Міністерство освіти і науки України

Луцький національний технічний університет
(повне найменування закладу вищої освіти)

Факультет комп'ютерних та інформаційних технологій
(повне найменування факультету)

Кафедра комп'ютерної інженерії та кібербезпеки
(повне найменування кафедри)

КВАЛІФІКАЦІЙНА РОБОТА ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

ЕЛЕКТРОННА СИСТЕМА ПОДАЧІ ПОКАЗНИКІВ ЛІЧИЛЬНИКА

ELECTRONIC SYSTEM FOR SUBMITTING METER READINGS

спеціальність 123 Комп'ютерна інженерія
(шифр і назва спеціальності)

освітня програма Комп'ютерна інженерія
(назва освітньої програми)

Виконав: здобувач вищої освіти
групи КІ-41
Чепіль Микола Анатолійович

(підпис)

Керівник:
к.т.н., доцент
Бортник Катерина Яківна

(підпис)

Кваліфікаційну роботу
допущено до захисту
« 08 » червня 2024 р.
Гарант освітньої програми:
к.т.н., доцент
Лавренчук Світлана Василівна

(підпис)

Луцьк – 2024 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *комп'ютерної інженерії та кібербезпеки*
Ступінь вищої освіти: *бакалавр*
Галузь знань: *12 Інформаційні технології*
Спеціальність: *123 Комп'ютерна інженерія*
Освітня програма: *«Комп'ютерна інженерія»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ проф. Н.Чернящук
« 10 » 01 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Чепілю Миколі Анатолійовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *Електронна система подачі показників лічильника*

Керівник роботи *к.т.н., доцент Бортник Катерина Яківна*

затверджені наказом закладу вищої освіти від «30» грудня 2023 року № 459/01-02
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи 11.06.2024р.

3. Вихідні дані до роботи *Джерелом розробки є науково-технічна література та публікації в періодичних виданнях з даного питання, опубліковані зарубіжні та вітчизняні роботи в даній області та різні інтернет-ресурси технічного спрямування*

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):
Аналіз сучасного стану проблеми, існуючих методів та засобів для подачі показів лічильника, розробка та обгрунтування електронної передачі показників, оцінка результатів дослідження

5. Перелік графічного (ілюстративного) матеріалу:

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз проблеми за темою роботи та постановка завдань дослідження	*Бортник К.Я., доцент*		
Теоретичне дослідження та практична реалізація	*Бортник К.Я., доцент*		
Практична реалізація додатку передачі показників	*Бортник К.Я., доцент*		
Нормоконтроль	*Багнюк Н.В., доцент*		
Гарант ОП	*Лавренчук С.В., доцент*		
Показник запозичень тексту	***___%***		
Академічна доброчесність	*Міскевич О.І.,асистент*		

7. Дата видачі завдання 10.01.2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	*Огляд літератури із досліджуваної проблеми*	до 15.02.2024 р.	Виконано
2.	*Провести аналіз технологій для оптимізації процесу передачі показань*	до 15.03.2024 р.	Виконано
3.	*Розробка та впровадження автоматизованої системи подачі показів*	до 04.05.2024 р.	Виконано
4.	*Висновки та пропозиції*	до 07.05.2025 р.	Виконано
5.	*Формування списку використаних джерел*	до 10.05.2024 р.	Виконано
6.	*Формування додатків*	до 15.05.2024 р.	Виконано
7.	*Оформлення ілюстративного матеріалу*	до 20.05.2024 р.	Виконано
8.	*Нормоконтроль*	до 01.06.2024 р.	Виконано
9.	*Інструментальна перевірка на академічний плагіат*	до 04.06.2024 р	Виконано
10.	*Представлення кваліфікаційної роботи бакалавра до захисту*	до 11.06.2024 р.	Виконано

Здобувач вищої освіти _____ Чепіль М.А.
(підпис) (прізвище, ініціали)

Керівник кваліфікаційної роботи _____ Бортник К.Я.
(підпис) (прізвище, ініціали)

АНОТАЦІЯ

Чепіль М.А. Електронна система подачі показників лічильника.

Кваліфікаційна робота бакалавра ОП «Комп'ютерна інженерія» спеціальності 123 Комп'ютерна інженерія. Луцький національний технічний університет. Луцьк, 2024.

Кваліфікаційна робота складається з вступу, трьох розділів, висновків, списку використаних джерел, трьох додатків.

У першому розділі висвітлюється проблематика, яку вирішує предмет розробки, проведено аналіз схожих пристроїв, також сформовані цілі для реалізації.

У другому розділі здійснено аналіз та вибір засобів розробки. Обґрунтований вибір засобів розробки: M5Stack TimerCam, ASP.NET Web API, .NET Blazor та бібліотеку Chart.js.

Третій розділ присвячено опису розроблених програм для автоматизації зйомки та передачі показників лічильника, обробки даних за допомогою бібліотеки Tesseract та візуалізації зібраних даних у вигляді графіків.

Об'єкт – автоматизація передачі показань лічильників.

Предмет – алгоритми та технології, що використовуються для створення системи автоматизації збору, обробки та візуалізації даних з лічильників.

Метою роботи є створення програмного проекту, який допоможе автоматизувати процес збору, обробки та візуалізації даних з лічильників, що дасть змогу інтерактивно працювати з ними.

Ключові слова: автоматизація, лічильник, M5Stack TimerCam, ASP.NET Web API, .NET Blazor, Chart.js, Tesseract.

ANNOTATION

Chepil M.A. Electronic System for Submitting Meter Readings. Bachelor's thesis in the educational program «Computer Engineering», specialty 123 Computer Engineering. Lutsk National Technical University. Lutsk, 2024.

The qualification work consists of an introduction, three chapters, conclusions, a list of used sources, and three appendices. The first chapter addresses the issues the development subject aims to solve, includes an analysis of similar devices, and formulates the goals for implementation. The second chapter involves the analysis and selection of development tools. The chosen tools are justified: M5Stack TimerCam, ASP.NET Web API, .NET Blazor, and the Chart.js library. The third chapter is dedicated to the description of the developed programs for automating the capture and transmission of meter readings, data processing using the Tesseract library, and visualization of the collected data in the form of charts.

The object of the study is the automation of meter readings transmission. The subject of the study includes the algorithms and technologies used to create a system for automating the collection, processing, and visualization of meter data. The aim of the work is to create a software project that helps automate the process of collecting, processing, and visualizing meter data, allowing for interactive work with them.

Keywords: automation, meter, M5Stack TimerCam, ASP.NET Web API, .NET Blazor, Chart.js, Tesseract.

ЗМІСТ

ВСТУП

Актуальність теми зумовлена трендом сучасності, щодня передові компанії світу випускають інноваційні продукти, які допомагають спростити життя своїм клієнтам. Ці пристрої об’єднуються у складні системи, для виконання своїх завдань, що дозволяє користувачу не витрачати час та сили на виконання буденних процесів.

Прикладом такого явища є поява «розумних» лічильників, які використовуються для передачі показань споживання у відповідні комунальні служби. Проблема таких девайсів полягає в тому, що використання нових технологій потребує заміни старого обладнання на нове, з втручанням комунальних підприємств, що не завжди є швидким та доступним варіантом для багатьох людей.

Вирішенням проблеми з заміною старого обладнання може бути додатковий пристрій, що вміє взаємодіяти з лічильниками без розумних функцій. Девайс може кріпитися на лічильник, та в потрібний момент робити фото показників, які в подальшому можна обробити і передати у відповідні реєстри.

Метою роботи є створення мінімально робочого прототипу пристрою-накладки і менеджмент-сервера, що взаємодіють між собою, та здатні обробляти зібрані з лічильника дані для подальшої передачі на власноруч створений застосунок для аналізу даних.

Об’єкт дослідження – автоматизація передачі показань лічильників.

Предмет дослідження – система, що складається з пристрою-накладки, сервера обробки вхідних даних та застосунку для їх аналізу.

Завдання, які необхідно виконати:

– проаналізувати проблематику питання;

– сформувати вимоги до предмету розробки;

– визначити інструменти, що найкраще підходять для виконання поставлених задач;

– реалізувати спосіб захоплення даних з лічильника;

– встановити з’єднання між пристроєм та сервером;

– провести збір даних з довільного лічильника протягом певного проміжку часу;

– розробити застосунок, що відобразить зібрані дані у вигляді графіка;

– оцінити переваги готового рішення;

– запропонувати можливі покращення системи.

РОЗДІЛ 1
АНАЛІЗ ПРОБЛЕМАТИКИ ТА ФОРМУЛЮВАННЯ ЗАВДАНЬ

1.1 Актуальність автоматизації поданя показань лічильника

У цьому підрозділі розглядається проблематика, пов’язана з мануальним збором даних з лічильників, який застосовується в багатьох системах комунальних послуг, включаючи водопостачання, газопостачання та електроенергію. Цей традиційний метод характеризується значними недоліками, які можуть призводити до серйозних фінансових та оперативних проблем.

1.1.1 Трудомісткість процесу

Мануальний збір даних вимагає значної кількості людських ресурсів. Співробітники мають регулярно відвідувати кожен об’єкт, де встановлено лічильник, для фізичного зчитування показників. Цей процес є особливо трудомістким у великих містах із розгалуженою інфраструктурою або в регіонах із великою кількістю віддалених або важкодоступних об’єктів. Вартість таких операцій включає не тільки заробітну плату персоналу, але й витрати на транспорт, планування маршрутів та час, витрачений на виконання цих завдань.

1.1.2 Ризики людської помилки

Мануальний збір даних характеризується великим впливом людського фактору, що вносить потенційні ризики помилок на кожному етапі: від зчитування до введення даних в систему обліку. Помилки можуть виникати через неправильне прочитання лічильників, невірний запис даних або їх некоректне введення в комп’ютерні системи. Такі помилки можуть призвести до неточностей у виставленні рахунків для споживачів, що, в свою чергу, веде до скарг, переглядів платежів і збільшення адміністративного навантаження на служби обслуговування клієнтів.

1.1.3 Затримки у передачі даних

При мануальному зборі даних існує значна затримка між моментом зчитування показників і їх доступністю для обробки та аналізу. Це ускладнює оперативне реагування на зміни у споживанні, ускладнює швидке виявлення та усунення витоків або інших проблем із інфраструктурою. Затримки можуть також впливати на здатність комунальних компаній до планування та прогнозування, що є критично важливим для ефективного управління ресурсами, особливо під час пікових навантажень або в кризових ситуаціях.

1.1.4 Вплив на планування та використання ресурсів

Неточності, викликані мануальним збором даних, можуть істотно вплинути на планування та використання ресурсів. Неправильні дані можуть призвести до надлишкового виробництва або, навпаки, до дефіциту ресурсів, порушуючи баланс попиту та пропозиції та збільшуючи витрати на виробництво та дистрибуцію. В довгостроковій перспективі, це може також вплинути на стратегічне планування інвестицій в інфраструктуру та розвиток нових технологій, що забезпечують стійкість та ефективність системи.

Цей аналіз підкреслює необхідність пошуку альтернативних рішень для оптимізації процесу збору даних, що мінімізують вплив людського фактору і забезпечують більшу точність, оперативність та ефективність у використанні ресурсів.

1.2 Компаративний аналіз існуючих систем автоматизації збору даних

У цьому підпункті проводиться детальний порівняльний огляд сучасних технологічних рішень, які застосовуються для автоматизації збору даних з лічильників. Важливим аспектом аналізу є оцінка «розумних» лічильників і систем дистанційного зчитування, їхніх ключових характеристик, переваг, обмежень, а також можливостей інтеграції з існуючими інфраструктурами.

Розгляд цих систем допоможе виявити найбільш ефективні методи для вдосконалення процесів збору даних і їх впровадження у практику.

1.2.1 Розумні лічильники

Смарт-лічильники, інтелектуальні лічильники або ж «смарти» виготовлені з електронних компонентів та не мають механічних рухомих частин [1]. Ці пристрої вбудовуються безпосередньо в інфраструктуру комунальних послуг і автоматично збирають дані про споживання енергії, газу, води або тепла (рис. 1.1).

Рисунок 1.1 – Приклад розумного лічильника природного газу [2]

Розумні лічильники забезпечують передачу даних у реальному часі через інтернет або інші мережеві технології, такі як Wi-Fi, GSM, або навіть новітні технології, такі як LoRaWAN або NB-IoT.

Основними перевагами розумних лічильників є здатність до миттєвого моніторингу та звітування, що дозволяє споживачам та постачальникам послуг швидко реагувати на зміни у споживанні, оптимізувати ресурси та зменшити втрати. Також ці системи дозволяють проводити дистанційне управління споживанням, автоматичне виявлення несправностей та витоків. Автоматизація зчитування виключає потребу в ручному введенні, зменшуючи ризик помилок у показниках лічильника та спрощуючи процес для користувачів [3].

Недоліками є те, що розумні лічильники вимагають значних капіталовкладень для інсталяції та підтримки. Крім того, вони можуть породжувати питання конфіденційності та безпеки даних, оскільки великі обсяги споживацької інформації зберігаються та передаються через мережі.

1.2.2 Системи дистанційного зчитування

Системи дистанційного зчитування забезпечують інший підхід до автоматизації збору даних. Вони часто використовуються для модернізації існуючих лічильників за допомогою встановлення на них спеціалізованих модулів, що дозволяють віддалено збирати дані без необхідності заміни всієї лічильної системи.

Перевагою даної системи є економічність, порівняно з повною заміною лічильників на розумні. Вони забезпечують здатність до інтеграції з багатьма типами існуючих систем та можуть значно покращити точність даних і швидкість їх передачі. Цей пристрій долає недоліки, такі як низька точність ручного зчитування лічильників, трудомісткість, трудомісткість, відсутність централізованого та ефективного управління та вирішує виробничі проблеми [4].

Недоліками є те, що системи дистанційного зчитування можуть вимагати додаткових витрат на обслуговування та підтримку. Крім того, як і розумні лічильники, вони висувають вимоги до безпеки даних та можуть бути обмежені у функціональності в порівнянні з повністю інтегрованими «розумними» системами.

1.2.3 Інтегровані системи управління

Ці системи представляють собою комплексний підхід до управління даними, інтегруючи не тільки лічильники, але й інші сенсори, системи моніторингу та управління на базі передових технологій. Вони дозволяють створити цілісну картину споживання, аналізувати дані на макрорівні і приймати обґрунтовані управлінські рішення.

Переваги інтегрованих систем. Дані системи можуть пропонувати глибокий аналітичний аналіз і широкі можливості для оптимізації використання

ресурсів. Вони також сприяють підвищенню загальної ефективності систем через автоматизацію управління та моніторингу.

Найбільшим обмеженням інтегрованих систем є їхня складність та висока вартість впровадження та підтримки. Крім того, вони вимагають високого рівня технічної підготовки персоналу і можуть бути занадто складними для деяких застосувань.

Цей аналіз дозволяє глибше зрозуміти переваги та можливі виклики, пов’язані з кожною з цих систем, і вибрати найбільш підходящий варіант для конкретних умов експлуатації.

1.3 Формулювання завдання для оптимізації процесу збору даних

Цей розділ визначає обсяг завдань, спрямованих на створення комплексного рішення, яке включає розробку допоміжного пристрою для збору даних та вдосконалену серверну архітектуру для їх обробки. Задача полягає не лише в покращенні механізмів збору даних, але й у створенні високоефективної системи обробки даних.

1.3.1 Вимоги до розробки допоміжного пристрою

Пристрій повинен бути спроєктований з використанням матеріалів та компонентів, які забезпечують його стійкість до зовнішніх умов, включаючи несприятливі погодні умови та механічні впливи. Це критично важливо для забезпечення безперебійної роботи в різних середовищах, від промислових заводів до міських інфраструктур. Важливо також врахувати оптимізацію енергоспоживання та використання енергоефективних технологій, що є життєво необхідною для забезпечення довгострокової та ефективної роботи пристрою. Пристрій повинен мати здатність до автономної роботи від батарейок, що дозволяє використовувати його в різних умовах, включаючи місця без постійного доступу до електромережі. Надійність і довговічність є ключовими факторами, що гарантують стабільну роботу пристрою протягом тривалого часу без потреби в частому технічному обслуговуванні.

1.3.2 Фінальний продукт

Основною метою проекту є створення надійного, ефективного та енергонезалежного рішення для збору даних, що буде інтегроване з вдосконаленою серверною інфраструктурою, здатною ефективно обробляти зібрані дані. Це включає поліпшення оперативності обробки, підвищення точності даних та оптимізацію управління ресурсами. Завдяки цьому, проект спрямований на зниження витрат на обслуговування, збільшення оперативної ефективності та покращення якості обслуговування кінцевих користувачів.

РОЗДІЛ 2
ОБҐРУНТУВАННЯ ВИБРАНИХ ТЕХНОЛОГІЙ ТА ЗАСОБІВ

2.1 Вибір апаратної складової

M5Stack TimerCam – це компактний і функціональний пристрій, що використовує керуюче ядро ESP32-WROOM-32E та 2-мегапіксельний датчик зображення (OV2640), який ідеально підходить для створення електронної системи подачі показників лічильника [5]. Цей програмований пристрій оснащений камерою та модулем для бездротової передачі даних (рис 2.1).

Рисунок 2.1 – M5Stack TimerCam [5]

Його вибір обумовлений низкою переваг, серед яких простота інтеграції, висока енергоефективність та гнучкість налаштувань. M5Stack TimerCam підтримує програмування на мовах C++ і Python, що робить його універсальним і зручним для розробників. Основні характеристики цього пристрою включають високоякісну камеру з можливістю зйомки та зберігання зображень, підтримку Wi-Fi для бездротової передачі даних, енергоефективні

компоненти з можливістю роботи від батареї, а також підтримку різних мов програмування, зокрема C++. Важливо відзначити, що M5Stack TimerCam підходить для експлуатації в несприятливих умовах завдяки своїй надійній конструкції та здатності автономно працювати від батарейок. Ці характеристики роблять M5Stack TimerCam оптимальним вибором для пропонованої системи, забезпечуючи надійність і функціональність пристрою при мінімальних витратах на енергію.

2.2 Вибір програмних компонентів

Готове рішення для електронної системи подачі показників лічильника складається з двох основних компонентів: серверної частини та клієнтської частини. Серверна частина розроблена з використанням ASP.NET Web API, що забезпечує надійну і масштабовану платформу для створення веб-сервісів та API. Клієнтська частина представляє собою прошивку мікроконтролера, написану на мові програмування C++. Додатковим до готового рішення є застосунок розроблений з використанням технології .NET Blazor, що призначений для графічного відображення зібраних даних.

2.2.1 Серверна частина

ASP.NET Web API – це платформа, розроблена компанією Microsoft для створення веб-додатків та служб на основі HTTP, що дозволяє легко створювати і керувати веб-сервісами. Ця технологія дозволяє створювати RESTful сервіси, які можуть обслуговувати запити від клієнтів з будь-якого пристрою, підключеного до мережі. Основні переваги ASP.NET Web API включають легкість інтеграції, високу продуктивність, безпеку та гнучкість, що дозволяє легко створювати служби, які охоплюють широкий спектр клієнтів, включаючи браузери та мобільні пристрої [6]. Легкість інтеграції забезпечується можливістю без проблем інтегруватися з іншими системами і додатками, що забезпечує гнучкість і можливість розширення функціональності системи. Висока продуктивність досягається завдяки оптимізації платформи

для роботи з великими обсягами даних та високою швидкістю обробки запитів, що є критично важливим для систем, які працюють у режимі реального часу. Гнучкість платформи дозволяє розширювати і налаштовувати її під специфічні потреби проекту, що забезпечує можливість адаптації до змінних вимог. Використання ASP.NET Web API дозволяє забезпечити надійність і масштабованість серверної частини системи, що є важливим для обробки та зберігання великих обсягів даних, які надходять від численних клієнтських пристроїв.

Для реалізації функцій розпізнавання тексту з зображень у нашій системі було обрано бібліотеку Tesseract. Tesseract є однією з найпопулярніших і найпотужніших бібліотек для оптичного розпізнавання символів (OCR), яка розроблялася під керівництвом Hewlett-Packard і наразі підтримується Google [7]. Вибір цієї бібліотеки обумовлений її високою точністю розпізнавання, підтримкою великої кількості мов та відкритим вихідним кодом, що дозволяє вільно використовувати й модифікувати її під специфічні потреби проекту. Основні переваги бібліотеки Tesseract включають високу точність розпізнавання, підтримку багатьох мов, можливість додаткового навчання, інтеграцію з .NET і відкритий вихідний код [8]. Tesseract показує високі результати точності розпізнавання тексту завдяки використанню сучасних алгоритмів машинного навчання. Це дозволяє з високою точністю розпізнавати текст з різноманітних зображень, навіть якщо вони мають певні шуми чи недоліки. Бібліотека підтримує більше 100 мов, що робить її універсальною для використання в міжнародних проектах. Tesseract надає можливість додаткового навчання на користувацьких наборах даних, що дозволяє поліпшити точність розпізнавання для специфічних шрифтів чи мов, які можуть бути недостатньо представлені у стандартному наборі. Бібліотека Tesseract легко інтегрується з .NET додатками за допомогою доступних обгорток, таких як Tesseract OCR for .NET. Це дозволяє використовувати можливості OCR безпосередньо у додатках, написаних на C#, що значно спрощує розробку та впровадження функцій розпізнавання тексту. Оскільки Tesseract є проектом з відкритим

вихідним кодом, він може бути вільно використаний і модифікований відповідно до потреб проекту. Це дозволяє налаштувати бібліотеку для досягнення найкращих результатів у конкретних умовах використання. Зазвичай OCR – це процес, який зазвичай не дає хороших результатів. Проте Tesseract, хоч і не є абсолютно безпомилковим, є найкращою бібліотекою, що може бути використана для перетворення зображення на текст [9].

У пропонованій системі бібліотека Tesseract використовується для розпізнавання тексту з зображень, отриманих з M5Stack TimerCam. Це дозволяє автоматично витягувати показники лічильників з фотографій, що значно полегшує процес збору даних. Завдяки використанню Tesseract забезпечується висока точність розпізнавання тексту, швидка обробка зображень та надійна робота системи в цілому. Інтеграція Tesseract у проект дозволяє автоматизувати процес збору даних з лічильників, мінімізуючи людський фактор і підвищуючи точність обліку спожитої енергії. Це забезпечує додаткову зручність для користувачів, а також підвищує ефективність та надійність всієї системи. Таким чином, використання бібліотеки Tesseract у поєднанні з іншими технологіями, такими як Blazor і Chart.js, дозволяє створити комплексну, надійну і ефективну систему подачі показників лічильника, яка відповідає сучасним вимогам до якості та продуктивності.

2.2.2 Клієнтська частина

Клієнтська частина системи представляє собою прошивку мікроконтролера, написану на мові програмування C++. Вибір C++ для розробки прошивки обумовлений кількома ключовими факторами. Перш за все, висока продуктивність: C++ забезпечує ефективне використання апаратних ресурсів, що є критично важливим для мікроконтролерів з обмеженими обчислювальними потужностями і пам’яттю. По-друге, контроль над апаратним забезпеченням: C++ дозволяє розробникам мати прямий доступ до апаратних компонентів, що дозволяє оптимізувати роботу пристрою і максимально використовувати його можливості. По-третє, велика кількість бібліотек: мова C++ має велику кількість бібліотек, які підтримують роботу з

різними апаратними модулями і сенсорами, адже є однією з найпопулярніших мов програмування для розробки прошивок, що спрощує розробку і дозволяє швидко додавати нові функції до прошивки [10]. Нарешті, портативність коду: проекти на C++ можна легко переносити між різними платформами, що забезпечує гнучкість у виборі апаратного забезпечення і спрощує процес масштабування системи. Прошивка на C++ забезпечує стабільну та швидку роботу клієнтської частини системи, дозволяючи ефективно збирати та передавати дані з лічильників до серверної частини. Завдяки використанню C++ досягається висока продуктивність та надійність, що є важливими критеріями для системи, яка повинна працювати в режимі реального часу і обробляти великі обсяги даних.

2.2.3 Застосунок з графічним інтерфейсом

Для розробки графічного застосунку було обрано технологію Blazor. Blazor є сучасним фреймворком від компанії Microsoft, який дозволяє створювати інтерактивні веб-додатки з використанням мови програмування C#. Використання Blazor надає можливість об’єднання клієнтської і серверної частин програми в єдиній мовній екосистемі, що значно спрощує процес розробки та підтримки коду. Основною перевагою Blazor є можливість написання клієнтської логіки без використання JavaScript, замість цього використовується C# і .NET, що дозволяє зосередити всі зусилля на одній мові програмування.

Blazor використовує технологію WebAssembly, що дозволяє запускати код, написаний на C#, безпосередньо в браузері. Це забезпечує високу продуктивність і можливість створення складних інтерфейсів користувача. Використання Blazor також дозволяє інтегрувати сучасні можливості .NET, такі як обробка даних, доступ до баз даних, впровадження різних шаблонів проектування, що робить розробку більш ефективною та зручною.

Для відображення графіків у графічному застосунку була використана бібліотека Chart.js, яка є однією з найпопулярніших та найпростіших у використанні бібліотек для створення інтерактивних графіків на основі

JavaScript. Chart.js дозволяє легко створювати різноманітні типи графіків, включаючи лінійні, стовпчасті, кругові, радарні та інші типи, що дозволяє візуалізувати дані в зручній і наглядній формі.

Chart.js забезпечує високу гнучкість у налаштуванні графіків, що дозволяє розробникам адаптувати їх під специфічні вимоги проекту. Бібліотека підтримує анімацію, реакцію на події, різноманітні опції налаштування стилів і кольорів, що робить її потужним інструментом для візуалізації даних у веб-додатках. Інтеграція Chart.js з Blazor дозволяє створювати складні графічні інтерфейси, де логіка побудови і відображення графіків керується з боку C#, тоді як Chart.js відповідає за рендеринг графіків у браузері.

Таким чином, вибір технології Blazor для розробки графічного застосунку та використання бібліотеки Chart.js для відображення графіків дозволяє об’єднати переваги сучасної платформи .NET з можливостями популярної бібліотеки для візуалізації даних. Це рішення забезпечує високу продуктивність, зручність розробки та підтримки, а також дозволяє створювати інтуїтивно зрозумілі та привабливі інтерфейси користувача.

2.3 Середовище розробки

Роль середовища розробки є неоціненною в процесі створення будь-якого продукту, оскільки воно забезпечує інструменти для написання, тестування та налагодження коду. Вибір правильного середовища розробки є критичним для ефективної роботи як серверної, так і клієнтської частини системи.

2.3.1 Серверна частина та графічний застосунок

Visual Studio 2022 від Microsoft є інтегрованим середовищем розробки (IDE), яке пропонує широкий набір інструментів для створення веб-додатків, зокрема ASP.NET Web API та Blazor. Visual Studio 2022 має інтуїтивно зрозумілий інтерфейс, який полегшує процес розробки. Він включає потужні засоби для написання коду, автодоповнення, інтелектуальне підсвічування синтаксису та інші функції, які значно спрощують роботу розробника (рис. 2.2).

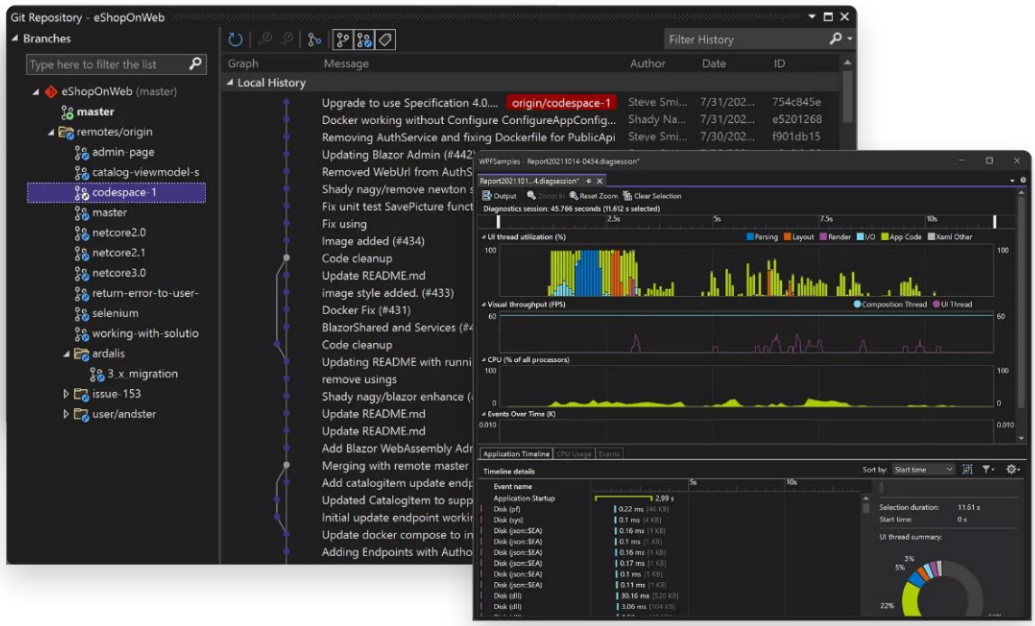

Рисунок 2.2 – Інтерфейс Visual Studio 2022 [11]

Інструменти для налагодження, що надає Visual Studio 2022, забезпечують вбудовані засоби для налагодження і тестування додатків [11]. Це дозволяє розробникам швидко виявляти і виправляти помилки, що є критично важливим для забезпечення стабільності та надійності системи. Вона підтримує розробку на багатьох мовах програмування, включаючи C#, JavaScript, Python та інші, що дозволяє розробникам використовувати одну IDE для різних проектів і технологій. Інтеграція з іншими продуктами Microsoft є ще однією значущою перевагою. Visual Studio 2022 легко інтегрується з іншими інструментами Microsoft, такими як Azure, SQL Server, і т.д. Це забезпечує безшовну роботу з різними компонентами екосистеми Microsoft, що є важливим для великих проектів з комплексною архітектурою. Вибір Visual Studio 2022 для розробки серверної частини системи на основі ASP.NET Web API та графічного додатку на основі Blazor забезпечує високий рівень продуктивності, зручність розробки і надійність коду, що є важливими критеріями для успішної реалізації проекту.

2.3.2 Клієнтська частина

Arduino IDE є популярним середовищем розробки, спеціально створеним для написання і завантаження коду на мікроконтролери. Основні переваги використання Arduino IDE включають зручність, широку підтримку, велику спільноту та портативність [12]. Arduino IDE має простий та інтуїтивно зрозумілий інтерфейс, який робить процес написання коду та його налагодження легким (рис. 2.3).

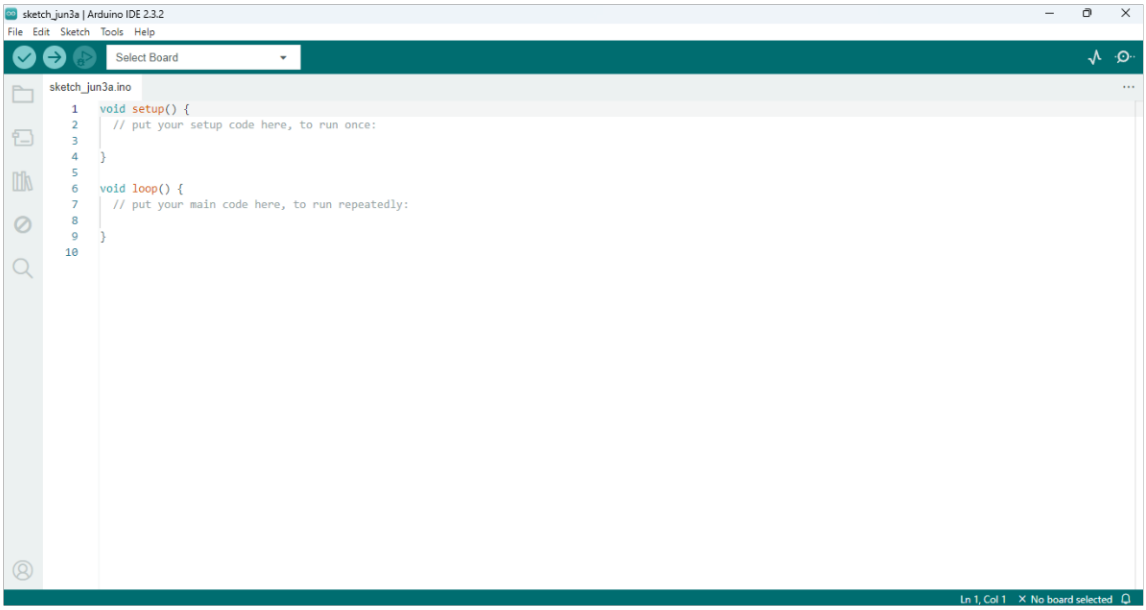

Рисунок 2.3 – Інтерфейс Arduino IDE

Це середовище розробки дозволяє швидко почати роботу з мікроконтролерами без необхідності тривалого навчання. Крім того, Arduino IDE підтримує велику кількість мікроконтролерів та плат, що робить його універсальним інструментом для розробників. Існує безліч бібліотек, які дозволяють легко додавати нові функціональні можливості до проекту. Велика і активна спільнота користувачів, яка постійно ділиться знаннями, прикладами коду і бібліотеками, значно полегшує процес розробки, оскільки розробники можуть знаходити рішення на свої питання в спільноті. Проекти, створені в Arduino IDE, легко переносити між різними платформами і середовищами, що забезпечує гнучкість у виборі апаратного забезпечення і спрощує процес

масштабування системи. Використання Arduino IDE для розробки клієнтської частини системи дозволяє швидко і ефективно створювати прошивки для мікроконтролерів, забезпечуючи при цьому високу продуктивність та надійність коду. Це середовище розробки є оптимальним вибором для проектів, які вимагають гнучкості і швидкої адаптації до нових вимог.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ТА РЕЗУЛЬТАТИ РОБОТИ СИСТЕМИ

3.1 Опис загальної архітектури

Архітектура системи електронної подачі показників лічильника складається з трьох основних компонентів: серверної частини, клієнтської частини та графічного інтерфейсу. Кожен з цих компонентів виконує певну роль у зборі, обробці та візуалізації даних. Серверна частина розроблена з використанням платформи ASP.NET Web API. Вона відповідає за прийняття, обробку та збереження даних, отриманих від клієнтських пристроїв. За допомогою бібліотеки Tesseract здійснюється розпізнавання тексту з зображень, що надходять з лічильників. Оброблені дані зберігаються в базі даних для подальшого аналізу та візуалізації, а також надаються через API для графічного інтерфейсу.

Клієнтська частина представлена програмним забезпеченням, яке виконується на пристрої M5Stack TimerCam. Цей пристрій здійснює зйомку показників лічильника і передає зображення на сервер. Пристрій регулярно робить знімки лічильника за допомогою вбудованої камери, передає зображення та інші необхідні дані на сервер через Wi-Fi і працює автономно від батарейок, що дозволяє використовувати його в різних умовах, включаючи несприятливі.

Графічний інтерфейс розроблений з використанням технології Blazor та бібліотеки Chart.js. Він призначений для візуалізації даних, зібраних та оброблених сервером. Комунікація між компонентами системи здійснюється за допомогою RESTful API. Клієнтські пристрої передають дані на сервер, де вони обробляються та зберігаються в базі даних. Графічний інтерфейс отримує дані з сервера для візуалізації.

Така архітектура забезпечує масштабованість, надійність та гнучкість системи. Таким чином, загальна архітектура системи електронної подачі показників лічильника поєднує в собі надійні програмні та апаратні

компоненти, що забезпечують ефективний збір, обробку та візуалізацію даних для кінцевих користувачів.

3.2 Реалізація серверної частини

Серверна частина системи електронної подачі показників лічильника побудована з використанням технології ASP.NET Web API та складається з кількох взаємопов’язаних проєктів. Проєкт Diploma.Entities містить сутності моделей даних для всієї системи. Diploma.Server.Api реалізує основну функціональність серверної частини, забезпечуючи прийом, обробку та видачу даних через RESTful API, саме через цей програмний інтерфейс пристрій-клієнт надсилає зчитані дані на сервер, а графічний інтерфейс отримує їх для відображення [13]. Diploma.Server.Business містить бізнес-логіку системи, включаючи розпізнавання тексту з зображень за допомогою бібліотеки Tesseract. Diploma.Server.DataAccess забезпечує доступ до бази даних з використанням Dapper і ADO.NET, що дозволяє ефективно виконувати SQL-запити [14]. Diploma.Server.DataBase містить налаштування для бази даних та скрипти для створення її структури. Сервер обробляє зображення, розпізнає текст за допомогою Tesseract і зберігає дані в базі даних. Такий підхід дозволяє ефективно збирати, обробляти та візуалізувати дані, забезпечуючи надійність і зручність використання системи.

3.2.1 Diploma.Server.Api

Інтерфейс взаємодії з сервером реалізований у вигладі API, що є одним з ключових компонентів системи, що дозволяє обробляти HTTP-запити на збереження та отримання даних. Технологія ASP.NET Web API дозволяє реалізовувати контролери, що вміщають методи для взаємодії. Для досягнення поставлених цілей було створено два контроллера: ImageController, StatisticController.

ImageController містить метод UploadImage (лістинг 3.1), що приймає в тілі запиту зображення для подальшої обробки, код для ознайомлення можна

переглянути в додатку Б. StatisticController створений для запитів зі сторони графічного інтерфейсу і при звернені на його метод GetAllStatistic повертає усі зібрані дані.

Лістинг 3.1 – Реалізація методу завантаження зображення

```
[HttpPost("UploadImage")]
[Consumes("image/jpeg")]
public async Task<IActionResult> UploadImage()
{
    try
    {
        using var memoryStream = new MemoryStream();
        await Request.Body.CopyToAsync(memoryStream);
        var bitmap = new Bitmap(memoryStream);

        string text = _imageService.GetTextFromImage(bitmap);

        var res = await _statisticService.CreateStatisticRecord(
            new CounterSnapshot
            {
                Id = Guid.NewGuid(),
                CurrentValue = double.Parse(text),
                CreatedAt = DateTime.UtcNow,
            });

        return res ? NoContent() : StatusCode(418);
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, ex.Message);
        return StatusCode(500);
    }
}
```

Кінець лістингу 3.1

3.2.2 Diploma.Entities

Основним призначенням цього проекту є виокремлення в окремий розділ сутностей, що використовуються в рішенні. Для створення робочого рішення було створено одну модель даних CounterSnapshot (лістинг 3.2), що об’єднує необхідні властивості для обробки показань лічильника. CounterSnapshot включає в себе Id типу Guid, що є унікальним ідентифікатором запису, CurrentValue типу double значення якого рівне показникам лічильника на момент зчитування даних, DifferenceWithPreviousValue типу double, що рівне

різниці актуальних показань лічильника та показань при останньому зчитуванні, CreatedAt типу DateTime і вказує на час проведеного зчитування.

Лістинг 3.2 – Модель даних CounterSnapshot

```
public class CounterSnapshot
{
    public Guid Id { get; set; }
    public double CurrentValue { get; set; }
    public double DifferenceWithPreviousValue { get; set; }
    public DateTime CreatedAt { get; set; }
}
```

Кінець лістингу 3.2

3.2.3 Diploma.Server.Business

Проект з бізнес-логікою відповідає за обробку даних. Він містить в собі два сервіси, перший – ImageService, що використовує бібліотеку для розпізнавання тексту – Tesseract (лістинг 3.3) і після проведених процедур відправляє дані на подальшу обробку, другий – StatisticService який виконує роль посередника при збереженні та отриманні оброблених даних.

Лістинг 3.3 – Метод розпізнавання тексту

```
public string GetTextFromImage(Bitmap image)
{
    string executableLocation =
Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location);
    string path = Path.Combine(executableLocation, "tessdata");
    if (!Directory.Exists(path))
        return null;

    using var engine = new TesseractEngine(path, "eng");
    string text = engine.Process(image).GetText();
    return text;
}
```

Кінець лістингу 3.3

3.2.4 Diploma.Server.DataAccess

Основним завданям проекту є взаємодія з базою даних з використанням Mini ORM Dapper, в проекті основним класом є StatisticDataController (лістинг 3.4), який передає отримані дані на збереження до бази даних та може отримувати їх для повернення при запиті на витягування з неї.

Лістинг 3.4 – Методи взаємодії з базою даних

```
public Task<bool> CreateStatisticRecord(CounterSnapshot counterSnapshot)
{
    if (counterSnapshot == null)
        throw new ArgumentNullException(nameof(counterSnapshot));

    var parameters = new DynamicParameters();
    parameters.Add("@Id", counterSnapshot.Id);
    parameters.Add("@CurrentValue", counterSnapshot.CurrentValue);
    parameters.Add("@CreatedAt", counterSnapshot.CreatedAt);
    return PerformNonQuery(parameters, "Create");
}

public Task<IEnumerable<CounterSnapshot>> GetAllStatistic()
{
    return GetManyAsync<CounterSnapshot>("GetAll");
}
```

Кінець лістингу 3.4

3.2.5 Diploma.Server.DataBase

Проект містить налаштування та структуру бази даних Sql Server, для рішення була розроблена наступна структура даних, дві процедури: sp_Statistic_Create та sp_Statistic_GetAll також створена одна таблиця – Statistic (лістинг 3.5), що призначені для створення, отримання та збереження статистичних даних [15]. За цією структурою в подальшому буде створена база даних.

Лістинг 3.5 – Структура таблиці, що зберігає зібрані дані

```
CREATE TABLE [dbo].[Statistic]
(
      [Id] UNIQUEIDENTIFIER NOT NULL PRIMARY KEY,
    [CurrentValue] DECIMAL(32, 16) NOT NULL,
    [DifferenceWithPreviousValue] DECIMAL(32, 16) NOT NULL,
    [CreatedAt] DATETIME NOT NULL
)
```

Кінець лістингу 3.5

3.3 Реалізація клієнтської частини

Клієнтська частина системи електронної подачі показників лічильника представлена програмним забезпеченням, яке виконується на пристрої M5Stack TimerCam. M5Stack TimerCam є компактним та функціональним пристроєм, який оснащений високоякісною камерою та модулем для бездротової передачі даних, що робить його ідеальним вибором для даної задачі.

Основні функції клієнтської частини включають зйомку зображень, передачу даних та автономну роботу. Зйомка зображень здійснюється регулярно за допомогою вбудованої камери пристрою, що дозволяє отримувати високоякісні знімки показників лічильника. Ці знімки є основним джерелом даних для системи, оскільки вони містять інформацію про поточний стан лічильника.

Передача даних є наступним критичним етапом роботи клієнтської частини. Зображення, разом з іншими необхідними даними, передаються на сервер через Wi-Fi. Це забезпечує бездротову комунікацію між клієнтським пристроєм та сервером, дозволяючи швидко і ефективно передавати великі обсяги даних. Сервер, у свою чергу, приймає ці дані для подальшої обробки та збереження.

Автономна робота пристрою M5Stack TimerCam є ще однією важливою функцією клієнтської частини. Пристрій працює автономно від батарейок, що

дозволяє використовувати його у різних умовах, включаючи несприятливі. Це означає, що пристрій може бути розміщений у віддалених або важкодоступних місцях без необхідності постійного підключення до електромережі. Автономність також забезпечує тривалу роботу пристрою без необхідності частого обслуговування, що є важливою перевагою для системи, яка повинна функціонувати надійно і безперервно.

Таким чином, клієнтська частина, реалізована на базі M5Stack TimerCam, забезпечує ефективний збір, передачу та обробку зображень показників лічильника, працюючи автономно в різних умовах. Це робить її критично важливою складовою системи, що сприяє загальній надійності та ефективності роботи всієї системи електронної подачі показників лічильника.

Прошивка пристрою написана мовою програмування C++ та включає в себе кілька основних компонентів: налаштування підключення до мережі, деталі підключення до сервера, Інтервал зйомки та відправки зображень на сервер, конфігурація камери, зйомку та відправку зображення на сервер, ознайомитись з кодом прошивки можна переглянувши додаток А.

3.4 Реалізація графічного застосунку

Реалізація графічного застосунку є важливою складовою системи електронної подачі показників лічильника, оскільки вона забезпечує зручний інтерфейс для користувачів, дозволяючи їм взаємодіяти з даними та аналізувати їх. Графічний застосунок розроблений з використанням технології Blazor та інтеграції з бібліотекою Chart.js, що дозволяє ефективно відображати результати у вигляді графіків [16-17].

Реалізація Blazor застосунку починається з створення основного каркасу веб-додатку. Blazor, як сучасний фреймворк від компанії Microsoft, дозволяє розробляти інтерактивні веб-додатки з використанням мови програмування C#. Це значно спрощує розробку, оскільки усуває необхідність використання JavaScript для реалізації клієнтської логіки. В Blazor застосунку створюються

компоненти, які відповідають за різні частини інтерфейсу, забезпечуючи їх взаємодію з сервером через API. Це дозволяє зручно керувати станом додатку та оновлювати інтерфейс у режимі реального часу.

Інтеграція з Chart.js (лістинг 3.6) є ключовим етапом у реалізації графічного застосунку, оскільки ця бібліотека забезпечує потужні засоби для створення інтерактивних графіків. Chart.js є однією з найпопулярніших та найпростіших у використанні бібліотек для візуалізації даних у веб-додатках. Інтеграція здійснюється шляхом додавання Chart.js до проєкту Blazor та створення компонентів, які використовують цю бібліотеку для рендерингу графіків. Це дозволяє відображати дані у вигляді лінійних, стовпчастих, кругових та інших типів графіків, що робить візуалізацію даних більш наочною та зручною для користувачів. З підготовкою даних для відображення можна ознайомитись в додатку В.

Лістинг 3.6 – Створення компонента Chart.js

```
@page "/"

<PageTitle>Home</PageTitle>

@if (groupedData.Any())
{
    @foreach (var day in groupedData)
    {
        <div>
            <h3>@day.Key</h3>
            <canvas id="chart-@day.Key" width="400" height="200"></c
        </div>
    }
}
else
{
    <p>Loading...</p>
}
```

Кінець лістингу 3.6

3.5 Результати роботи системи

Пристрій M5Stack TimerCam було встановлено на лічильник електроенергії для збору даних про показники споживання. За період з 20 по 26 травня 2024 року було зібрано та оброблено значний обсяг даних, які включали розпізнані значення споживання електроенергії, що зображені на рисунку 3.1.

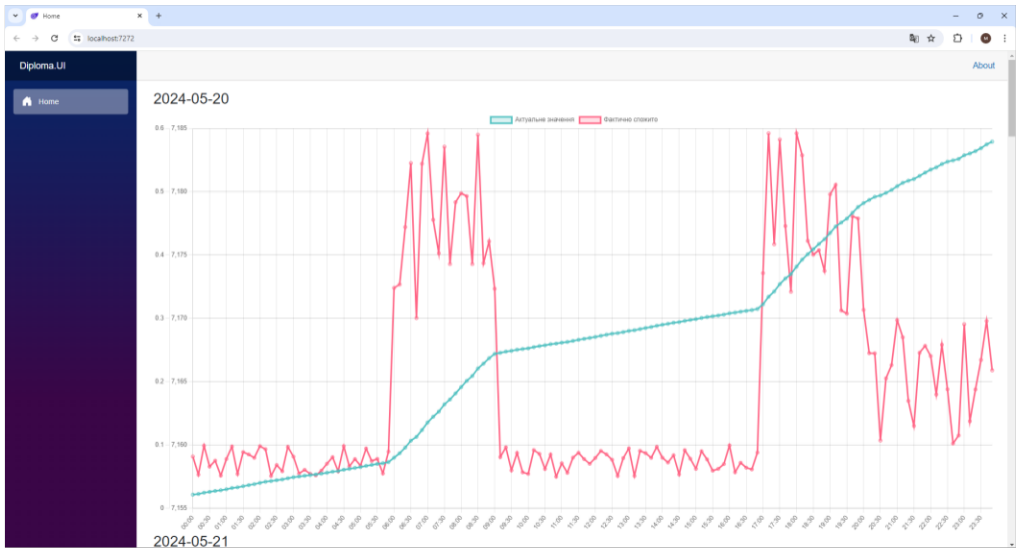

Рисунок 3.1 – Зібрана статистика відображена у застосунку з графічним інтерфейсом

3.5.1 Оцінка ефективності системи на основі зібраних даних

Протягом тестового періоду пристрій M5Stack TimerCam регулярно здійснював зйомку показників лічильника кожні 10 хвилин. Зображення передавалися на сервер через Wi-Fi, де вони оброблялися за допомогою бібліотеки Tesseract для розпізнавання тексту. Розпізнані дані зберігалися в базі даних для подальшого аналізу та візуалізації. Зібрані дані були відображені у графічному застосунку, розробленому з використанням Blazor та Chart.js, що дозволило користувачам аналізувати споживання електроенергії у зручній графічній формі.

Оцінка ефективності системи показала, що M5Stack TimerCam успішно виконує свої функції зі збору та передачі даних. Завдяки регулярній зйомці та

точному розпізнаванню тексту, система забезпечила високий рівень точності даних про споживання електроенергії. Візуалізація даних у вигляді графіків дозволила користувачам легко аналізувати споживання та виявляти пікові періоди навантаження.

3.5.2 Аналіз стабільності та надійності системи

Аналіз стабільності та надійності системи показав, що M5Stack TimerCam демонструє стабільну роботу в умовах реального використання. Пристрій працював автономно від батарейок, що забезпечило його безперервну роботу протягом усього тестового періоду. Незважаючи на можливі несприятливі умови, такі як змінна температура та вологість, пристрій показав високу стійкість та надійність.

Серверна частина системи також продемонструвала високу надійність та продуктивність. Обробка зображень за допомогою Tesseract виконувалася ефективно, забезпечуючи точне розпізнавання показників лічильника. Всі отримані дані були успішно збережені в базі даних, а графічний інтерфейс забезпечував своєчасне відображення інформації для користувачів.

Таким чином, результати роботи системи показали її високу ефективність, стабільність та надійність. Використання M5Stack TimerCam у поєднанні з серверною частиною на базі ASP.NET Web API та графічним застосунком на основі Blazor і Chart.js дозволило створити потужний інструмент для автоматизованого збору, обробки та візуалізації даних про споживання електроенергії.

3.6 Оптимізація та вдосконалення

3.6.1 Пропозиції щодо вдосконалення функціоналу

Для покращення функціоналу системи варто розглянути можливість конфігурації пристрою з сервера. Це дозволить адміністратору системи віддалено змінювати налаштування пристрою M5Stack TimerCam, такі як

частота зйомки, параметри з’єднання та інші важливі параметри. Це значно полегшить управління пристроями та підвищить гнучкість системи.

Ще однією корисною функцією може стати встановлення камери нічного бачення. Це дозволить підвищити якість знімків, зроблених в умовах низького освітлення, і забезпечити точність розпізнавання показників лічильника незалежно від часу доби.

Додавання авторизації до системи є важливим кроком для підвищення безпеки та контролю доступу. Впровадження механізмів аутентифікації користувачів дозволить обмежити доступ до даних і функцій системи лише авторизованим користувачам. Це може включати використання токенів аутентифікації, ролей та прав доступу, що забезпечить захист чутливих даних та запобігатиме несанкціонованому доступу.

3.6.2 Перспективи розвитку системи

Перспективи розвитку системи включають розширення її функціональних можливостей та підвищення її надійності. Одним із напрямів розвитку може стати розширення аналітичних можливостей системи. Впровадження більш складних алгоритмів аналізу даних, таких як машинне навчання, може дозволити передбачати майбутні тенденції у споживанні енергії та надавати рекомендації користувачам щодо оптимізації їх енергоспоживання.

Важливим кроком у розвитку системи може стати впровадження функцій віддаленого оновлення програмного забезпечення пристроїв. Це дозволить швидко впроваджувати нові функції та виправлення без необхідності фізичного доступу до пристроїв, що значно підвищить ефективність підтримки системи.

Таким чином, вдосконалення функціоналу системи електронної подачі показників лічильника шляхом додавання можливості конфігурації пристрою з сервера, встановлення додаткового освітлення або використання камери нічного бачення, а також впровадження авторизації та інших заходів безпеки значно підвищить її ефективність, гнучкість та надійність. Це забезпечить користувачам більш зручний та функціональний інструмент для моніторингу та

управління енергоспоживанням, сприяючи загальному покращенню роботи системи.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було досягнуто виконання всіх поставлених завдань.

Було проведено детальний аналіз проблематики автоматизації подачі показників лічильника. Встановлено, що існуючі методи є трудомісткими та схильними до людських помилок, що вимагає створення автоматизованої системи для підвищення точності та ефективності. Аналіз також виявив потенційні ризики, пов’язані з надійністю передачі даних та зберіганням інформації.

Сформовано вимоги до предмету розробки, які включають стійкість до зовнішніх умов, енергоефективність, автономність, та здатність до обробки та передачі даних. Особлива увага приділена забезпеченню безперебійної роботи пристрою в різних середовищах, включаючи промислові та міські інфраструктури.

Визначено інструменти, які найкраще підходять для виконання поставлених задач. Обрано M5Stack TimerCam для апаратної частини завдяки його компактності, функціональності, високій енергоефективності та можливості автономної роботи від батарейок. Для програмної частини обрано ASP.NET Web API для серверної частини, бібліотеку Tesseract для обробки даних, та .NET Blazor з бібліотекою Chart.js для графічного застосунку.

Реалізовано спосіб захоплення даних з лічильника за допомогою M5Stack TimerCam, який забезпечує регулярну зйомку показників та передачу зображень на сервер. Це дозволяє отримувати точні дані з лічильника без необхідності ручного зчитування.

Встановлено з’єднання між пристроєм та сервером за допомогою Wi-Fi, що дозволяє надійно передавати зібрані дані для подальшої обробки та зберігання. Це забезпечує безперебійну комунікацію між клієнтською та серверною частинами системи.

Проведено збір даних з лічильника електроенергії протягом певного проміжку часу з інтервалом 10 хвилин. Отримані дані були використані для тестування системи, що дозволило оцінити її ефективність та стабільність роботи в реальних умовах.

Розроблено графічний застосунок, який відображає зібрані дані у вигляді графіка. Це дозволяє користувачам легко аналізувати дані та робити висновки щодо споживання енергії. Візуалізація даних забезпечує зручний інтерфейс для моніторингу та аналізу.

Оцінено переваги готового рішення, включаючи зниження трудомісткості процесу, підвищення точності та можливість безперебійної роботи в різних умовах. Система забезпечує автоматизацію збору, обробки та візуалізації даних, що робить її ефективним інструментом для управління споживанням енергії.

Запропоновано можливі покращення системи, такі як можливість конфігурації пристрою з сервера, використання камери нічного бачення та впровадження механізмів авторизації для підвищення безпеки. Це дозволить зробити систему більш гнучкою та безпечною для користувачів.

Таким чином, виконання цих завдань дозволило створити ефективну, надійну та зручну систему для автоматизації збору, обробки та візуалізації даних лічильника.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Що таке розумний лічильник і чи варто його встановлювати?. *Прикарпаттяобленерго*. URL: https://oe.if.ua/uk/articles/63219124db9c421c239d4af4 (дата звернення: 28.02.2024).

2. Карпець А. Нові «розумні лічильники»: В чому їх особливість та кому встановлять БЕЗКОШТОВНО?. *Радіо ТРЕК*. URL: https://radiotrek.rv.ua/news/novi-rozumni-lichilniki-v-chomu-yih-osoblivist-ta-komu-vstanovlyat-bezkoshtovno_305023.html (дата звернення: 17.03.2024).

3. Гринів С. В., Кизимишин Л. П. Технології виробництва та використання розумних лічильників води. *Студентські наукові дискусії поза форматом* : Міжнар. наук. конф., м. Івано-Франківськ, 11 квіт. 2024 р. 2024. С. 135–137. URL: https://ukd.edu.ua/sites/default/files/2024-05/збірник%20Студентські%20дискусії_2024.pdf#page=136 (дата звернення: 8.03.2024).

4. Огляд системи дистанційного зчитування лічильників електроенергії. *LINSHU*. URL: https://ua.wirelessmartsystems.com/info/overview-of-remote-meter-reading-system-of-ele-71236224.html (дата звернення: 14.03.2024).

5. Unit CAM. *m5-docs*. URL: https://docs.m5stack.com/en/unit/unit_cam (дата звернення: 21.03.2024).

6. ASP.NET Web APIs Rest APIs with .NET and C#. *Microsoft*. URL: https://dotnet.microsoft.com/en-us/apps/aspnet/apis (дата звернення: 29.03.2024).

7. Tesseract OCR. *GitHub*. URL: https://github.com/tesseract-ocr/tesseract/blob/main/README.md (дата звернення: 4.04.2024).

8. Research Guides: Tesseract OCR Software Tutorial. *Home - Research Guides at New York University*. URL: https://guides.nyu.edu/tesseract/home (дата звернення: 10.04.2024).

9. How to do OCR in C# with OCR Tesserate. *Luis Llamas*. URL: https://www.luisllamas.es/en/csharp-ocr-tesseract/ (дата звернення: 12.04.2024).

10. Firmware Development: Things You Need to Know - Sirin Software. *Sirin Software*. URL: https://sirinsoftware.com/blog/firmware-development-things-you-need-to-know (дата звернення: 14.04.2024).

11. Visual Studio: IDE and Code Editor for Software Developers and Teams. *Visual Studio*. URL: https://visualstudio.microsoft.com/ (дата звернення: 18.05.2024).

12. About Arduino. *Arduino - Home*. URL: https://www.arduino.cc/en/about (дата звернення: 21.04.2024).

13. Singh, C. Build a RESTful Web API with .NET 8. *Medium*. URL: https://medium.com/@chandrashekharsingh25/build-a-restful-web-api-with-net-8-44fc93b36618 (дата звернення: 29.04.2024).

14. GitHub - DapperLib/Dapper: Dapper - a simple object mapper for .Net. *GitHub*. URL: https://github.com/DapperLib/Dapper (дата звернення: 5.05.2024).

15. Create a New Database Project - SQL Server Data Tools (SSDT). *Microsoft Learn*. URL: https://learn.microsoft.com/en-us/sql/ssdt/how-to-create-a-new-database-project?view=sql-server-ver16 (дата звернення: 9.05.2024).

16. Chart.js Open source HTML5 Charts for your website. *Chartjs* URL: https://www.chartjs.org/docs/latest/ (дата звернення: 17.05.2024).

17. Blazor Build client web apps with C# .NET. *Microsoft*. URL: https://dotnet.microsoft.com/en-us/apps/aspnet/web-apps/blazor (дата звернення: 22.05.2024).

ДОДАТКИ

Додаток А
Прошивка M5Stack TimerCam

```
#include <WiFi.h>
#include <HTTPClient.h>
#include <esp_camera.h>

// WiFi credentials
#define ssid "wifi"
#define password "Dddd111,"

// Server details
#define SERVER "192.168.1.108"
#define PORT 5003

// Timer interval
#define INTERVAL 5000

// Timer variable
unsigned long timer = 0;

// Camera pin configuration for M5Stack TimerCAM
#define PWDN_GPIO_NUM     -1
#define RESET_GPIO_NUM    15
#define XCLK_GPIO_NUM     27
#define SIOD_GPIO_NUM     25
#define SIOC_GPIO_NUM     23

#define Y9_GPIO_NUM       19
#define Y8_GPIO_NUM       36
#define Y7_GPIO_NUM       18
#define Y6_GPIO_NUM       39
#define Y5_GPIO_NUM       5
#define Y4_GPIO_NUM       34
#define Y3_GPIO_NUM       35
#define Y2_GPIO_NUM       32

#define VSYNC_GPIO_NUM    22
#define HREF_GPIO_NUM     26
#define PCLK_GPIO_NUM     21

camera_config_t config;

void setup() {
  Serial.begin(115200);

  // Initialize WiFi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connected to WiFi");

  // Configure camera
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
```

```
  config.pin_sccb_sda = SIOD_GPIO_NUM;
  config.pin_sccb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
  config.pixel_format = PIXFORMAT_JPEG;

  config.frame_size = FRAMESIZE_UXGA;
  config.jpeg_quality = 10;  // Low value for higher quality
  config.fb_count = 1;

  // Camera init
  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    return;
  }

  // Adjust camera settings for better quality
  sensor_t *s = esp_camera_sensor_get();
  s->set_brightness(s, 0);     // -2 to 2
  s->set_contrast(s, 2);       // -2 to 2
  s->set_saturation(s, 0);     // -2 to 2
  s->set_special_effect(s, 0); // 0 to 6
  s->set_whitebal(s, 1);       // 0 = disable , 1 = enable
  s->set_awb_gain(s, 1);       // 0 = disable , 1 = enable
  s->set_wb_mode(s, 0);        // 0 to 4, 0 = auto, 1 = sun, 2 = cloud, 3 =
office, 4 = home
  s->set_exposure_ctrl(s, 1);  // 0 = disable , 1 = enable
  s->set_aec2(s, 1);           // 0 = disable , 1 = enable
  s->set_ae_level(s, 0);       // -2 to 2
  s->set_aec_value(s, 300);    // 0 to 1200
  s->set_gain_ctrl(s, 1);      // 0 = disable , 1 = enable
  s->set_agc_gain(s, 15);      // 0 to 30
  s->set_gainceiling(s, (gainceiling_t)0);  // 0 to 6
  s->set_bpc(s, 1);            // 0 = disable , 1 = enable
  s->set_wpc(s, 1);            // 0 = disable , 1 = enable
  s->set_raw_gma(s, 1);        // 0 = disable , 1 = enable
  s->set_lenc(s, 1);           // 0 = disable , 1 = enable
  s->set_hmirror(s, 0);        // 0 = disable , 1 = enable
  s->set_vflip(s, 0);          // 0 = disable , 1 = enable
  s->set_dcw(s, 1);            // 0 = disable , 1 = enable
  s->set_colorbar(s, 0);       // 0 = disable , 1 = enable

  // Focus adjustment (if applicable)
  if (s->id.PID == OV3660_PID) {
    s->set_vflip(s, 1);       // Flip it back
    s->set_brightness(s, 1);  // Up the brightness just a bit
    s->set_saturation(s, -2); // Lower the saturation
  }

  // Start timer
  timer = millis();
}

void loop() {
  if (millis() - timer > INTERVAL) {
    captureAndSendImage();
    timer = millis();
  }
}
```

```
void captureAndSendImage() {
  camera_fb_t *fb = esp_camera_fb_get();
  if (!fb) {
    Serial.println("Camera capture failed");
    return;
  }

  HTTPClient http;
  http.begin("http://" SERVER ":" + String(PORT) + "/Image/UploadImage");
  http.addHeader("Content-Type", "image/jpeg");


  int httpResponseCode = http.POST(fb->buf, fb->len);
  if (httpResponseCode > 0) {
    String response = http.getString();
    Serial.println(httpResponseCode);
    Serial.println(response);
  } else {
    Serial.print("Error on sending POST: ");
    Serial.println(httpResponseCode);
  }

  http.end();
  esp_camera_fb_return(fb);
}
```

Додаток Б

Вміст файлу ImageController

```
using Diploma.Entities.Models;
using Diploma.Server.Business.Interfaces;
using Microsoft.AspNetCore.Mvc;
using System.Drawing;

namespace Diploma.Server.Api.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class ImageController : ControllerBase
    {
        private readonly ILogger<ImageController> _logger;
        private readonly IStatisticService _statisticService;
        private readonly IImageService _imageService;

        public ImageController(
            ILogger<ImageController> logger,
            IStatisticService statisticService,
            IImageService imageService)
        {
            _logger = logger;
            _statisticService = statisticService;
            _imageService = imageService;
        }

        [HttpPost("UploadImage")]
        [Consumes("image/jpeg")]
        public async Task<IActionResult> UploadImage()
        {
            try
            {
                using var memoryStream = new MemoryStream();
                await Request.Body.CopyToAsync(memoryStream);
                var bitmap = new Bitmap(memoryStream);

                string text = _imageService.GetTextFromImage(bitmap);

                var res = await _statisticService.CreateStatisticRecord(
                    new CounterSnapshot
                    {
                        Id = Guid.NewGuid(),
                        CurrentValue = double.Parse(text),
                        CreatedAt = DateTime.UtcNow,
                    });

                return res ? NoContent() : StatusCode(418);
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, ex.Message);
                return StatusCode(500);
            }
        }
    }
}
```

Додаток В

Програмний код домашньої сторінки

```
using Microsoft.AspNetCore.Components;
using Microsoft.AspNetCore.Components.Web;
using Microsoft.JSInterop;
using Newtonsoft.Json;

namespace Diploma.UI.Components
{
    public partial class ChartComponent : ComponentBase
    {
        [Inject] IJSRuntime JSRuntime { get; set; }

        protected override Task OnInitializedAsync()
        {
            return base.OnInitializedAsync();
        }

        protected async override Task OnAfterRenderAsync(bool firstRender)
        {
            var chartData = new
            {
                type = "bar",
                data = new
                {
                    labels = new[] { "Red", "Blue", "Yellow", "Green", "Purple",
"Orange" },
                    datasets = new[]
                    {
                        new {
                            label = "# of Votes",
                            data = new[] { 12, 19, 3, 5, 2, 3 },
                            backgroundColor = new[]
                            {
                                "rgba(255, 99, 132, 0.2)",
                                "rgba(54, 162, 235, 0.2)",
                                "rgba(255, 206, 86, 0.2)",
                                "rgba(75, 192, 192, 0.2)",
                                "rgba(153, 102, 255, 0.2)",
                                "rgba(255, 159, 64, 0.2)"
                            },
                            borderColor = new[]
                            {
                                "rgba(255, 99, 132, 1)",
                                "rgba(54, 162, 235, 1)",
                                "rgba(255, 206, 86, 1)",
                                "rgba(75, 192, 192, 1)",
                                "rgba(153, 102, 255, 1)",
                                "rgba(255, 159, 64, 1)"
                            },
                            borderWidth = 1
                        }
                    }
                },
                options = new
                {
                    scales = new
                    {
                        yAxes = new[]
                        {
                            new {
                                ticks = new
                                {
                                    beginAtZero = true
                                }
                            }
                        }
```

```
                    }
                }
            };

            var chartDataJson = JsonConvert.SerializeObject(chartData);

            await JSRuntime.InvokeVoidAsync("eval", $"new
Chart(document.getElementById('myChart').getContext('2d'), {chartDataJson})");
            await base.OnAfterRenderAsync(firstRender);
        }
    }
}
```