратором.

Сисема буде корисна для підприємств різного розміру, але особливо тих у яких є багато комп’ютерів, стан яких фізично важко відслідкувати системному адміністратору, а також буде корисною для підприємств, фізичні сервери чи віртуальні машини яких повинні постійно бути у ввімкненому стабільному стані.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Кристофер Д. Маннинг. Введение в информационный поиск. Київ: «И.Д. Вильямс», 2011. 512 с.
2. Mark Masse. REST API Design. O'Reilly Media. 2011. 116 с.
3. Siraj ul Haq. Introduction to Monolithic Architecture and MicroServices Architecture. URL: https://medium.com/koderlabs/introduction-to-monolithic-architecture-and-microservices-architecture-b211a5955c63 (дата звернення: 11.04.2021).
4. Surabhi Pandey. Three tier architecture: the beginning. URL: https://medium.com/coffeetechandme/three-tier-architecture-the-beginning-2d2f6063fa1e (дата звернення: 15.04.2021).
5. ASP.NET Web API. URL: https://docs.microsoft.com/en-us/aspnet/web-api/ (дата звернення: 19.05.2021).
6. Sonny Recio. What you need to know about ASP.NET Core. URL: https://codeburst.io/what-you-need-to-know-about-asp-net-core-30fec1d33d78 (дата звернення: 10.07.2021).
7. Mátyás Lancelot Bors. An overview of Angular. URL: https://medium.com/@mlbors/an-overview-of-angular-3ccd2950648e (дата звернення: 13.07.2021).
8. MSDN. Tour of .NET. URL: https://docs.microsoft.com/en-us/dotnet/standard/tour (дата звернення: 11.04.2021).
9. Language Service Essentials. Wikipedia. URL: https://en.wikipedia.org/wiki/Microsoft_Visual_Studio (дата звернення: 04.04.2021).
10. Веб-сайт відкритої хмарної системи CloudStack. Документація хмарної системи, опис архітектури хмарної системи. URL: http://cloudstack.apache.org (дата звернення: 03.02.2021).
11. Волокита А.Н, Тимошенко А.И. Разделение рисков между клиентом и провайдером Cloud Computing. 2011. 209 с.
12. Инфосистемы Джет. ЦОД. Вычислительные комплексы и СХД. URL: http://www.jet.su/about/ (дата звернення: 11.08.2021).

13. Берсикас Д. Гаппагер Р. Сети передачи данных. Москва: Мир.1998. 728 с.

14. Weiming Shen. Distributed Manufacturing Scheduling Using Intelligent Agents. National Research Council Canada. 2002. 341 с.

15. VMWare ESX Server. Main page. URL: http://www.vmware.com/products/esx (дата звернення: 17.09.2021).

16. T. Dierks, C. Allen. RFC 2246. The TLS protocol version 1.0. 1999. 231 с.

17. Apache HTTP Server Project. Documentation. URL: http://httpd.apache.org/ (дата звернення: 07.11.2021).

18. Badger Lee, Grance Tim, Patt-Corner Robert, Voas Jeff. Cloud Computing Synopsis and Recommendations Special Publication. 2011. 870 с.

19. Cloud Security Alliance. URL: https://cloudsecurityalliance.org (дата звернення: 11.04.2021).

20. Дяків Р. І. Моніторинг комп'ютерних систем компанії. *Розвиток освіти, науки та бізнесу: результати 2021*: матеріали міжнародної науково-практичної інтернет-конференції, м. Дніпро, 6-7 грудня 2021 р. С. 80-81.

21. Дяків Р. І. Цифрова трансформація як актуальний напрямок розвитку підприємства. *Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення*: матеріали міжнародної наукової інтернет-конференції, м. Тернопіль, 10 грудня 2021 р. С. 22-24.

22. Комар М.П., Саченко А.О., Васильків Н.М. Методичні рекомендації до виконання кваліфікаційної роботи з освітньо-професійної програми «Комп'ютерні науки» спеціальності 122 «Комп'ютерні науки» за другим (магістерським) рівнем вищої освіти, м. Тернопіль: ЗУНУ, 2021 р. С. 32.

23. Вікіпедія. Docker. URL: https://uk.wikipedia.org/wiki/Docker (дата звернення: 11.08.2021).

24. Quality Assurance Group. Що таке Docker і навіщо він? URL: https://qagroup.com.ua/publications/shcho-take-docker-i-navishcho-vin/ (дата звернення: 11.10.2021).

25. Кент Бек. Екстримальне програмування. Москва: Захаров. 2002. 200 с.

26. Douglas Crockford. JavaScript: The Good Parts. O'Reilly. 2008. 345 с.

27. Eric Evans. Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley Professional. 2003. 560 p.

28. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, Grady Booch. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional. 1994. 416 p.

29. Eric Freeman, Elisabeth Robson. Head First Design Patterns: Building Extensible and Maintainable Object-Oriented Software. O'Reilly Media. 2020. 672 p.

30. Martin Fowler. Refactoring: Improving the Design of Existing Code. Addison-Wesley Professional. 2018. 448 p.

31. Mark J. Price. C# 10 and .NET 6 – Modern Cross-Platform Development. Packt Publishing. 2021. 824 p.

32. Jamie Chan. Learn C# in One Day and Learn It Well: C# for Beginners with Hands-on Project. CreateSpace Independent Publishing Platform. 2015. 160 p.

33. Lee Holmes. PowerShell Cookbook: Your Complete Guide to Scripting the Ubiquitous Object-Based Shell. O'Reilly Media. 2021. 1002 p.

34. Joseph Albahari. C# 9.0 in a Nutshell: The Definitive Reference. O'Reilly Media. 2021. 1060 p.

35. Brett D. McLaughlin, Gary Pollice, Dave West. Head First Object-Oriented Analysis and Design. O'Reilly Media. 2006. 636 p.

36. Chris Hanson, Gerald Jay Sussman. Software Design for Flexibility: How to Avoid Programming Yourself into a Corner. The MIT Press. 2021. 448 p.

37. Alistair Cockburn. Writing Effective Use Cases. Addison-Wesley Professional. 2000. 304 p.

38. Matt Weisfeld. Object-Oriented Thought Process. Addison-Wesley Professional. 2019. 240 p.

39. Stephen Cleary. Concurrency in C# Cookbook: Asynchronous, Parallel, and Multithreaded Programming. O'Reilly Media. 2019. 254 p.

40. Andrew Lock. ASP.NET Core in Action. Manning. 2021. 832 p.

41. Kent Beck. Implementation Patterns. Addison-Wesley Professional. 2007. 176 p.

42. Craig Larman. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. Pearson. 2004. 736 p.

Додаток А

Фрагменти програмного коду класів

```
using System;
using System.Collections.Concurrent;
using System.Collections.Generic;
using System.Diagnostics;
using System.Globalization;
using System.Linq;
using System.Threading.Tasks;
using System.Timers;

namespace DataCollector
{
   public class Collector
   {
      private static readonly Lazy<Collector> Value = new Lazy<Collector>(() => new Collector());
      private List<ProcessData> processData;
      private Collector() { }
      public static Collector Instance => Value.Value;
      public async Task<CollectedData> Collect()
      {
         CollectedData dataItem = null;
         try
         {
            var allProcesses = await GetProcesses();
            dataItem = new CollectedData
            {
               TotalRamMBytes = (float)Math.Round(GetTotalRam(), 2),
               RamUsagePercentage = (float)Math.Round(GetUsageRamPercentages(), 2),
               UsageRamMBytes = (float)Math.Round(GetUsageRam(), 2),
               CpuUsagePercentage = (float)Math.Round(GetUsageCpuPercentages(),2),
               LocalDiskTotalMBytes = (float)Math.Round(GetDiskTotalMbytes(), 2),
               LocalDiskUsageMBytes = (float)Math.Round(GetDiskTotalMbytes() - GetDiskFreeMbytes(), 2),
               LocalDiskUsagePercentage = (float)Math.Round(GetLocalDiskUsagePercent(), 2),
               Processes = allProcesses,
               ProcessesCount = allProcesses.Count,
               Time = DateTime.Now
            };
         }
         catch (Exception)
         {
            // ignored
         }
         return dataItem;
      }
      private float GetTotalRam()
      {
         string ramData = Bash("free -b | awk '{print $2 \";\" $3 \";\" $4}'");
         var ramSwap  = ramData.Split("\n");
         var ram = ramSwap[1].Split(";");
         return float.Parse(ram[0], System.Globalization.NumberStyles.Any, CultureInfo.InvariantCulture) / 1024.0f /
1024.0f;
      }

      private float GetUsageRam()
      {
         string ramData = Bash("free -b | awk '{print $2 \";\" $3 \";\" $4}'");
         var ramSwap = ramData.Split("\n");
         var ram = ramSwap[1].Split(";");
```

```
        return float.Parse(ram[1], System.Globalization.NumberStyles.Any, CultureInfo.InvariantCulture) / 1024.0f /
1024.0f;
    }

    private float GetUsageCpuPercentages()
    {
        string cpu = Bash(" cat <(grep 'cpu ' /proc/stat) <(sleep 1 && grep 'cpu ' /proc/stat) | awk -v RS=\"\" '{print ($13-
$2+$15-$4)*100/($13-$2+$15-$4+$16-$5)}'");

        return float.Parse(cpu, System.Globalization.NumberStyles.Any, CultureInfo.InvariantCulture);
    }

    private float GetUsageRamPercentages()
    {
        string ramData = Bash("free -b | awk '{print $2 \";\" $3 \";\" $4}'");
        var ramSwap  = ramData.Split("\n");
        var ram = ramSwap[1].Split(";");
        var freeRam = float.Parse(ram[1], System.Globalization.NumberStyles.Any, CultureInfo.InvariantCulture);
        var totalRam = float.Parse(ram[0], System.Globalization.NumberStyles.Any, CultureInfo.InvariantCulture);
        return freeRam/totalRam*100.0f;
    }

    private float GetDiskTotalMbytes()
    {
        string strData = Bash("df -t xfs -t ext4 | awk '{print $2 \";\" $3 \";\" $4}'");
        var allParts = strData.Split("\n");
        var disc = allParts[1].Split(";");
        return float.Parse(disc[0], System.Globalization.NumberStyles.Any, CultureInfo.InvariantCulture)/1024.0f;
    }

    private float GetDiskFreeMbytes()
    {
        string strData = Bash("df -t xfs -t ext4 | awk '{print $2 \";\" $3 \";\" $4}'");
        var allParts = strData.Split("\n");
        var disc = allParts[1].Split(";");
        return float.Parse(disc[2], System.Globalization.NumberStyles.Any, CultureInfo.InvariantCulture) / 1024.0f;
    }

    private float GetLocalDiskUsagePercent()
    {
        string strData = Bash("df -t xfs -t ext4 | awk '{print $2 \";\" $3 \";\" $4 \";\" $5}'");
        var allParts = strData.Split("\n");
        var disc = allParts[1].Split(";");
        return float.Parse(disc[3].Split("%")[0], System.Globalization.NumberStyles.Any, CultureInfo.InvariantCulture);
    }

    private async  Task<List<ProcessData>> GetProcesses()
    {
        await Task.Delay(100);
        string output = Bash("ps -xo rss,pmem,pcpu,comm | awk '{print $1 \";\" $2 \";\" $3  \";\" $4}'");
        processData = new List<ProcessData>();
        int counter = 0;

        foreach (string row in output.Split("\n"))
        {
            if (counter == 0)
            {
                counter++;
                continue;
            }

            var cols = row.Split(";");
            if(cols.Length == 4)
            {
                processData.Add( new ProcessData
```

```
                {
                    RamMBytes = float.Parse(cols[0], System.Globalization.NumberStyles.Any, CultureInfo.InvariantCulture) /
1024,
                    PRam = float.Parse(cols[1], System.Globalization.NumberStyles.Any, CultureInfo.InvariantCulture),
                    PCpu = float.Parse(cols[2], System.Globalization.NumberStyles.Any, CultureInfo.InvariantCulture),
                    Name = cols[3]
                });
            }
        }
        processData = GroupProcesses(processData);
        return processData;
    }

    private List<ProcessData> GroupProcesses(List<ProcessData> processes)
    {
        var temp = processes.GroupBy(proc => proc.Name).Select(
              group => new ProcessData
              {
                  Name = group.Key,
                  PCpu = (float)Math.Round(group.Sum(proc => proc.PCpu), 2),
                  PRam = (float)Math.Round(group.Sum(proc => proc.PRam), 2),
                  RamMBytes = (float)Math.Round(group.Sum(proc => proc.RamMBytes), 2)
              }).ToList();
        return temp;
    }

    private static string Bash(string cmd)
    {
        var escapedArgs = cmd.Replace("\"", "\\\"");

        var process = new Process()
        {
            StartInfo = new ProcessStartInfo
            {
                FileName = "/bin/bash",
                Arguments = $"-c \"{escapedArgs}\"",
                RedirectStandardOutput = true,
                UseShellExecute = false,
                CreateNoWindow = true,
            }
        };
        process.Start();
        string result = process.StandardOutput.ReadToEnd();
        process.WaitForExit();
        return result;
    }
  }
}

using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Management;
using System.Threading;
using System.Threading.Tasks;

namespace DataCollector
{
#if Windows
  public class Collector
  {
    private static readonly Lazy<Collector> Value = new Lazy<Collector>(() => new Collector());
    private readonly Dictionary<int, PerformanceCounter> _processCpuCounters;
    private readonly Dictionary<string, PerformanceCounter> _systemCounters;
```

```
private readonly float _totalRamMbyte;

private Collector()
{
    _processCpuCounters = new Dictionary<int, PerformanceCounter>();
    _systemCounters = new Dictionary<string, PerformanceCounter>
    {
        { "FreeRam", new PerformanceCounter("Memory", "Available MBytes") },
        { "Interrupts", new PerformanceCounter("Processor", "Interrupts/sec", "_Total") },
        { "DiskFreeMb", new PerformanceCounter("LogicalDisk", "Free Megabytes", "C:") },
        { "CPU", new PerformanceCounter("Processor Information", "% Processor Time", "_Total") },
        { "RAM", new PerformanceCounter("Memory", "% Committed Bytes In Use") },
        { "InterruptsTime", new PerformanceCounter("Processor", "Interrupts/sec", "_Total") },
        { "LocalDisk", new PerformanceCounter("LogicalDisk", "% Free Space", "C:") }
    };
    _systemCounters["CPU"].NextValue();
    _totalRamMbyte = GetTotalRAM();
    Thread.Sleep(1000);
}

public static Collector Instance => Value.Value;

public async Task<CollectedData> Collect()
{
    CollectedData dataItem = null;
    try
    {
        var allProcesses = await GetProcesses();
        var freeRam = _systemCounters["FreeRam"].NextValue();
        var freeDiskMb = _systemCounters["DiskFreeMb"].NextValue();
        var localDisk = _systemCounters["LocalDisk"].NextValue();
        var diskFreeMb = _systemCounters["DiskFreeMb"].NextValue();

        float GetDiskTotalMbytes() => (diskFreeMb / localDisk) * 100.0f;
        float GetDiskUsageMbytes() => GetDiskTotalMbytes() - diskFreeMb;

        dataItem = new CollectedData
        {
            InterruptsPerSeconds = (float)Math.Round(_systemCounters["Interrupts"].NextValue(), 2),
            InterruptsTimePercent = (float)Math.Round(_systemCounters["InterruptsTime"].NextValue(), 2),
            TotalRamMBytes = (float)Math.Round(_totalRamMbyte, 2),
            RamUsagePercentage = (float)Math.Round(100 - (freeRam / (_totalRamMbyte / 100)), 2),
            UsageRamMBytes = (float)Math.Round(_totalRamMbyte - freeRam, 2),
            CpuUsagePercentage = (float)Math.Round(_systemCounters["CPU"].NextValue(), 2),

            LocalDiskTotalMBytes = (float)Math.Round(GetDiskTotalMbytes()),
            LocalDiskUsageMBytes = (float)Math.Round(GetDiskUsageMbytes()),
            LocalDiskUsagePercentage = (float)Math.Round(100 - localDisk, 2),
            Processes = allProcesses,
            ProcessesCount = allProcesses.Count,
            Time = DateTime.Now
        };

    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }
    return dataItem;
}

private float GetTotalRAM()
{
    var totalRam = 1.0f;
```

```
        using (var ramMonitor = new ManagementObjectSearcher("SELECT
TotalVisibleMemorySize,FreePhysicalMemory FROM Win32_OperatingSystem"))
        {
            foreach (ManagementObject objram in ramMonitor.Get())
            {
                totalRam = Convert.ToUInt64(objram["TotalVisibleMemorySize"]) / 1024;    // Total RAM
            }
        }

        return totalRam;
    }

    private async Task<List<ProcessData>> GetProcesses()
    {
        var processes = Process.GetProcesses().Where(item => item.ProcessName != "Idle").ToArray();
        var result = new List<ProcessData>(processes.Length);
        var ListCPU = new Dictionary<int, float>(processes.Length);

        _processCpuCounters.Clear();

        foreach (var item in processes)
        {
            if (_processCpuCounters.ContainsKey(item.Id)) continue;

            var cpu = new PerformanceCounter("Process", "% Processor Time", item.ProcessName, true);
            cpu.NextValue();
            _processCpuCounters.Add(item.Id, cpu);
        }

        await Task.Delay(1000);

        foreach (var item in processes)
        {
            if (!_processCpuCounters.TryGetValue(item.Id, out var counter)) continue;
            ListCPU.Add(item.Id, (float)Math.Round(counter.NextValue() / Environment.ProcessorCount, 2));
            counter.Dispose();
        }

        foreach (var item in processes)
        {
            try
            {
                var name = item.ProcessName;
                item.Refresh();
                var ramMBytes = (item.PrivateMemorySize64 / 1024) / 1024;
                var pCpu = 0f;
                if (ListCPU.TryGetValue(item.Id, out var cpuP))
                {
                    pCpu = cpuP;
                }
                var pRam = (ramMBytes / _totalRamMbyte) * 100;
                if (_processCpuCounters.TryGetValue(item.Id, out var counter))
                {
                    counter.Dispose();
                }
                result.Add(new ProcessData
                {
                    Name = name,
                    RamMBytes = ramMBytes,
                    PCpu = pCpu,
                    PRam = pRam
                });
            }
            catch (Exception e)
```

```
            {
                Console.WriteLine(e.Message);
            }
        }
        result = GroupProcesses(result);

        return result;
    }

    private List<ProcessData> GroupProcesses(List<ProcessData> processes)
    {
        var temp = processes.GroupBy(proc => proc.Name).Select(
            group => new ProcessData
            {
                Name = group.Key,
                PCpu = (float)Math.Round(group.Sum(proc => proc.PCpu), 2),
                PRam = (float)Math.Round(group.Sum(proc => proc.PRam), 2),
                RamMBytes = (float)Math.Round(group.Sum(proc => proc.RamMBytes), 2)
            }).ToList();
        return temp;
    }
  }
#endif
}

public class Chart : Entity<int>, ISoftDeletable
  {
    public override int Id { get; set; }
    public bool ShowCommon { get; set; }
    public int Threshold { get; set; }
    public int MostLoaded { get; set; }
    public int HistoryTime { get; set; }
    public int DashboardId { get; set; }
    public Dashboard Dashboard { get; set; }
    public bool IsDeleted { get; set; }
    public string SchemeType { get; set; }
    public bool ShowLegend { get; set; }
    public string LegendTitle { get; set; }
    public bool Gradient { get; set; }
    public bool ShowXAxis { get; set; }
    public bool ShowYAxis { get; set; }
    public bool ShowXAxisLabel { get; set; }
    public bool ShowYAxisLabel { get; set; }
    public string YAxisLabel { get; set; }
    public string XAxisLabel { get; set; }
    public bool AutoScale { get; set; }
    public bool ShowGridLines { get; set; }
    public double RangeFillOpacity { get; set; }
    public bool RoundDomains { get; set; }
    public bool IsTooltipDisabled { get; set; }
    public bool IsShowSeriesOnHover { get; set; }
    public string Title { get; set; }
    public ChartType Type { get; set; }
    public string Sources { get; set; }
  }

public class Chat : Entity<int>, ISoftDeletable
  {
    public override int Id { get; set; }
    public string Name { get; set; }
    public ChatType Type { get; set; }
    public string CreatedById { get; set; }
    public User CreatedBy { get; set; }
    public int? OrganizationId { get; set; }
```

```
        public Organization Organization { get; set; }
        public IList<Message> Messages { get; set; }
        public IList<NotificationSetting> UsersSettings { get; set; }
        public IList<UserChat> UserChats { get; set; }
        public bool IsDeleted { get; set; }
    }

public class CollectorAppVersion : Entity<int>
    {
        public override int Id { get; set; }
        public DateTime CreatedAt { get; set; }
        public string Version { get; set; }
        public string ExeLink { get; set; }
        public string DebLink { get; set; }
        public string TgzLink { get; set; }
        public bool IsActive { get; set; }
    }

public class Dashboard : Entity<int>, ISoftDeletable
    {
        public override int Id { get; set; }
        public string Title { get; set; }
        public DateTime CreatedAt { get; set; }
        public int InstanceId { get; set; }
        public Instance Instance { get; set; }
        public IList<Chart> Charts { get; set; }
        public bool IsDeleted { get; set; }
    }

public class Feedback : Entity<int>, ISoftDeletable
    {
        public override int Id { get; set; }
        public string Text { get; set; }
        public ShortAnswerType WillUse { get; set; }
        public LongAnswerType Informatively { get; set; }
        public LongAnswerType Friendliness { get; set; }
        public LongAnswerType Quickness { get; set; }
        public string Name { get; set; }
        public string Email { get; set; }
        public DateTime CreatedAt { get; set; }
        public string UserId { get; set; }
        public User User { get; set; }
        public int? ResponseId { get; set; }
        public Response Response { get; set; }
        public bool IsDeleted { get; set; }
    }

public class Instance : Entity<int>, ISoftDeletable
    {
        public override int Id { get; set; }
        public string Title { get; set; }
        public string Address { get; set; }
        public string Platform { get; set; }
        public bool IsActive { get; set; }
        public Guid GuidId { get; set; }
        public bool AggregationForHour { get; set; }
        public bool AggregationForDay { get; set; }
        public bool AggregationForWeek { get; set; }
        public bool AggregationForMonth { get; set; }
        public float CpuMaxPercent { get; set; }
        public float RamMaxPercent { get; set; }
        public float DiskMaxPercent { get; set; }
        public DateTime StatusCheckedAt { get; set; }
        public int OrganizationId { get; set; }
        public Organization Organization { get; set; }
```

```
        public IList<Dashboard> Dashboards { get; set; }
        public bool IsDeleted { get; set; }
    }

public class Notification : Entity<int>, ISoftDeletable
    {
        public override int Id { get; set; }
        public string Text { get; set; }
        public DateTime CreatedAt { get; set; }
        public bool WasRead { get; set; }
        public string UserId { get; set; }
        public User User { get; set; }
        public int? InstanceId { get; set; }
        public Guid? InstanceGuidId { get; set; }
        public int NotificationSettingId { get; set; }
        public NotificationSetting NotificationSetting { get; set; }
        public bool IsDeleted { get; set; }
    }

public class Organization : Entity<int>, ISoftDeletable
    {
        public override int Id { get; set; }
        public string Name { get; set; }
        public string Description { get; set; }
        public string Email { get; set; }
        public string WebSite { get; set; }
        public string ContactNumber { get; set; }
        public bool IsActive { get; set; }
        public string ImageURL { get; set; }
        public string CreatedByUserId { get; set; }
        public User CreatedByUser { get; set; }
        public int? ThemeId { get; set; }
        public Theme Theme { get; set; }
        public Chat Chat { get; set; }
        public bool IsDeleted { get; set; }
        public IList<UserOrganization> UserOrganizations { get; set; }
        public IList<Instance> Instances { get; set; }
        public IList<OrganizationInvite> OrganizationInvites { get; set; }
    }

public class OrganizationInvite : Entity<int>, ISoftDeletable
    {
        public override int Id { get; set; }
        public string Link { get; set; }
        public int OrganizationId { get; set; }
        public Organization Organization { get; set; }
        public DateTime CreatedDate { get; set; }
        public DateTime ExperationDate { get; set; }
        public string InviteEmail { get; set; }
        public string InvitedUserId { get; set; }
        public User InvitedUser { get; set; }
        public string CreatedByUserId { get; set; }
        public User CreatedByUser { get; set; }
        public OrganizationInviteState State { get; set; }
        public bool IsDeleted { get; set; }
    }

public class Role : Entity<int>, ISoftDeletable
    {
        public override int Id { get; set; }
        public string Name { get; set; }
        public bool IsDeleted { get; set; }
    }
```

```
public class User : Entity<string>, ISoftDeletable
    {
        public override string Id { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string DisplayName { get; set; }
        public string Email { get; set; }
        public string EmailForNotifications { get; set; }
        public bool IsActive { get; set; }
        public DateTime CreatedAt { get; set; }
        public bool IsDeleted { get; set; }
        public string PhotoURL { get; set; }
        public string Bio { get; set; }
        public int RoleId { get; set; }
        public Role Role { get; set; }
        public int? LastPickedOrganizationId { get; set; }
        public Organization LastPickedOrganization { get; set; }
        public IList<NotificationSetting> NotificationSettings { get; set; }
        public IList<UserOrganization> UserOrganizations { get; set; }
        public IList<Notification> Notifications { get; set; }
        public IList<Feedback> Feedbacks { get; set; }
        public IList<Response> Responses { get; set; }
        public IList<Message> Messages { get; set; }
        public IList<UserChat> UserChats { get; set; }
        public IList<Chat> CreatedChats { get; set; }
        public IList<Organization> CreatedOrganizations { get; set; }
        public IList<OrganizationInvite> OrganizationInvites { get; set; }
    }

public class Theme : Entity<int>, ISoftDeletable
    {
        public override int Id { get; set; }
        public string Name { get; set; }
        public string BodyColor { get; set; }
        public string ThemePrimaryColor { get; set; }
        public string ThemeSecondaryColor { get; set; }
        public string ControlsHeight { get; set; }
        public string ButtonFontSize { get; set; }
        public bool IsDeleted { get; set; }
    }
```