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

Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

«На правах рукопису»

УДК 004.08

До захисту допущено:

В. о. завідувача кафедри

_____ Олександр РОЛІК

«___»_____2021 р.

Магістерська дисертація

на здобуття ступеня магістра
за освітньо-професійною програмою «Інформаційне забезпечення робототехнічних систем»
зі спеціальності 126 «Інформаційні системи та технології»
на тему: «Веб-орієнтована система аналізу показників робототехнічних датчиків в розумному будинку»

Виконав:
студент VI курсу, групи ІК-01мп
Костючик Ілля Вікторович _____

Керівник:
доцент, к.т.н., доц.,
Крилов Євген Володимирович _____

Рецензент:
доцент, к.т.н., доц.,
Ліхоузова Тетяна Анатоліївна _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Київ – 2021 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій

Рівень вищої освіти – другий (магістерський)

Спеціальність – 126 «Інформаційні системи та технології»

Освітньо-професійна програма «Інформаційне забезпечення робототехнічних систем»

ЗАТВЕРДЖУЮ

В. о. завідувача кафедри

_____ Олександр РОЛІК

«___»_____2021 р.

ЗАВДАННЯ

на магістерську дисертацію студенту

Костючику Іллі Вікторовичу

1. Тема дисертації «Веб-орієнтована система аналізу показників робототехнічних датчиків в розумному будинку», науковий керівник дисертації Крилов Євген Володимирович, к.т.н., доц., затверджені наказом по університету від «27» 10 2021 р. № 3587-с

2. Термін подання студентом дисертації: _____

3. Об’єкт дослідження: Веб-додаток для швидкого здійснення аналізу, моніторингу та операцій над показниками робототехнічних датчиків в розумному будинку

4. Вихідні дані: Технології та засоби для створення веб-орієнтованої системи аналізу показників робототехнічних датчиків в розумному будинку

5. Перелік завдань, які потрібно розробити: аналіз та опис предметної області; вибір та опис технологій, методів та засобів для розробки системи; опис програмного забезпечення; розробка програмного забезпечення; маркетинговий аналіз стартап-проєкту

6. Орієнтовний перелік ілюстративного матеріалу: структура системи; структура бази даних; взаємодія додатку з сервісом складних задач; структурна схема складних задач; алгоритм кешування веб-ресурсів; алгоритм реєстрації користувачів;

7. Орієнтовний перелік публікацій: _____

8. Дата видачі завдання: 01.09.2020

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Аналіз предметної області та постановка завдання	05.09.2021р.	
2	Вибір та аналіз технологій для розробки веб-додатку	17.09.2021р.	
3	Проєктування бази даних	25.09.2021р.	
4	Створення серверної та клієнтської частини	08.10.2021р.	
5	Оптимізація роботи веб-додатку	27.10.2021р.	
6	Тестування створеної системи	03.11.2021р.	
7	Оформлення пояснювальної записки	10.11.2021р.	
8	Висновки	25.11.2021р.	

Студент Ілля КОСТЮЧИК

Науковий керівник Євген КРИЛОВ

АНОТАЦІЯ

У даній роботі розглянуто вирішення проблеми аналізу та моніторингу показників датчиків температури повітря, вологості та світла в розумному будинку. До того ж, був проведений аналіз існуючих програмних засобів з визначенням їхніх головних недоліків та вигодів під час використання.

В роботі створено веб-орієнтовану систему аналізу показників робототехнічних датчиків в розумному будинку, що надасть змогу власникам оселі здійснювати аналіз показників температури повітря, вологості повітря та світла на визначених інтервалах часу та перегляд показників, отримані від датчиків, в поточний момент часу та здійснювати керування над ними. Також був розроблений адаптивний і дружній до користувача дизайн, що надасть змогу останнім користуватись додатком не лише на персональному комп’ютері, а також з мобільних пристроїв. Створений додаток може бути використаний в власній оселі.

Ключові слова: показник, аналіз, статистика, розумний будинок, датчик, веб-додаток, оптимізації HTTP запитів ASP.NET Core, C#, Sql Server, Azure App Service.

Розмір пояснювальної записки – 110 аркушів, містить 47 ілюстрацій, 8 додатків.

ABSTRACT

This paper considers the solution of the problem of analysis and monitoring of air temperature, humidity and light sensors in a smart home. In addition, an analysis of existing software was conducted to identify their main disadvantages and benefits during use.

A web-based system of analysis of robotic sensors in a smart home was created, which will allow homeowners to analyze air temperature, humidity and light at certain time intervals and view the indicators received from the sensors at the current time and control over them. An adaptive and user-friendly design has also been developed, which will allow the latter to use the application not only on a personal computer, but also from mobile devices. The created application can be used in your own home.

Keywords: indicator, analysis, statistics, smart home, sensor, web application,, optimization of HTTP requests ASP.NET Core, C #, Sql Server, Azure App Service.

The size of the explanatory note is 110 pages, contains 58 illustrations, 8 appendices.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА ТЕРМІНІВ

IoT (Internet of Things) – фізичні об’єкти, що містять технології для обміну даними з іншими пристроями через Інтернет або інші мережі зв’язку

SPA (Single-page application) – веб-додаток, який взаємодіє з користувачем шляхом динамічного переписування веб-сторінки новими даними з веб-сервера

API (Application programming interface) – програмний інтерфейс, який надає можливість іншим додаткам взаємодіяти з собою

ORM (Object-relational mapping) – техніка, яка дозволяє запитувати та маніпулювати даними з бази даних за допомогою об’єктно-орієнтованої парадигми

HTTP (Hypertext Transfer Protocol) – найпоширеніший протокол для обміну даними між браузером та сервером

IDE (Integrated development environment) – програмне забезпечення, яке надає розробникам комплексні засоби для розробки програмного забезпечення

TaskTrigger – елемент фонової задачі, який відповідає за її виконання

TaskDetail – елемент фонової задачі, який містить метадані необхідні для її виконання

AOT (Ahead of Time) – тип компіляції, що компілює код перед запуском програми, зазвичай додається як крок збірки додатку

JIT (Just-in-Time) – тип компіляції, що компілює код під час виконання програми

ВСТУП

Останні десятиліття варто відзначати надшвидкими темпами розвитку технологій для вирішення чимало проблем в різноманітних сферах життя, починаючи від важкої індустрії та закінчуючи повсякденним побутом. На сьогоднішній час найбільші темпи розвитку введуться в області інтернет технологій, оскільки останні полегшують життя як для бізнесу, так і для звичайної людини, а саме: створення та автоматизація бізнес процесів, здійснення аудиту даних, можливість спілкування в режимі реального часу в будь якій точці світу, пошук необхідної інформації, отримання даних про останні події в світі, навчання та відаллена робота, використовуючи будь який елетронний пристрій з доступом до глобальної мережі інтернет.

В повсякденному побуті за останні кілька років найбільшу популярність набирає система «Розумний будинок». Розумний будинок – це система, що дозволяє власникам осель дистанційно здійснювати керування безпечним доступ до дому, терморегуляторами, освітленням, а також отримувати актуальні показники зовнішніх умов в приміщенні в режимі реального часу за допомогою підключення до мережі Інтернету з використанням комп’ютерних, мобільних або інших мережевих пристроїв. Оскільки вони підключені до портативного пристрою, користувачі можуть отримувати регулярні сповіщення та оновлення щодо проблем у своїх будинках. Наприклад, розумні дзвінки в двері дозволяють власникам будинків бачити і спілкуватися з людьми, які підходять до їх дверей, навіть коли їх немає вдома. На сьогоднішній час існує чимало систем, які призначені для вирішення задач розумного будинку, однак більшість пропонують надмірний пакет функціональних можливостей в базовій версії, складні в конфігурації та вимагають високу плату за використання.

Варто зазначити, що в теперішній час при побудові нових будинків, прикладаються додаткові проводки та елементи управління, що необхідні для використання систем домашньої автоматизації. Оздоблення старого будинку

подібними елементами обходиться набагато дорожче через складність прокладання та розміщення датчиків у відповідних місцях.

Метою магістерської дисертації є створення веб-орієнтованої системи аналізу показників робототехнічних датчиків в розумному будинку, що надасть змогу власникам оселі спостерігати за зовнішніми умовами, здійснювати аналіз показників температури, вологості повітря та рівня освітлення, а також надсилати сповіщення на мобільні або мережеві пристрої при перевищенні останніми нормативних норм через засоби комунікації. Основна функціональність включатиме можливість створення складних задач для індивідуального збору та аналізу даних на заданих інтервалах часу для кожного користувача веб-додатку. До того ж, для спрощеного перегляду показників буде використовуватись багатофункціональна таблиця, яка надасть змогу здійснювати фільтрацію, групування, сортування тощо.

Основними вимогами до системи є простий, адаптивний та дружній до користувача інтерефейс, стійкість і надійність системи, висока продуктивність, коректність і точність аналізу отриманих даних. Дані вимоги будуть досягатись за рахунок використання сучасних засобів створення адаптивних елементів, оптимізації запитів між сервером та клієнтом, кешування веб-ресурсів та використанням технологій для встановлення двонаправленого зв’язку між браузером користувача та центральним сервером.

1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Загальний опис предметної області

Поняття «Розумний будинок», будинок, що здатний виконувати рутинну роботу замість вас (готувати їжу, прати ваш одяг, розважати мешканців), з’явилося близько півстоліття тому. Вперше такий термін згадується у творах американського письменника Рея Бредбері, а саме: спеціальний пристрій, який зв’язує ваші шнурки, засоби для чищення килимів, та інші, були представлені у багатьох творах письменника. З часом дану концепцію запозичили й інші письменники, і розумні будинки почали з’являтися у багатьох романах, а згодом – у фільмах. Однак, на той час, через відсутність технологій підключення між побутовими пристроями, розумні будинку були не дуже поширені [1].

Поширення інтернету в 90-х роках дало поштовх розвитку багатьом технологіям, включаючи технологій для розумного будинку. В даних роках розробники всесвітньо відомої компанії Oracle прогнозували, що найближчим часом за допомогою Інтернету холодильники зможуть здійснювати облік та замовлення продуктів харчування. Однак, на сьогоднішній час, така технологія ще не реалізована.

Сучасне визначення розумного будинку означає поєднання основних компонентів будинку в єдину систему управління. Освітлення, живлення, прибирання, опалення та кондиціонування, безпека вашого будинку чи квартири взаємодіють, роблячи власника оселі вільним від нудної та рутинної роботи. До речі, один із сучасних аспектів розумних будинків – це економія енергії.

Існують різні підходи до розумного будинку. Наприклад, в Європі головною причиною встановлення технології розумного будинку є її екологічність, тобто зменшення споживання енергії та викидів з дому, у більшості інших країн це престиж.

Для правильного функціонування розумного будинку найважливішим є взаємозв’язок усіх компонентів. Насправді розумний будинок – це надбудова, яка об’єднує окремі функції будинку. Існує кілька типів технологій розумного будинку.

Найпопулярніші з них це:

– LanDrive – найбільш поширена на сьогоднішній день. Серед головних фу це контроль внутрішнього та вуличного освітлення, електроенергії, електроприладів, а також таких систем, як опалення, кондиціонування, вентиляції, охоронної сигналізації, контролю доступу та витоку води. Крім того, є можливість стежити за аудіо та відеотехнікою, домашніми кінотеатрами, жалюзі, шторами та воротами;

– X10 – протокол управління електроприладами. Аналогічний попередньому типу, однак має декілька головних недоліків: низька швидкість передачі даних, проблема помилкової тривоги, відсутність зворотного зв'язку приймача з передавачем. Крім того, можуть виникнути конфлікти між пристроями різних виробників та можливий несанкціонований доступ до пристроїв через електричну мережу;

– 1-Wire – технологія, що об’єднує безліч датчиків і пристроїв в одну мережу, яка управляється з використанням пристроїв, що підтримують мережевий зв’язок. Для передачі даних використовується лише один провід;

Розумний будинок здатний виконувати багато функцій, наприклад: кондиціонер не почне працювати, коли відчинене вікно; коли в двір заходить людина, зовнішнє освітлення вмикається автоматично; коли вдома нікого немає, він може увімкнути систему безпеки або режим економічного споживання ресурсів: вимкнути електрику там, де це не потрібно, відключити воду, контролювати роботу вентиляційної системи тощо.

Існують спеціальні компанії, які допомагають вибрати розумний будинок, вони уточнюють потреби та програмують систему відповідно до побажань власника. Такі системи зовсім не дешеві, але вони можуть бути оснащені численними автоматизованими компонентами.

1.2 Огляд існуючих рішень

Перед розробкою веб-орієнтованої системи аналізу показників датчиків в розумному будинку було проведенно дослідження та аналіз існуючих рішень, та на їх

основі було визначено, що на сьогоднішній час існує безліч систем для здійснення основних функцій в розумному будинку, однак, багато з них мають безліч недоліків, а саме: повільна обробка та завантаження інформації, перенасичення веб-сторінок зайвою інформаційєю, а також трапляються випадки з недружнім до користувача веб-інтерфейсом. Вище наведені фактори, можуть «відштовхнути» кінцевого користувача від користування даними системами, оскільки їм буде не зручно користуватись додатками, що вимагають посиленої концентрації на потрібній інформації, а також здійснюють обробку певної інформації за великий проміжок часу.

В даній частині розділу наведено огляд трьох найрозповсюджених та найбільш популярних рішень для розумного будинку, а саме: Home Assistant, openHAB та Microsoft IoT Azure. Розглянемо спочатку «Home Assistant».

Home Assistant – це безкоштовне програмне забезпечення з відкритим вихідним кодом для організації розумного будинку, що була розроблена як центральна система управління пристроями розумного будинку з акцентом на локальний контроль та конфіденційність. Доступ неї можна отримати через веб-інтерфейс користувача, через відповідні мобільні додатки, як для платформи Android, так і для iOS або за допомогою голосових команд через підтримувані голосові помічники, такі як Google Assistant або Apple Siri.

Головний графічний інтерфейс системи зображений на рис.1.1:

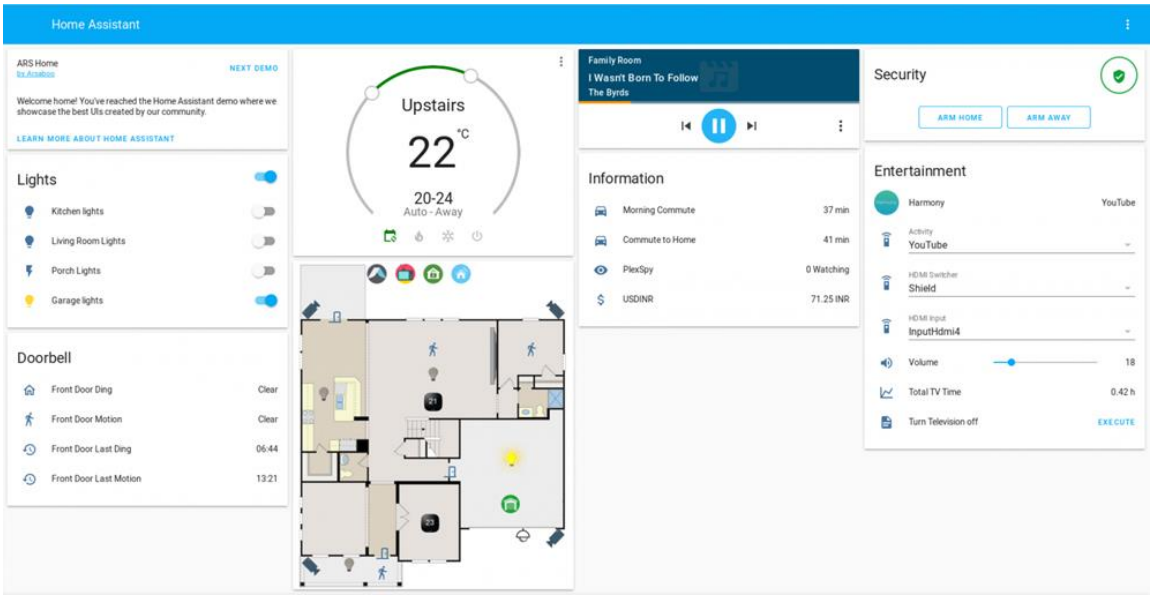


Рисунок 1.1 – Головний графічний інтерфейс [2]

Даний інтерфейс, який ще називається Lovelace UI представляє собою SPA (Single-Page-Application) – використовує єдиний HTML-документ як оболочку для всіх сторінок, де дані оновлюються шляхом динамічного переписування однієї веб-сторінки.

На вище наведеному рисунку, можна помітити, що інтерфейс скаладається з карток різноманітність типів, які використовуються для виконання функцій та відображення отриманих з датчиків даних. Картки можна розподіляти по сторінках, по аналогії з закладками браузера. Налаштовувати позиції карточок можна як через налаштування, так і за допомогою yaml-коду, оскільки є вбудований текстовий редактор.

Такоє є можливість створювати і свої картки. Такі картки прийнято називати «custom cards», і для їх створення використовується javascript. У мережі нескладно знайти готові картки, створені різними розробниками, а якщо хочеться поекспериментувати самому, то варто звернути увагу на спеціальні js-модулі, створені для спрощення роботи над написанням карточок.

Серед недоліків інтерфейсу варто виділити недостатню адаптивність як на екрані монітора, так і на мобільному пристрої. LUI, за замовчуванням, любить самостійно визначати розташування і розміри карток, що іноді може виглядати нормально на екрані монітора, але досить не адаптивно на екрані смартфона, і навпаки. Присутні деякі прості інструменти для упорядкування інтерфейсу, однак, вони не є завжди ефективними.

Технології інтернету речей, пристрої та сервіси підтримуються за допомогою компонентів модульної інтеграції, які не тільки включають вбудовану інтеграцію для локальних протоколів підключення, таких як Bluetooth, MQTT, але також підтримуються сторонніми екосистемами, якщо вони надають загальний доступ через Open API для сторонніх інтеграцій. Після того, як програмне забезпечення Home Assistant встановлено як комп'ютерний пристрій, воно буде діяти як центральна система управління для домашньої автоматизації.

Для керування та отримання даних з пристроїв Home Assistant використовує окремі модулі. Будь-який модуль складається з певного набору об'єктів та сервісів (по

суті – функції). Об'єкти використовуються для зберігання різних отриманих від пристрою даних. Наприклад, sensor.temperature – температуру повітря в приміщенні, sensor.light_state – показник освітлення. Показники об'єкта складаються з одного основного значення та довільного набору додаткових атрибутів.

Якщо об’єкти використовуються для зберігання даних, то сервіси в свою чергу – для передачі команд і значень пристроїв. Наприклад, lights_state.on – команда, що відповідає за ввімкнення світла в приміщенні, або homeassistant.check_config – здійснює пошук та виправлення помилок в файлах налаштувань НА. Для перегляду та керування доступними сервісами необхідно перейти в розділ Services, що знаходиться в списку Developer Tools. Поруч розташовується розділ States, де, відповідно, можна переглянути і поредагувати значення об’єктів.

Створити окремий модуль доволі нескладно. На офіційному сайті знаходиться перелік основних (схвалених і підтримуваних спільнотою) модулів. Серед загальної їх кількості (1485 штук) трапляються абсолютно різноманітні.

Одна з основних функцій розумного будинку – оповіщення користувача при певних умовах, і НА містить дану функціональність. В НА такі сповіщення називаються «notifications» і останні поділяються на два базових типи:

– внутрішні сповіщення: Для їх відправлення використовується вбудований сервіс «persistent_notification.create». Список таких повідомлень доступний через іконку дзвіночка в графічному інтерфейсі;

– зовнішні сповіщення. Сповіщення даного типу надсилаються на сторонні платформи, які потребують додаткового підключення. Прикладом може бути надсилання сповіщень на месенджери;

Наприкінці опису даної системи, варто зазначити, що Home Assistant може зацікавити тих, хто має намір спробувати організувати локальне управління розумним будинком. Це широкий, цікавий та відкритий проєкт, що активно розвивається за рахунок зусиль ком’юніті. Серед головних мінусів варто додати відносну складність розуміння системи на початку користування, заплутаність і неповноту документації.

Наступним проаналізованим існуючим рішенням стала система openHAB (Open Home Automation Bus). openHAB – система автоматизації пристроїв в розумному будинку з відкритим вихідним кодом, що була написані на мові програмування Java в 2010 році. Під автоматизацією пристроїв розуміється об’єднання всіх останніх в єдину систему упраління.

На відміну від аналогів система реалізує уніфікований протокол обміну даних, тобто дозволить об'єднати всі пристрої з різними протоколами в єдину мережу, тим самим абстрагуючись від виробників. В кінцевому результаті, це дає можливість використовувати єдиний засіб управління (наприклад, додаток на смартфоні) та реалізовувати будь яку складну логіку взаємозв'язку між пристроями [3].

Основні переваги:

– надає можливість об'єднати різні технології домашньої автоматизації в одну;
– може працювати на будь якій ОС, що підтримує Java Virtual Machine;
– надає API для інтеграції в інші системи;
– повністю з відкритим вихідним кодом;
– незалежність від виробника;
– підтримується спільнотою;

Щодо інтерфейсу то система має простий, адаптивний, дружній до користувача інтерфейс, а також надає можливість побудови користувацького інтерфейсу для управління пристроями на свій вибір за допомогою файлів опису «sitemap», що знаходяться в директорії configurations/sitemaps. На рис. 1.2 наведено приклад побудови інтерфейсу:

```
sitemap demo lbl="Main" {
  Frame lbl="Widgets" {
    Slider item=DimdLight switchSupport
    Colorpicker item=RGBLight icon="slider"
    Switch item=DemoShutter
    Slider item=DemoBlinds
  }
}
```

Рисунок 1.2 – Приклад побудови інтерфейсу в openHAB [3]

Як можна зрозуміти з коду, то тут UI має один "фрейм", що містить 4 елементи (вибір кольору для підсвічування, два слайдера для управління диммерами, простий перемикач).

Однак, опис за допомогою sitemap, можна віднести до недоліків системи, оскільки, набагато зручніше було б описувати за допомогою HTML, проте, за словами розробників, будь-яка програма або javascript клієнт в браузері, використовуючи sitemap, зможе візуалізувати інтерфейс по-своєму, таким чином, абстрагуючи опис інтерфейсу від конкретної реалізації. Також варто зазначити, що зміна sitemap призводить до зміни UI на клієнті без перезавантаження сервера.

Однією з основних функцій системи є тригери. Тригера – це подія, яка виникає при виконанні одніє з наступних умов. Найбільш поширеними є:

– надходження команди від підключеного пристрою (наприклад, при ввімкненні перемикача світла);

– при зміні часу (наприклад, при опівночі);

– зміна стану підключених пристроїв (наприклад, зміна рівня освітлення при відкритті занавіски);

– старт або зупинка сервера openHAB;

Можна помітити, що тригерів цілком вдосталь для реалізації будь-якої логіки автоматизації, заснованої на на той чи інший тип події.

Наостанок аналізу системи незважаючи на її численні переваги варто виділити головні недоліки:

– складний процес встановлення. Незважаючи на те, що він має функцію автоматичного виявлення додаткових компонентів, які необхідно додатково встановити, повне встановлення системи для більшості користувачів є громіздким через доволі тривалий процес;

– повільний розвиток. Оскільки OpenHAB вірить у перевірку всіх пунктів перед інтеграцією нового пристрою Інтернету речей, процес розробки та інтеграції повільніший порівняно з його аналогами;

– ретельний процес тестування та схвалення додавання цих нових пристроїв займає значну кількість часу. Ось чому ви можете не бачити нові випуски або оновлення OpenHAB так часто;

Варто пам’ятати, що openHAB – це лише комп’ютерна програма, яка буде виконувати все, що ви йому скажете. Вона може запропонувати багато стандартних рішень, які легко налаштувати. З іншого боку, чим більше ви наполягаєте на тому, що все повинно працювати та виглядати саме так, тим більше роботи вам доведеться вкласти, однак це вимагатиме від вас чимало зусиль.

Останньою системою для розгляду став сервіс «Azure IoT» платформи Microsoft Azure. Сервіс «Azure IoT» – це набір керованих служб платформи Azure, що дозволяють підключити, моніторити та керувати сотнями пристроями IoT. Інакше кажучи, складається з одного або декількох пристроїв IoT, які комунікують з одним або декількома сервісами, розміщеними в хмарі. Також містить пакети безпеки та операційні системи для пристроїв, дані й аналітику, які допомагають кінцевим користувача створювати та керувати пристроями Інтернету речей.

Серед головних можливостей варто відзначити просту інтеграцію з будь-якими існуючими системами, чудову масштабованість. Сервіс дозволяє користувачу підключати сотні пристроїв різних виробників, збирати та обробляти дані за допомогою служб машинного навчання та аналізу Analytics Azure. Також є можливість отримувати повідомлення, які надсилаються в IoT Hub з інших програм і служб в Azure, тому можна створити інформаційну панель Microsoft BI для перегляду даних у реальному часі. Надає сервіс «Azure Advisor», який є персональним помічником та надає рекомендації щодо покращення продуктивності, безпеки та витрат.

Платформа представляє приємний і дружній до користувача інтерфейс, головна сторінка якого зображена на рис 1.3.

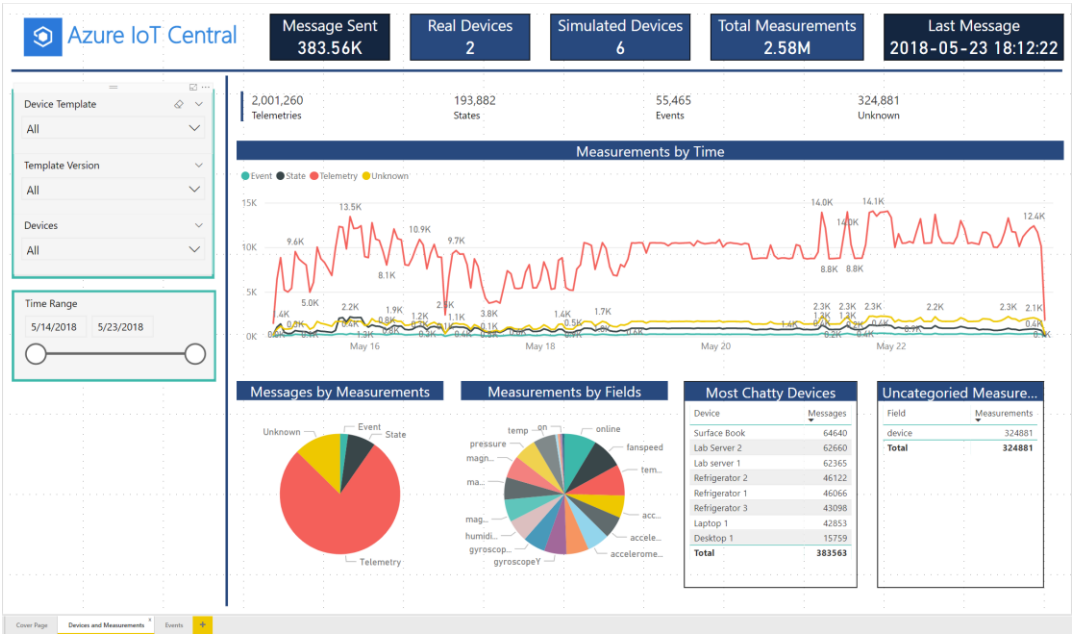


Рисунок 1.3 – Головна сторінка сервісу «Azure IoT» [4]

Сторінка має сучасний, приємний та дружній до користувача інтерфейс. Зліва розташована панель фільтрів, за допомогою якої можна здійснити фільтрацію даних отриманих з певних пристроїв та в визначеному проміжку часу. Справа зображений графік, який в реальному часу виводить інформацію про кількість отриманих повідомлень в певний момент часу. Знизу розміщені діаграми, що показують відношення повідомлень за їхніми типами та пристроями.

Перш ніж користуватись сервісом необхідно здійснити початкову реєстрацію на платформі «Microsoft Azure». Для цього необхідно заповнити реєстраційну форму, що містить наступні поля: номер телефону, email, прізвище, ім’я, місто та адресу проживання. Наступним етапом є підтвердження свого номеру телефону та прив’язка банківської карти з якої будуть списуватись кошти за використання платних сервісів. Після успішного підтвердження номера телефону та перевірки банківської карти, вас буде запропоновано ознайомитись з оновними можливостями платформи. Після успішної реєстрації новим користувачам надається 200$ кредиту, а також вільний доступ до дванадцяти найбільш популярних платних сервісів протягом одного року.

Для початку роботи з сервісом, спочатку необхідно створити IoT Hub. Під IoT Hub розуміється певний набір взаємопов'язаних IoT пристроїв. Далі потрібно створити IoT пристрій (наприклад датчик освітлення), ключ та зареєструвати його в

IoT Hub. Цей ключ базується на ідентифікаторі пристрою, і це робиться шляхом створення невеликої консольної програми, яка генерує ключ пристрою та реєструє його в IoT Hub. Останнім етапом є налаштування зареєстрованих пристроїв, тобто створення програмного забезпечення, що буде здійснювати збір даних з датчика та відправляти їх до сервісу, використовуючи рядок підключення, що була згенерована останнім. Для перегляду отриманих даних необхідно перейти на панель керування.

Головними недоліками системи є висока плата за використання, надання надмірного функціоналу в базовому пакеті, а також складість підключення пристроїв.

1.3 Постановка завдання

Поставленою метою в даній магістерській дисертації є розробка веб-орієнтованої системи аналізу показників робототехнічних датчиків в розумному будинку. Щоб досягнути вище поставлену мету необхідно здійснити наступні дії:

- здійснити аналіз і виявити основні переваги та недоліки існуючих рішень;
- на основі проведеного аналізу визначитись з структурою, основними модулями та елементами майбутнього додатку;
- створити структуру бази даних;
- реалізувати механізм фонових задач, що полягає в створенні та керуванні отриманню відповідних даних на визначених проміжках часу;
- реалізувати REST API та логіку запитів, отриманих з браузера користувача;
- створити простий та дружній до користувача інтерфейс системи;
- покрити основну бізнес логіку системи модульними тестами;
- оптимізувати систему за рахунок бандлінгу та мініфікації веб-ресурсів, що дозволить зменшити кількість запитів до веб-серверу та розмір завантаження даних;
- здійснити масштабування системи;

В результаті буде створена система, яка буде здійснювати отримання та аналіз показників датчиків в режимі реального часу, керувати фоновими задачами для підтягування даних в визначений проміжок часу, матиме сучасний, адаптивний дизайн та буде доступна для персонального використання.

Висновок до розділу 1

Напочатку даного розділу був здійснений опис предметної області. В підсумку даного опису, можна сказати, що сучасне визначення розумного будинку означає поєднання основних компонентів будинку в єдину систему управління, а саме: освітлення, живлення, опалення тощо, і одним із сучасних аспектів розумних будинків – це економія енергії. Також варто навести, що поштовх розвитку технологій для розумного будинку дав стрімкий розвиток інтернет технологій 90-х роках.

Після цього було проаналізовано декілька найпопулярніших існуючих систем для розумного будинку для отримання необхідної інформації щодо їх переваг та недоліків. Першою системою стала платформа з відкритим вихідним кодом «Home Assistant». Головними переваги «Home Assistant» є її швидкодія, сумісність з пристроями різних виробників, кофіденційність, а також не вимагає плати за використання. Серед недоліків варто виділити заплутність, відносну складність розуміння системи на початку користування та неповноту документації. Другою системою стала openHAB. Варта відзначити в ній можливість об'єднувати різні технології домашньої автоматизації в одну, надання API для інтеграції в інші системи та кросплатформеність. Головними недоліками є повільний розвиток, складний процес встановлення, а також ретельний процес тестування та схвалення додавання цих нових пристроїв займає значну кількість часу. Останньою системою для розгляду став сервіс «Azure IoT». Головними перевагами даного сервісу є проста інтеграцію з будь-якими існуючими системами, чудову масштабованість, обробка даних за допомогою машинного навчання. Однак система має декілька головних недоліків, а саме: висока плата за використання та складність підключення нових пристроїв.

В кінці розділу була поставлена мета створити власну систему, яка буде здійснювати аналіз даних в розумному будинку в режиму реального часу, керувати задачами для отримання даних в певні моменти часу, а також матиме адаптивний та сучасний дизайн. Разом з цим було описано основні кроки для досягення вище поставленої мети.

2 ПЕРЕЛІК ТА ОПИС ЗАСОБІВ РОЗРОБКИ

Оскільки метою дисертації є розробка системи аналізу показників датчиків, яка матиме форму веб-додатку, нам необхідно здійснити аналіз та обрати технології для розробки серверної та клієнтської частини. Для створення серверної частини була вибрана мова програмування C# та платформа .NET, оскільки на сьогоднішній час є одним з найкращих варіантів для створення масштабованих та високопродуктивних систем. Оскільки інтерфейс матиме форму SPA (Single-page application), то для створення REST API будемо використовувати фреймворк ASP.NET Core, що широко використовується для створення веб-додатків в даній платформі. Для покриття основної бізнес логіки використаємо інстумент написання тестів та тестування xUnit.net.

Для розробки клієнтської частини буде використовуватись мова розмітки HTML та каскадні таблиці стилів CSS. Для створення клієнтської логіки обробки подій та взаємодії між елементами сторінок буде використовуватись мова програмування JavaScript. Щоб зменшити час розробника при створенні клієнтської частини використаєм фреймворк стилів Bootstrap, оскільки навідміну від головного конкурента Material UI, забезпечує чіткий і послідовний інтерфейс користувача для всіх платформ. Разом з цим містить систему сіток з дванадцяти колонок, що буде використовуватись для створення адаптивного контенту. Також для пришвидшення та полегшення створення стилів будем використовувати препроцесор Less, який містить зручні доповнення до звичайних каскадних стилів.

Для створення бази була вибрана реляційна система управління базами даних Microsoft SQL Server, оскільки є найбільш сумісною з вибраною платформою та підходить для створення додатків різних масштабів. В даній СУБД різновидом SQL являється T-SQL, що буде використовуватись для читання, модифікації та видалення існуючих записів з таблиць бази даних.

Розглянемо більш детально про засоби та технології, що будуть використані для розробки додатку.

2.1 C# та платформа .NET

Мова програмування C# є простою, універсальною та об’єктно-орієнтованою мовою програмування, створена компанією Microsoft в 2000 році, головна мета якої – об’єднання обчислювальних потужностей C++ з легкістю програмування на Visual Basic.

Використання даної мови програмування розробники мають змогу створювати крос-платформені додатки, веб-портали, веб-сервіси, десктопні додатки, ігрові застосунки, що придатні для розгортання в розполілених хмарних середовищах.

Основні примітні функціональні можливості:

– Портативність – відображає базову загальномовну інфраструктуру CLR, тобто більшість його внутрішніх типів відповідають типам значень, реалізованим фреймворком CLR;

– Метапрограмування – використання рефлексії для отримання метаданих про об’єкт, expression tree, що дозволяє динамічно змінювати виконуючий код під час виконання, використання атрибутів для надання об’єктам певних метаданих тощо;

– LINQ (Language Integrated Query) – зручна мова запитів до різних джерел даних, таких як: стандартні колекції, документ XML, база даних тощо. LINQ дозволяє використовувати одні й тіж самі методи, незалежно від джерела даних;

– Функціональне програмування – надає легкий синтаксис для використання локальних функцій, анонімних методів, лямда виразів, техніки pattern marching та замикання;

Під час створення C# Microsoft активно співпрацювала з ECMA, міжнародним органом стандартів, щоб створити стандарт для C#. Визнання C# Міжнародною організацією зі стандартизації (ISO) спонукало б інші компанії розробляти власні версії мови. Співпраця з вище згаданими організаціями ведеться і дотепер [5].

Програму на C# можна представити у вигляді взаємопов'язаних взаємодіючих між собою об'єктів, оскільки в її основі лежить об’єктно-орієнтоване програмування (ООП). Об’єкти створюються на основі класів, що містять опис їхніх станів та поведінки. Класи можуть наслідуватись один від одного для успадкування базової

функціональності, що надає можливість повторно використовувати код і незалежно розширювати оригінальне програмне забезпечення за допомогою загальнодоступних класів та інтерфейсів[6]. На рис. 2.1 зображено приклад побудови класів:

```
public class Person
{
    public string FstName { get; }
    public string LstName { get; }

    public virtual string RetrieveDesc()
        => $"Name: {FstName}{Environment.NewLine}" +
           $"Surname: {LstName}{Environment.NewLine}";
}
public class Employee : Person
{
    public decimal Sal { get; }

    public override string RetrieveDesc()
    {
        var baseDesc = base.RetrieveDesc();
        return baseDesc += $"Salary: {Sal}{Environment.NewLine}";
    }
}
```

Рисунок 2.1 – Приклад побудови класів

На вище наведеному рисунку зображено два класи: Person та Employee. Клас Person містить дві властивості: FirstName та LastName, що позначають стан об’єкта та один метод GetDescription, що позначає поведінку об’єкта. Даний метод містить ключове слово virtual, що надасть дочірнім класам можливість його перевизначити. Клас Employee наслідується від Person, тим самим успадковуючи вище описані властивості та методи. До успадкованої функціональності він додає одну властивість Salary та перевизначає метод GetDescription за допомогою ключового слова override. На початку оголошення властивості чи об’єкта знаходиться ключове слово public, це один з модифікаторів доступу, що робить їх доступними поза межами класу. Основними модифікаторами доступу в C# є: public, protected та private [7].

C# працює на базі платформи .NET (рис. 2.2). .NET – відкритий та кросплатформений фрейворк розробки для створення веб-додатків, мобільних, настільних, IoT-додатків тощо.

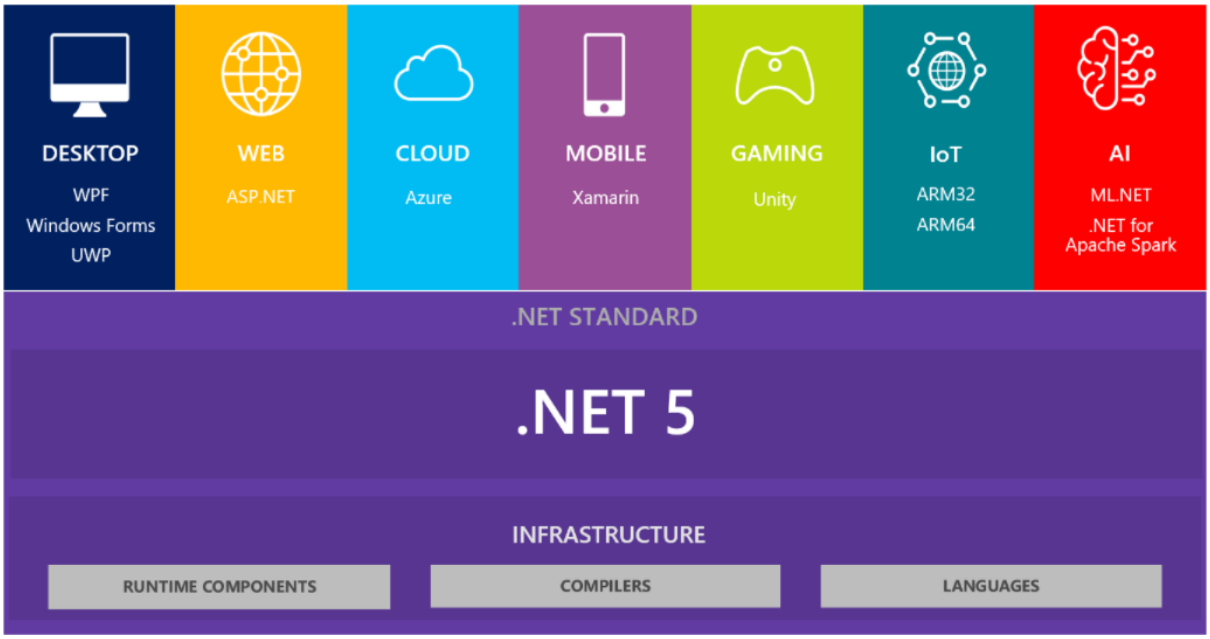


Рисунок 2.2 – Платформа .NET [8]

Хоча C# вважається основною мовою розробки на .NET, ви також можете використовувати інші мови програмуваня на ваш вибір, такі як: Visual Basic та F#.

В основі .NET лежить CLI (Common Language Infrastructure), тобто вихідний код підтримуваної мови компілюється в CIL (Common Intermediate Language. Даний механізм дає можливість створювати окремі модулі однієї програми різними мовами.

Окрім вищезгаданих мов, які безпосередньо підтримуються Microsoft, багато інших мов програмування можуть компілюватися в .NET CIL. Наприклад, IronPython, Eiffel, ClojureCLR тощо.

.NET оснований на двох основних компонентів:

– CoreCLR – це механізм виконання додатку, що містить наступні основні компоненти: JIT (Just-in-Time) – забезпечує компілювання коду безпосередньо перед його виконанням; Thread Management – дозволяє запускати кілька потоків виконання; Type loading – знаходження та завантаження збірок і типів даних; Garbage Collector – автоматичне керування пам’яттю та видалення непотрібних об’єктів;

– CoreFX – уніфікована бібліотека базових класів, що реалізує стандартні бібліотеки CLI, тобто набір бібліотек, які забезпечують найпоширеніші функції, наприклад: створенння та маніпулювання колекціями, обробка винятків, керування

файловою системою, мережеві комунікації, керування пулом потоків та рефлексією тощо;

.NET містить різні моделі додатків, що працюють поверх вище описаних компонентів, тобто бібліотеки, які надають базову функціональність для розробки різних типів додатків. Наприклад:

– Blazor – UI-фреймворк для створення веб-додатків за допомогою C#, які можуть працювати як на стороні сервера, так і на стороні клієнта. А для опису візуального інтерфейсу використовуються стандартні HTML та CSS. Він також дозволяє створювати клієнтські веб-програми в коді WebAssembly;

– WPF – підсистема .NET для створення графічного інтерфейсу користувача для настільних програм Windows. Програми написані на WPF засновані на архітектурі векторної графіки, що дозволяє чудово виглядати на моніторах з високим DPI;

– ASP.NET – веб-фреймворк, що застосовується для створення динамічний сайтів, додатків та веб-API;

– ML.NET – фреймворк для машинного навчання, який спрощує інтеграцію моделей машинного навчання у вашу програму;

– Xamarin – фреймворк для створення кросплатформних мобільних, телевізійних і настільних додатків;

На додаток до моделей, .NET пропонує підтримку для більшості поширених завдань програмування: від керування файлами до безпеки доступу, від мережевого зв’язку до бази даних. Наприклад, на стороні мережі він підтримує програмування сокетів, зв’язок HTTP та gRPC. Це дозволяє створювати мікросервіси з протоколом, який краще відповідає вашим потребам [9].

Для будь-яких інших потреб, не вбудованих у фреймворк, ви можете знайти величезну кількість конкретних бібліотек у загальнодоступному сховищі NuGet. Фактично NuGet є менеджером пакетів для .NET. Він дозволяє створювати, ділитися та використовувати багато бібліотек .NET практично для будь-яких цілей.

Спочатку .NET був представлений як .NET Framework. З самого початку метою Microsoft було створення універсальної платформи для програмування з будь-якою

мовою і Windows була одна з них. Однак швидкий розвиток та популярність інших платформ призвело до народження кількох проєктів портування, таких як: .NET Core та .NET Standard.

.NET Core це по суті повністю переписаний .NET Framework з урахуванням кросплатформної мети. Його перероблена архітектура визначає мінімальний набір функцій як загальне ядро для платформ Windows, Linux і Mac.

.NET Standard це специфікація API, яка допомагає створювати міжплатформні бібліотеки. Якщо платформа підтримує певну версію .NET Standard, на ній працюватиме бібліотека, яка підтримує ту саму версію, незалежно від типу пристрою та реалізації фреймворка.

2.2 EF Core

EF (Entity Framework) Core – це розширена технологія об’єктно-реляційного відображення (ORM) від Microsoft, яка абстрагує об’єктну модель програми від її реляційної або логічної моделі. Тобто він ізолює об’єктну модель від того, як дані фактично представлені в реляційному сховищі і бере на себе всі зобов'язання по збереженню моделі в сховище. Ця структура робить концептуальну модель реальною, використовуючи розширену модель зв’язку сутності.

Реляційне сховище використовується для збереження, узгодженості та безпеки даних. Він містить дані програми і зазвичай складається з набору таблиць, представлень, відносин, функцій тощо. Для використання та маніпуляціями даними ми зазвичай використовуєм T-SQL до БД, яке повертає результат, що містить стовпці та рядки даних.

Проте в більшості випадків модель повернутих даних не обов’язково збігаються з моделями об’єктів вашого додатку. Зазвичай ми не використовуємо дані, що повертаються з реляційного сховища. Пишемо необхідний код щоб перетворити дані, повернуті з реляційного сховища на бізнес-об’єкти в даних рівень доступу до програми [10]. Аналогічно, вам потрібно написати код, щоб перетворити ваш бізнес-об’єкти програми у форму, яку можна зберегти у вашому реляційному сховищі.

Використання ORM спрямовано на збільшення продуктивності розробників, оскільки ORM бере на себе більшість базових задач таких як:

– створення з'єднань з базою даних, виконання команд, а також отримання результатів запитів і автоматичну конвертацію результатів в моделі об'єктів вашої програми;

– відслідковування змін в цих об’єктах та за певними вказівками зберігання цих змін у базу даних;

Навідміну від інших ORM на ринку таких як Nhibernate, які зіставляють типи доменів безпосередньо зі схемою бази даних, що продемонстровано на рис. 2.4.

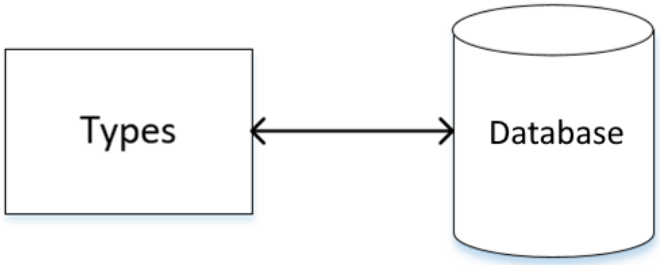


Рисунок 2.3 – Мапінг в звичайні ORM

Entity Framework надає можливість створювати більш детальний рівень мапінгу, наприклад: зіставлення однієї сутності з кількома таблицями бази даних або навпаки, що зображено на рис. 2.4.

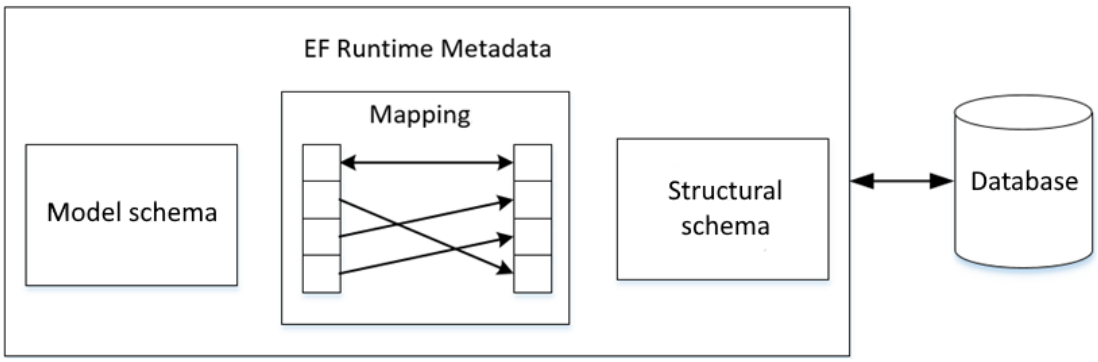


Рисунок 2.4 – Мапінг в EF Core

Однією з головною перевагою Entity Framework Core є те, що він дозволяє зосередитися на доменній моделі:

– ключовим поняттям в даній ORM є концептуальна модель – модель об’єктів у вашій програмі, а не модель бази даних, яку ви використовуєте для збереження даних;

– концептуальна модель може бути ідентичною зі схемою вашої БД або повністю відрізнятись;

– створити концептуальну модель можна за допомогою візуального конструктора, яка в результаті згенерується в C# класи, які в кінцевому підсумку використовуватимуться у додатку;

На рис. 2.5 зображена схема базової взаємодії Entity Framework Core з веб додатком:

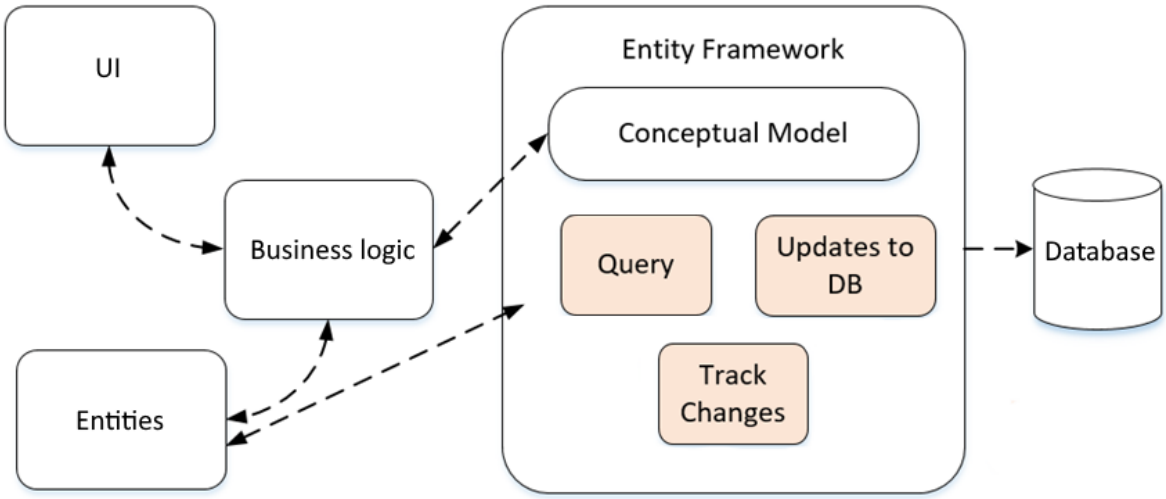


Рисунок 2.5 – Взаємодія EF з веб-додатком [10]

Entity Framework пропонує три підходи для взаємодії з даними:

– Database First – один з перших підходів для взаємодії з даними, спочатку відбувається проєктування бази даних, а далі EF генерує коди моделі (класи, властивості тощо) з бази даних у проєкті, і ці класи стають проміжними між базою даних і контролером;

– Model First – в даному підході вам необхідно створити сутності, зв’язки та ієрархії успадкування, використовуючи відповідний редактор моделі та на основі створеної моделі відбувається генерації бази даних;

– Code First – зосередження йде на домені вашої програми, тобто спочатку йде створення класів для сутностів доменної області, замість проєктування бази даних;

Так як на сьогоднішній час не існує єдиного синтаксису SQL (Sql Query Language) для взаємодії з базою даних в різних систем управління базами даних, тобто мова запитів написана для однією СУБД може не працювати на іншій, може виникнути проблема при розробці додатків, що використовують декілька баз даних. Entity Framework усуває дану проблему шляхом використання діалектів популярних баз даних, серед них: Sql Server, Oracle DB, Sqlite тощо [11].

2.3 ASP.NET Core

ASP.NET Core – одна з основних моделей платформи .NET, що використовується для створення веб-застосунків, сервісів, IoT додатків різних масштабів, що легко масштабуються та можуть бути розміщені в хмарному середовищі.

Підтримуючи концепцію платформи .NET, він був розроблений за модульним принципом з мінімальними накладними витратами, інші модульні компоненти можуть бути додані як NuGet пакети відповідно до вимог програми. В кінцевому результаті ми отримуємо додаток з високою продуктивністю, що потребує менше пам’яті, меншого розміру розгортання та легко підтримується [12].

Основні переваги використання даної платформи:

– швидко: навідміну від свого попередника ASP.NET, що залежив від бібліотеки System.Web.dll для зв’язку браузер-сервер, ASP.NET Core дозволяє включати веб-компоненти, що безспосередньо необхідні для створення програми, як результат, це покращує масштабованість та продуктивність;

– кросплатформеність: веб-застосунки створені на базі ASP.NET Core можуть працювати на Windows, Linux і Mac, тому немає необхідності в створенні різних програми для різних платформ;

– інтеграція з сучасними веб-фреймворками: містить інструменти для легкої інтеграції з сучасними фреймворками, такими як Angular, ReactJS, Bootstrap тощо, та керування ними, використовуючи вбудований менеджер пакетів Bower;

– спільний доступ до коду: бібліотеки створені на базі даної платфтрми, легко можуть бути використані .NET Framework та Mono, тобто єдина кодова база може бути спільною між фреймворками;

– вбудований контейнер IoC: представляє механізм, який дозволяє зробити об'єкти слабопов'язаними, тобто об'єкти пов'язані між собою через абстракції, що робить всю систему більш розширюваною, гнучкішою та адаптованішою;

– хостинг: веб-додаток ASP.NET Core може розміщуватися на кількох платформах із будь-яким веб-сервером, таким як IIS, Apache тощо. Він не залежить лише від IIS навідмінно від свого попередника;

В основі ASP.NET Core лежить використання компонентів middleware для здійсненні обробки запитів [13]. Тобто, спочатку за обробку HTTP запиту береться перший компонент, далі після обробки він передає дані другому компоненту тощо. Компонент middleware може або передати запит далі наступному в конвеєрі компоненту, або виконати обробку та закінчити роботу конвеєра. Також компонент middleware у конвеєрі може виконувати обробку запиту як до, так і після наступного у конвеєрі компонента. На рис. 2.6 зображений даний процес:

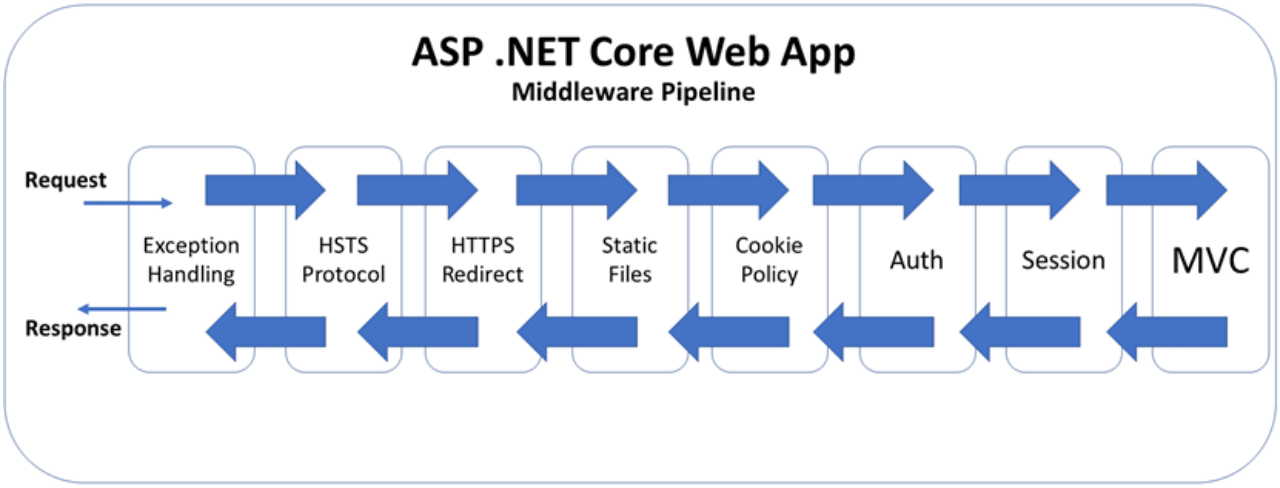


Рисунок 2.6 – Компоненти middleware [14]

Використання ASP.NET Core дає нам можливість створювати веб-додатки за паттерном MVC (Model-View-Controller). Хоча даний паттерн в деяких комьюніті розробників вважається легасі, оскільки він був створений ще в минулому столітті, а також для створення гнучких більш доцільним є використання веб-сервісів та веб-фрейворків, він дотепер є одним з кращих рішень для створення інформаційних систем простих та середніх масштабів. В концепції даного паттерни лежать три основних компоненти: модель, представлення та контроллер.

– модель містить лише чисті дані програми, вона не повинна містити логіки для опису представлення даних користувачеві;

– представлення відповідає за представлення користувачеві даних моделей. Представлення знає, як отримати доступ до даних, проте воно не повинно знати як ними маніпувати та що вони означають;

– контроллер виступає проміжним мостом між моделлю та представленням. Він відповідає за обробку вхідних даних з клієнта користувача, шляхом виклику методів бізнес логіки для маніпулювання даними моделі, після закінчення оброки він передає результат обробки назад до представлення, щоб користувач міг побачити їх побачити [15];

2.4 SQL Server

SQL Server – це одна з найпопулярніших реляційних систем управління базами даних (СУБД) у світі, що підходиться для проєктів різних масштабів, починаючи від невеликих додатків та закінчуючи високонавантаженими системами.

Як і будь яка інша система управління базами даних, SQL Server побудований на основі структурованої мови запитів (SQL) для взаємодії з базами даних. Дана СУБД виділяє власну реалізацію SQL у формі Transact-SQL (T-SQL), яка додає набір власних програмних конструкцій. Починаючи з 2016 року SQL Server став доступним для розгортання на Linux платформі, до цього часу він працював виключно в середовищі Windows.

На рис. 2.7 зображено базову архітектуру SQL Server:

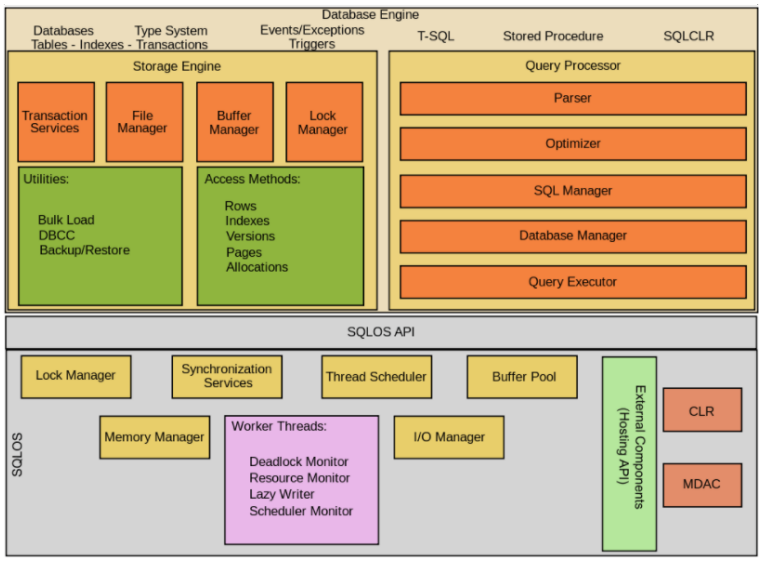


Рисунок 2.7 – Архітектура SQL Server [16]

Дана СУБД складається з двох основних компонентів:

– Database engine – являється основним компонентів системи управління базами даних. Він містить Relational Engine (Query Processor), що в свою чергу має компоненти, які визначають найкращий спосіб виконання запиту. Relational Engine запитує дані від механізму зберігання даних на основі вхідного запиту та обробляє результати. Деякі завдання реляційного механізму включають управління потоками, пам'яттю та обробку запитів, розподілену обробку запитів та управління буфером. За створення та виконання об’єктів баз даних (представлень, тригерів) також відповідає Database Engine. Також він має Storage engine, що відповідає за керування файлами бази даних, сторінками, індексами, здійснення обробки, зберігання та отримання даних з систем зберігання, таких як диски, тощо [16].

– SQLOS – знаходиться на найнижчому рівні архітектури СУБД. Даний компонент відповідає за надання служб операційної системи, управління вводом-виводом, пам’яттю тощо. Разом з цим здійснює надає служби синхронізації та здійснює обробки виключень;

Структуровану мова запитів в залежності від типів команд можна розділити на три основні типи:

– Data Definition Language (Мова визначення даних) – до даного типу належать команди, які відповідають за створення, зміну, видалення об’єктів баз даних та всіх даних в таблиць. Прикладом таких команд є: CREATE, ALTER, DROP та TRUNCATE;

– Data Manipulation Language (Мова маніпулювання даними) – містить команди з додавання, оновлення, видаленн, вибірки даних, тобто усі команди за допомогою яких ми маємо змогу здійснювати маніпуляцію даних. Основні команди даного типу: INSERT, UPDATE, DELETE та SELECT;

– Data Control Language (Мова керування доступу до даних) – один з типів команд, які здійснюють керування правами доступу, наприклад: надання або видалення прав на додавання даних до таблиць. Зокрема, це такі команди як: GRANT та REVOKE [17];

SQL Server має чотири основні версії, які надають розробникам різні інструменти та пакети служб. В наступній таблиці 2.1 міститься перелік та опис кожного з них.

Таблиця 2.1 – Порівняння версій SQL Server

Версія	Визначення
Enterprise	Забезпечує повний набір можливостей для зберігання та керування даними з необмеженими можливостями засобів бізнес-аналітики, що дозволяє досягти рівня обслуговування важливих робочих навантажень
Standard	Надає основні функції управління даними та підтримуються поширені засоби розробки в обчислювальних хмарах, за рахунок якого досягається ефективне управління БД.
Developer	Містить функціональні можливості версії Enterprise, однак на відмінну від останнього, ліцензується як система для розробки та тестування.

Версія	Визначення
Express	Даний випуск є безкоштовною базою даних та чудово підходить для навчання, а також для створення невелеких проєктів, що працюють на робочих станціях та невеликих серверах.

Для створення веб-орієнтованої системи аналізу показників датчиків в розумному будинку, ми використаєм версію Express, оскільки вона містить достатній функціонал для керування базою даних та легко може бути завантажена в хмарне середовище.

2.5 CSS та препроцесор Less

Cascading Style Sheets (CSS) – це одна з найпопулярніших мов каскадних таблиць стилів, яку використовують для опису представлення HTML та/або XML документа та його елементів, підтримуючи діалекти XML, такі як XHTML, SVG та MathML. Каскадні таблиці стилів здійснюють опис елементів, що мають бути відображені на електронних носіях.

CSS є однією з найпопулярнішою мов каскадних стилів в Інтернеті, наведемо основні його переваги:

– економія часу: написавши стилі для однієї сторінки, ви можете повторно використати їх для інших веб-сторінок, або глобально описати як буде той чи інший елемент відображатися на всіх сторінках;

– легкість: для внесення глобальних змін, необхідно лише змінити стиль таблиць, і всі елементи автоматично оновляться;

– швидкість: можливість написати єдине правило для атрибутів тегу, що буде застосовуватись в усіх місцях його використання, таким чином це дозволить не описувати атрибути щоразу, що збільшить швидкість завантаження сторінок;

– сумісність: таблиці стилів надають можливість описати інший спосіб відображення сторінок на інших типах пристроїв, тобто використовуючи один

HTML-документ, можна описати різне представлення веб-сайту для портативних пристроїв, таких як мобільні пристрої, тощо;

– кращі стилі перед HTML: HTML має менший набір атрибутів ніж CSS, тому ви можете надати набагато кращий вигляд веб-сторінці, використовуючи атрибути CSS;

– глобальні веб-стандарти: Оскільки HTML-атрибути стають застарілими і рекомендацією є використання CSS-атрибутів, тому для сумісності з майбутніми браузерами варто почати використовувати CSS на всіх веб-сторінках;

CSS є стандартизованим відповідно до W3C специфікації та однією з основних мов відкритої мережі. Оскільки розробка різних частин CSS раніше відбувалась синхронно, що дозволяло вести версійність, тобто CSS1, CSS2.1 та CSS3. Обсяг специфікації, починаючи з CSS3, значно збільшився, що призвело до розробки та випускати рекомендації окремо для кожного модуля, оскільки прогрес у різних модулях CSS почав сильно відрізнятись. Замість версій специфікації CSS, W3C тепер періодично робить знімок останнього стабільного стану специфікації CSS [19].

Наведемо короткий опис синтаксису.

CSS містить правила стилів для кожного модуля, при підключенні їх на сторінку, браузер здійснює їхню інтерпретацію, а потім застосовує до відповідних елементів у веб-сторінці. Кожне правило складається з трьох основних частин:

– селектор (selector): зазвичай це HTML тег, до якого будуть застосовані стилі, наприклад: <p> або тощо;

– властивість (property): містить тип атрибута HTML-тегу, наприклад: це може бути колір тексту, тип шрифта, ширина елемента тощо;

– значення (value): значення що присвоюються до властивостей, наприклад: значення для властивості типу шрифта може мати значення ‘Times New Roman’;

Правило стилю CSS зображено на рис. 2.8:

```
selector { property: value }
```

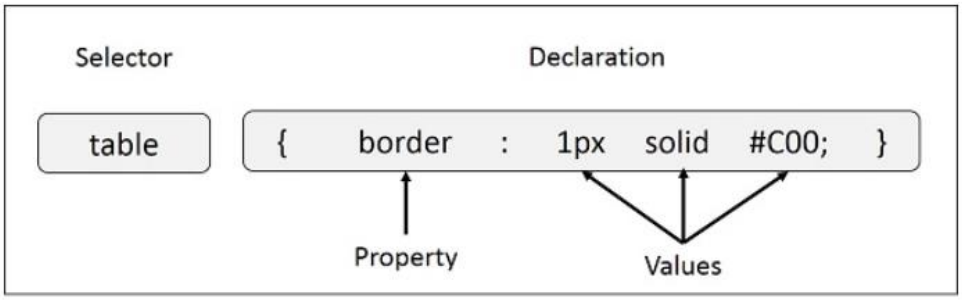


Рисунок 2.8 – Опис CSS стилів [20]

На сьогоднішній час при створенні веб-систем, як правило, замість CSS використовують його препроцесори, найбільш популярними з них є Sass та Less. В роботі ми використаєм Less, оскільки він має більш лаконічний синтаксис на відміну від його конкурента.

Less – один з найпопулярніших CSS препроцесорів, який дозволяє спростити написання, керування та повторне використання правил стилів для веб-сайту. Простіше кажучи, являється динамічною мовою таблиць стилів, що розширює можливості CSS та підтримує роботу з різними браузерами.

CSS препроцесор в свою чергу є мовою для написання сценаріїв, яка розширює можливості CSS та компілюється в звичайний CSS, оскільки браузер не розуміє препроцесорів [21]. Він надає такі зручні функції, як міксини, змінні та операції, які дозволяють створювати динамічний CSS.

Особливості:

– створення чистішого та зручного для різних браузерів CSS швидше та простіше;

– створений на JavaScript, а також створено для використання в реальному часі, який компілюється швидше, ніж інші препроцесори CSS.

– робить ваш код модульним;

– швидка підтримка за рахунок використання змінних;

Головні переваги:

– легка компіляція в CSS, який розуміють всі браузери;

– дозволяє писати більш лаконічний та організований код за допомогою вкладень;

– підтримка стилів досягається швидше за допомогою змінних;

– легке повторне використання цілого набору стилів в окремих класах, за рахунок посилань;

– надає використання операцій, що робить процес написання коду швидшим та економить час;

Однак використання препроцесорів несе за собою декілька недоліків:

– необхідний певний час, щоб ознайомитись та вивчити основні функціональні можливості;

– потрібно докладати більше зусиль для повторного використання залежних модулів, через їхній тісний зв’язок;

Проте, незважаючи на вище наведені недоліки, використання Less пришвидшує розробку в порівннянні з CSS.

2.6 JavaScript та Bootstrap

JavaScript – це легка, мульти-парадигмова, інтерпретована мова програмування. Призначена для створення мережевих додатків та підтримує декларативний, об’єктно-орієнтований та імперативний стиль програмування.

JavaScript являється відкритим та кросплатформеним, а також дуже простий у реалізації, оскільки має тісну інтеграцію з HTML. Сьогодні використання даного інструменту надає можливість писати код не тільки в браухері, але і на сервері, або власне на будь-якому пристрої, який має JavaScript Engine. Кожний браузер має вбудований двіжок, який іноді називають «Віртуальна машина JavaScript». Різні віртуальні машини мають різні «кодові назви», наприклад: V8 у Chrome, Opera та Edge, SpiderMonkey – у Firefox “SquirrelFish” – Safari тощо. Принцип інтерпретації віртуальними машинами скриптів здійснюється наступним чином: спочатку браузер парсить скрипт, далі відбувається компіляція сценаріїв на машинну мову, а потім машинний код запускається досить швидко. Движок застосовує оптимізацію на

кожному кроці процесу. Він навіть спостерігає за виконанням скомпільованого сценарію, аналізує дані, які протікають через нього, і додатково оптимізує машинний код на основі цих знань. Оскільки кожна віртуальна машина має власну реалізацію, тому при розробці необхідно враховувати даний аспект [21].

На сьогоднішній час існує немало клієнтських бібліотек та фреймворків, що написані на даній мові програмування, такі як: Angular, React, Vue.js, jQuery тощо. Остання бібліотека, до речі, буде використовуватись для розробки додатку, оскільки за допомогою вбудованих функцій, дозволяє зменшити в рази написання веб-скриптів. Як було сказано раніше, даний інструмент використовується практично в усіх областях створення додатків, найбільш поширеними при створенні як простих так і складних є:

– маніпуляція HTML елементами: надає функціонал для створення, видалення тегів документа, а також змінення параметрів та зовнішнього вигляду тегів, на основі використовуючих пристроїв;

– валідація на стороні клієнта: перевірка введених даних користувача перед відправкою останніх на сервер, і JS відіграє важливу роль у перевірці цих введених даних на самому інтерфейсі;

– завантаження серверних даних: підтримує техніку Ajax, яка відповідає за завантаження серверних даних, поки виконуються інші процеси, що надає чудовий досвід користувачам відвідувачам вашого веб-сайту;

– сповіщення: створення динамічних динамічних спливаючих вікон на веб-сторінках, для надання різних типів;

– серверні додатки: Node JS побудований на середовищі виконання Chrome Javascript для створення швидких і масштабованих мережевих програм. Це бібліотека на основі подій, яка допомагає розробляти дуже складні серверні програми, включаючи веб-сервери;

Незважаючи на вище наведені переваги, помилково вважати JavaScript як повноцінну мову програмування, оскільки останній має декілька відсутніх головних функцій:

– не дозволяє читати та записувати файли;

– не можна використовувати для мережевих програм;
– не має жодних багатопоточних чи багатопроцесорних можливостей;

Розглянемо базовий синтаксис даного інструменту (рис. 2.9):

```
var a = 0; // Оголошення змінної 'a'

a += 5 + b; // Присвоєння значення до змінної 'a'

bam(a, b); // Виклик функції 'bam', яка приймає агрументи 'a' та 'b'
obj1.bam(10); // Виклик методу 'bam' на об'єкті 'obj1'

// Умовний запит
if (a === 10) // Чи є змінна 'a' рівна нулю?
  x = 150;

function bam(x, y) { // Визначення функції 'bam' з аргументами 'x' та 'y'
  return x + y;
}
```

Рисунок 2.9 – Синтаксис JavaScript

Спочатку відбувається оголошення змінної a, оголошення здійснюється за допомогою ключового слова var, на сьогоднішій час це також можна зробити за допомогою таких ключових слів як: let та const (різниця між ними полягає в області визначення); далі здійснюється її ініціалізація, після того іде виклик визначеної функції bam, яка принімає два аргумента a та b; наступним етапом є виклик методу bam об’єкта obj1; далі йде умовна конструкція if, результат якої є присвоєння змінній a значення 150, якщо остання дорівнює нулю; наприкінці йде оголошення глобальної функції bam з двома параметрам, яка повертає результат суми останніх. Варто зазначити, що оголошення глобальних функцій також може йти після її виклику, оскільки під час парсингу віртуальна машина спочатку шукає глобальні змінні та функції, а тоді запускає скрипт послідовно на виконання [22].

Стандартами для JavaScript є специфікація мови ECMAScript (ECMA-262) та специфікація ECMAScript (ECMA-402). Але, оскільки специфікація є досить формалізованою, спочатку її важко зрозуміти, однак вона є найбільш надійним джерелом інформації про деталі мови, специфікація – це правильне місце. Але це не для щоденного використання. На сьогоднішній момент часу останньою версією JS це ES2021.

В нинішній час для пришвидшення розробки адаптивних макетів веб-сторінок, як правило використовують різні UI-фреймворки, ми використаємо ui-фреймворк Bootstrap.

Bootstrap – це один з найкращих та найпопулярніших front-end фреймворків, призначений для простого та швидкого створення адаптивних веб-додатків.

Даний UI-фреймворк містить в собі шаблони дизайну на основі CSS, JavaScript та HTML для створення дружніх до користувача компонентів інтерфейсу, таких як: кнопки, форми введення, сповіщення, модальні форми, вкладки, тощо. Також за допомогою Bootstrap можна зробити багато інших речей, наприклад: легко створити макет із кількома стовпцями за допомогою попередньо визначених класів; швидко створювати різні макети форм; адаптивне створення різних варіантів панель навігації, тощо [23].

Головні переваги при використанні Bootstrap:

– економія часу: використовуючи заздалегідь визначені HTML-класи та шаблони для створення звичайних компонентів, розробник зекономить чимало часу, і зможе зосередитись більш на бізнес логіці;

– послідовний дизайн: дизайн вашої веб-сторінки набуває узгодженого вигляду, оскільки усі компоненти використовують однакові стилі дизайну через через основну бібліотеку;

– простий у використанні: розробник, який має базові знання HTML, CSS та JavaScript, може почати роботу з Bootstrap без викладання значних зусиль;

– адаптивні функції: застосування класів, які адаптивно відображатимуть вміст сторінки на різних пристроях та на роздільних можливостях екрана без будь яких змін у розмітці;

– сумісність з браузерами: даний UI-фреймворк було створено з урахуванням сучасних браузерів таких як: Chrome, Firefox, Safari тощо;

Одним з основних компонентів Bootstrap є система сіток, що була створена за допомогою flexbox з mofile-first підходом. Дана система сіток складається з дванадцяти стовпців у кожному рядку та містить шість основних рівнів за замовчуванням (табл. 2.2).

Таблиця 2.2 – Система сіток Bootstrap

Функціонал	X-Small <576px	Small ≥576px	Medium ≥768px	Large ≥992px	X-Large ≥1200px	XX-Large ≥1400px
Максимальна ширина контейнера	None (auto)	540px	720px	960px	1140px	1320px
Префікс класу	.col	.col-sm	.col-md	.col-lg	.col-xl	.col-xxl
Кількість колонок	12					
Ширина падингу	1.5rem (.75rem зліва та справа)					
Власний падинг	Так					
Впорядкування стовпців	Так					

Згідно вище наведеної таблиці можемо сказати, що сітка містить шість основних рівнів: x-small, small, medium, large, x-large та xx-large. Для створення макетів для маленьких пристроїв, таких як мобільні телефони доцільно використовувати x-small; рівень small може бути використаний для мобільних пристроїв у альбомному режимі; рівень medium та large доцільно використовувати для створення макетів з орієнтацією на планшети та невелеких моніторів; x-large та xx-large як можна здогадатись для екранів вашого комп’ютера або моніторів з високою роздільною здатністю.

Для застосування сітки спочатку необхідно створити контейнер, який буде діяти як обгортка для рядків і стовпців, використавши клас контейнера .container, далі необхідно створити рядки всередині контейнера за допомогою класу .row, тепер всередині будь-якого рядка можемо створювати стовпці, використовуючи класи необхідного нами рівня.

2.7 Платформа Microsoft Azure

Для розгортання та хостингу додатку використаємо хмарну платформу Microsoft Azure. Microsoft Azure – це загально доступна платформа Microsoft для хмарних обчислень, яка надає цілий ряд сервісів, включаючи бчислення, зберігання та аналітику. Користувачі можуть вибирати з цих служб для розробки та масштабування нових додатків або запуску існуючих програм у загальнодоступній хмарі.

В попередньому абзаці був згаданий вислів «хмарні обчислення», який є досить поширеною тенденцією на сьогоднішній час. Під ним зазвичай розуміють зберігання даних і доступ до них через мережу Інтернет. Це означає, що ваші дані зберігаються ні на жорсткому диску, ні на виділеній мережі, а на віддаленому хосту, який може знаходитись навіть за межами країни в якій ви проживаєте, та разом з цим синхронізуються з іншою веб-інформацією. Дана синхронізація з іншою веб-інформацією та пов’язаними пристроями проілюстрована на рис. 2.10:

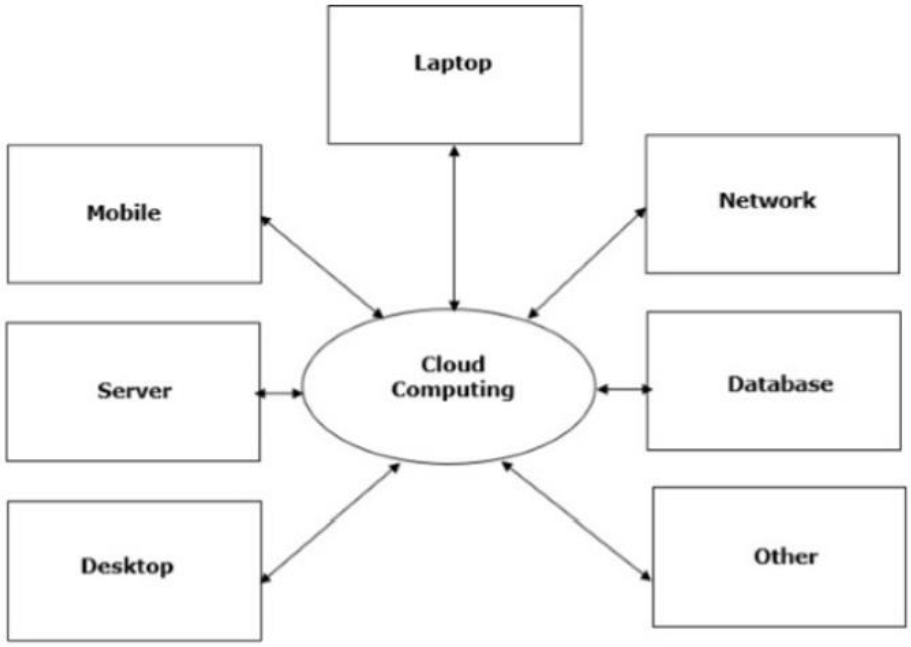


Рисунок 2.10 – Хмарні обчислення [25]

Найбільш поширеним прикладом хмарних обчислень є Office365, який дозволяє кінцевим користувачам керувати, отримувати доступ до своїх документів, не встановлюючи необхідний пакет програм на свої пристрої [25].

Існують три форми зберігання даних в хмарі:

– public: збережені дані стають доступними для широкої громадськості, тобто є відкритими для всіх, а їхня інфраструктура керується та належить постачальниками послуг;

– private: зазвичай використовується для розміщення бізнес-додатків або даних певної організації. Цими даними не можна поділитися з іншими організаціями, та даним типом керує лише сама організація;

– hybrid: даний тип хмар означає об’єднання приватних та публічних хмар, для, щоб надати переваги обох одночасно. Зазвичай використовується організаціями коли необхідно використати публічні для нечутливих додатків, а приватні для чутливих;

Використання хмарної платформи надає наступні переваги:

– масштабованість: хмарний сервіс забезпечує динамічний розподіл та вивільнення ресурсів відповідно до заданих вимог;

– економія: за рахунок зменшення капітальної інфраструктури відбувається економія на витратах;

– безпека: надає сучасні засоби та протоколи для шифрування та доступу до даних, тому ризик втратити дані є мінімальним;

– доступність: дозволяє користувачеві отримати доступ до веб-служб та програм незалежно від їхньої конфігурації та розташування;

Для розгортання та хостингу даного веб-додатку використаємо службу Azure App Service (рис. 2.11). Azure App Service – це одна з найбільш популярних слажб платформи, яка дозволяє створювати та розміщувати мобільні сервери, REST API та власне веб-додатки, без керування інфраструктурою та прив’язки до мови програмування. Разом з цим забезпечує високу продуктивність та автоматичне масштабування, а також підтримує завантаження коду безпосередньо з Azure DevOps та GitHub.

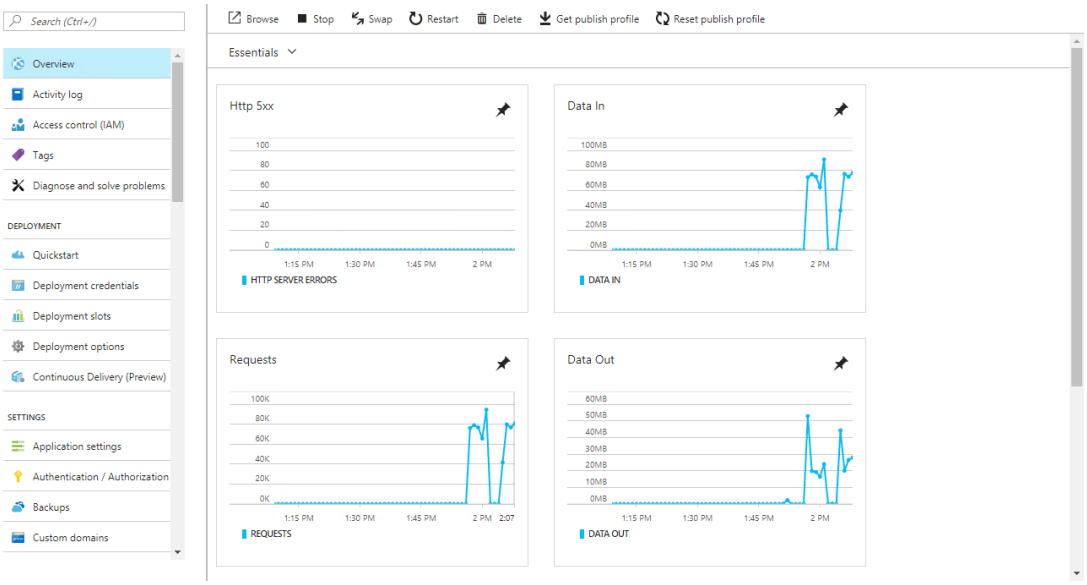


Рисунок 2.11 – Azure App Service інтерфейс [26]

Також однією з основних причин чому була вибрана саме платформа Microsoft Azure, а не її аналоги такі як Amazon Web Service, Digital Ocean і т.д. заключається в тому, що для розробки додатку була вибрана IDE Microsoft Visual Studio, яка в свою чергу містить внутрішні інструменти для розгортання додатку і відповідно дозволяє безкоштовно розгорнути та завантажувати зміни лише декількома операціями.

Висновок до розділу 2

Перед виконанням робіт по розробці веб-орієнтованої системи аналізу показників робототехнічних датчиків в розумному будинку був здійснений детальний аналіз сучасних серверних та клієнтських технологій, враховуючи з цим вище поставлену мету. В результаті для ведення робіт на серверній стороні був вибраний та описаний стек технологій .NET.

Під .NET зазвичай розуміють застосування мов програмування C#, F# та Visual Basic. В даному випадку був вибраний C#, оскільки він є золотою серединою для створення інформаційних додатків різних масштабів навідмінну від .NET мов.

Для забезпечення взаємодії з базою даних будемо використовувати ORM-інструмент Entity Framework Core. За допомогою нього ми зможемо абстрагуватись

від бази даних та її таблиць і здійснювати керування даними незалежно від типу сховища. До того ж, використовуючи підхід по управлінню даними Code First, ми зосередимось безпосередньо на доменній області додатку, тобто спочатку будемо створювати класи бізнес сутностей, і на основі останніх ORM спроєктує нам базу даних. Крім того, даний підхід дозводить легко здійснювати міграцію схеми БД. Для створення бази даних, було прийнято рішення використовувати реляційну систему управління базами даних Microsoft SQL Server, оскільки її досить легко інтегрувати з ORM-інструментами, а також даний тип баз даних дозволить легко класифікувати дані за різними категоріями та ефективно їх зберігати.

Після вибору ORM-інструмента та бази даних, був здійснений аналіз веб-фреймворків платформи .NET та був вибраний ASP.NET Core, оскільки навідмінну від другого за популярністю .NET веб-фреймворка Blazor ми зможемо використовувати нативні інструменти для написання клієнтської логіки. До того ж, враховуючи те, що ASP.NET Core був розроблений за модульним принципом, ми зможемо легко здійснювати масштабування та розгортання веб-додатка в хмарному середовищі.

Для ведення робіт на клієнтській стороні були вибрані нативні інструменти. Для здійснення розмітки використовуватиметься мова розмітки HTML, яка в даний час є найкращим інструментом для розмітки каркасу веб-компонентів. Також випуск версії HTML5 приніс чимало багатих на функціонал нововведень, які будуть використані в процесі розробки. Прикладом можуть атрибути валідації елементів ведення форми, таких як: електронна пошта користувача, телефон тощо. Для опису представлення HTML документа та його елементів, використаємо каскадну таблицю стилів CSS, в результаті це надасть нам такі переваги як: легкість внесення глобальних змін; швидкість написання стилів тощо.

На сьогоднішній час, як правило, для створення адаптивних веб-сторінок використовують різні UI-фреймворки, які значно спрощують та зменшують час розмітки елементів. Для аналізу були відібрані такі фреймворки як Bootstrap та Material UI, і було прийнято рішення використати Bootstrap, оскільки останній навідміну від Material UI – забезпечує послідовний інтерфейс для користувач. До того

ж, Bootstrap надає гнучку сіткову систему для побудову адаптивного вмісту, яка містить шість основних рівнів за замовчуванням, де певний рівень призначений для певної роздільної здатності екрана. Щоб зробити написання власних стилів більш лаконічним та гнучким, ми використаємо препроцесор Less, що призначений для розширення можливостей CSS. Для створення стилів він надає такі зручні функції як: змінні, міксини, та операції, які дозволяють створювати динамічний CSS та зменшують загальний час розробки додатку.

Останніми інструментами клієнтської частини є динамічна мова програмування JavaScript та одна з її бібліотек jQuery. JavaScript використовуватиметься для написання клієнтської логіки маніпуляції HTML елементами, валідації на стороні клієнта, завантаження та обробку серверних даних, створення сповіщень тощо. Враховуючи вище наведені задачі та обсяг необхідної клієнтської логіки, було прийнято рішення використати одну з найпопулярніших JavaScript бібліотеки – jQuery. JQuery інтегрує в веб-додаток, набір функцій, які значно спрощують обсяг нативного js-коду для створення анімації елементів, обробку подій та маніпуляцією гіпертексту.

3 ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Структура системи

Для створення веб-орієнтованої системи аналізу показників робототехнічних датчиків в розумному будинку було розроблено структуру системи, яку наведено в додатку А.

Розглянемо детально кожного з них. Першим незалежним модулем є «Авторизація користувачів». Цей модуль відповідає за автентифікацію користувачів у системі і на основі прав користувача визначає чи достаньо привілегій в останнього для виконання певної операції. Для реєстрації в системі необхідно буде натиснути на відповідну кнопку, що розташована в правому верхньому углу і система перенаправить вас на сторінку з формою заповнення даних. Дана форма потребує від користувача вказати електронну скриньку, прізвище, ім'я та пароль, який необхідно буде ввести двічі, оскільки це зменшить ймовірність забування та відновлення останнього. Необов’язковими полями для введення є мобільний номер користувача, а також пункт «Не виходити», при відмітці якого система рідше буде закривати сесію автоматично. Після заповнення даних, користувач буде проінформований тим, що для активації аккаунта необхідно буде підтвердити свій e-mail, в інакшому випадку він не зможе увійти в системи. Варто зазначити, що система здійснює валідацію введених даних, наприклад: при спробі ввести некоректний формат електронної адреси або номеру телефону, будуть відображені відповідні повідомлення. До того ж, валідація здіснюється як на стороні клієнта та і на серверній, що дозволить унеможливити відправлення некоректних даних. При виконання вище описаних дій користувач автоматично авторизується в системі та отримує унікальний ідентифікатор, що буде збережений в куках браузера. Окрім того, даний модуль містить механізм відновлення та зміну пароля, однак, оскільки паролі перед зберіганням в базу даних завжди хешуються за допомогою алгоритмів дана операцію є складною. Для зміни або відновлення пароля необхідно буде вести свій e-mail на який буде відправлений лист, що міститиме посилання на форму відновлення/зміни паролю, після виконання

даної операції сесія користувача буде скинута та останній перенаправиться на сторінку входу.

Наступним модулем є «Керування користувачами». Даний модуль доступний лише для користувачів з правами адміністратора та дозволяє виконувати операції для перегляду активності останніх з правами користувач, створювати їх та редагувати їхні дані. До того ж, користувачі, чиї акканути були створені адміністраторами, у випадку зміни або скидання пароля, можуть звернутися до останніх з проханням відновити. В даному випадку не потрібно буде підтверджувати дані дії через електронну скриньку, після редагування даних користувачів, останні будуть про це проінформовані через push-сповіщення.

Функціональність для створення та відправки push-сповіщень міститься в модулі «Відправки сповіщень». Перш ніж керувати push-сповіщеннями необхідно встановити зв’язок в реальному часі між клієнтом та сервером. Для цього система автоматично встановлює двонаправлене з’єднання між системою та всіма клієнтами, що дозволить здійснювати обмін інформацією в реальному часі. Під катопом такого зв’язку використовуються ряд механізмів такі як:

- WebSockets;
- Server-Side Events;
- Long-Polling;

В більшості випадках використовуватиметься протокол WebSockets, оскільки на сьогоднішній час він є найкращим варіантом з мінімальною затримкою відправки запитів. Однак для кращої сумісності та стабільності між старими версіями браузера будуть використовуватись також Server-Side Events та Long Polling. Тобто на початку виконання програми буде перевірятись чи підтримується протокол WebSockets, при незадовільному результаті – перевіряється Server-Side Events, якщо результат знову незадовільний то врешті буде застосовуватись Long-Polling оскільки він є найпершим механізмом та відповідно підтримується всіма браузерами. На рис. 3.1 продемонстровано двонаправлений зв’язок.

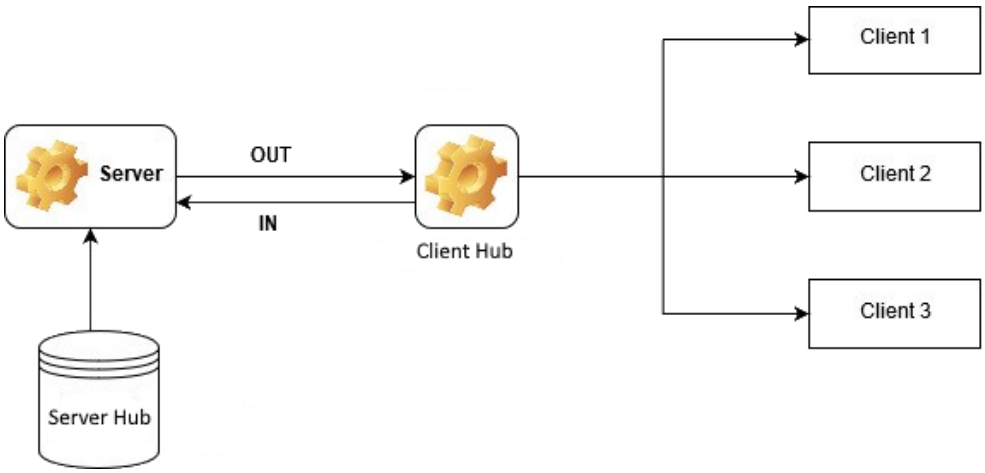


Рисунок 3.1 – Двонаправлений зв’язок

Як тільки з’єднання в реальному часі було встановлено даний модуль має змогу інформувати користувача про запуск, раптову зупинку та результат виконання задач, а також надсилати сповіщення на електронну скриньку при ненормативних показниках датчиків.

Після успішної авторизації в системі користувач перенаправляється на головну сторінку, котра містить панель табів, кожна з яких завантажує та відображує дані отримані з певного датчика в реально часі. Саме за такі дії відповідає модуль «Завантаження показників в реальному часі». Панель складається з трьох табів, а саме: показники температури повітря, вологості повітря та рівня освітлення. Для простоти сприйняття інформації будемо описувати таб датчика температури повітря. Таб розпочинається з показу даних отримані з певного датчика в поточний момент часу, протягом однією години та протягом цілого дня. Отримані дані оновлюються кожні десять тисяч мілісекунд за замовчуванням, якщо є бажання, інтервал оновлення можна зменшити, однак не рекомендується зменшувати менше п’ять тисяч мілісекунд, оскільки це спричинить надмірні запити до бази даних. Після цього таб ділиться на три підтаби: результати теперішнього дня, минулого дня і за тиждень. Почнімо з першого табу. Першим елементом на ньому йде графік, що динамічно демонструє зміну показника температури повітря в залежності від часу в поточний момент часу, де вісь абсцис містить шкалу часу, а вісь ординат шкалу показників температури. Після цього йде стовпчаста діаграма, котра містить середні показники температури отримані в межах однієї години. Діаграма складається з шістьох

стовпців, що знаходяться на вісі абсцис; вісь ординат, як в попередньому графіку, містить шкалу показників температури. Кожен стовпець представляє інтервал часу в межах десять хвилин, наприклад з 16:00 по 16:10, і оновлюється поки поточний час знаходиться в межах його інтервалу. При настанні наступної години система автоматично скине показники діаграми. Останній графік на табі графічно демонструє середні показники температури для кожної години протягом дня. Тобто, як можна здогадатись вісь абсцис складається з двадцяти чотирьох поділок, де кожна з них предсталяє одну годину; а вісь ординат – показники температури. Наступний підтаб відображає загальні результати отримані протягом минулої доби. Він містить мінімальний, максимальний та середній показник температуру за минулу добу та лише одну стовпчасту діаграму, яка відображає середню температуру на кожній годині протягом минулої доби. Останній таб аналогічний попередньому, однак містить інформацію за поточний тиждень, а саме: мінімальний та максимальний показник температури з датою коли був отриманий; загальний середній показник протягом тижня; діаграму, що містить середні результати протягом днів тижня.

Одним з найбільш важливіших модулів є «Виконання складних задач». Сам процес модуля є невидимим для користувачів, оскільки він відповідає за виконання задач, що запускаються та зупиняються на основі метаданих, що були задані користувачем. Створення та управління складними задачами буде описано в наступних розділах. Основними компонентами в даному модулі є scheduler, task-detail та task-trigger. Scheduler виконує збір метаданих складних задач з бази даних, а саме: тип задачі, операції, які необхідно виконати, час запуску та зупинку, та їх основі реєструє окрему задачу в таблиці виконання та ставить її в очікування запуску. Task-detail являється одним з компонентів задачі, який містить її метадані, а також словник значень, що використовується для зберігання простих даних для подальшого їх використання під час наступного запусків. Для збереження складних об’єктів застосовується серіалізація в json рядок. Одним з останніх компонентів задачі є task-trigger, який містить інформацію коли необхідно поставити останню на виконання. Існують різні типи тригерів, найбільш поширеними є простий та крон-тригер. Простий тригер, як правило, застосовується коли необхідно виконати задачу один раз

або задати простий інтервал виконання. В свою чергу крон тригер використовують для задання більш конкретних параметрів коли ту чи іншу задачу необхідно поставити на виконання. Для підвищення продуктивності та більш гнучкого масштабування, виконання складних задач було винесено в окремий фізичний процес. Тобто додаток взаємодіє з сервісом складних задачі і дана взаємодія наведена в додатку В.

На вище наведеному рисунку в ролі клієнта виступає Web, а в ролі сервера – BatchTasks. На сервері інстанс планувальника Scheduler створюється та реєструється як “відкритий об’єкт” використовуючи .NET Remoting. За допомогою цього інстанс залишається в поточному домені, а для його виклику з інших доменів створюються проксі об’єкта, які перехоплюють виклики методів та перенаправляють їх до оригінального інстанса. Простіше кажучи, всі обчислювальні ресурси, які необхідні для виконання задач беруться з процесу BatchTasks, в той час як Web виступає клієнтом, який говорить, що необхідно виконати.

Наступним модулем є «Збереження графіків та діаграм», який відповідає за збереження даних в графічному представленні на персональний комп’ютер. За допомогою нього користувач має змогу завантажити графік або діаграму в популярних графічних форматах таких як: .PNG, .JPEG, SVG vector image. У випадку потреби даних в табличному форматі є можливість завантажити в форматі CSV та XLS. Надається можливість зберегти графік в PDF документі або зразу ж надрукувати через принтер. Додатковими можливостями є перегляд даних безпосередньо на сторінці та в повноекранному режимі. Для виконання вище описаних дій необхідно знайти кнопку, яка має вигляд «бургера», та знаходиться в верхньому правому углу над кожним графіком та діаграмою. Після натискання на неї з’явиться випадаючий список з переліком операцій.

За створення та управління складними задачами відповідає модуль «Керування складними задачами». Він надає користувачу всю необхідну інформацію про задачі в системі, включаючи: кількість задач поставлених на виконання, кількість призупинених, виконаних тощо. До того ж, містить таблицю, яка надає загальну інформацію про кожну з них, включаючи: назву задачі, її тип, стан, дата початку

виконання та останнього запуску, а також перелік додаткових операцій над ними, а саме: перегляд отриманих показників, додаткових подробиць, редагування та видалення. Варто зазначити, що існують три типи задач: задачі для збору показників температури, вологості повітря та освітлення в приміщенні. Окрім того користувач може додати нові задачі, для цього необхідно натиснути на кнопку додати та заповнити наступні поля: назва задачі, дата початку/закінчення виконання та пункт “Поставити на виконання”, що відразу ж після збереження задачі в базі даних зареєструє її в таблиці планувальника. Для зручності роботи користувач може легко переглянути зібрані дані кожної задачі, а також перейти до модуля «Керування зібраними показниками».

Останнім модулем системи є модуль «Керування зібраними показниками». Даний модуль по функціональності схожий на попередній, однак містить декілька основних відмінностей. Головною відмінністю є таблиця показників, оскільки окрім звичайного перегляду даних, дозволяє здійснювати фільтрацію за багатьма критеріями, сортування за зростаннням та спаданням, а також групування. Щоб відсортувати дані за певним стовпцем за зростанням, необхідно лише раз натиснути на відповідну колонку, за спаданням – два рази. По замовчуванню, система автоматично здійснює сортування даних за датою створення. Для групування за певною колонкою необхідно лише перетянути її ім’я на відповідий блок, що знаходиться над таблицею. До того ж, містить фунціонал для створення, редагування, видалення та аналізу показників.

3.2 Структура бази даних

База даних веб-орієнтованої системи аналізу показників робототехнічних датчиів в розумному будинку складається з десяти таблиць, з них дев’ять є головними та одна проміжна.

Перш ніж використовувати базу даних необхідно здійснити міграцію моделей за допомогою спеціальних команд EntityFramework Core. Міграція моделей представляє собою клас в якому здійснюється опис полів сутностей, їхніх типів,

обмеженнями цілісності та зовнішніми ключами. На рис. 3.2 зображений приклад міграції сутності складної задачі:

```
migrBuild.CreateTable(
    name: "Tasks",
    columns: table => new
    {
        Id = table.Column<int>(type: "int", nullable: false)
            .Annotation("SqlServer:Identity", "1, 1"),
        JobType = table.Column<string>(type: "nvarchar(max)", nullable: true),
        JobName = table.Column<string>(type: "nvarchar(max)", nullable: false),
        TriggerName = table.Column<string>(type: "nvarchar(max)", nullable: true),
        GroupName = table.Column<string>(type: "nvarchar(max)", nullable: false),
        TimeInSeconds = table.Column<int>(type: "int", nullable: false),
        IsNotRemovable = table.Column<bool>(type: "bit", nullable: false),
        JobState = table.Column<int>(type: "int", nullable: false),
        JobData = table.Column<string>(type: "nvarchar(max)", nullable: true),
        TriggerData = table.Column<string>(type: "nvarchar(max)", nullable: true),
        StartDate = table.Column<DateTime>(type: "datetime2", nullable: true),
        SecondDate = table.Column<DateTime>(type: "datetime2", nullable: true),
        CreationDate = table.Column<DateTime>(type: "datetime2", nullable: false),
        UserId = table.Column<string>(type: "nvarchar(450)", nullable: true)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_Tasks", x => x.Id);
        table.ForeignKey(
            name: "FK_Tasks_AspNetUsers_UserId",
            column: x => x.UserId,
            principalTable: "AspNetUsers",
            principalColumn: "Id",
            onDelete: ReferentialAction.Restrict);
    });
```

Рисунок 3.2 – Приклад опису міграції сутності Task

Кількість опису міграцій сутностей відповідає кількості таблиць в базі даних. Тобто кількість сутностей дорівнює кількості таблиць. При створенні міграцій важливу роль відіграють зовнішні ключі, тому необхідно спочатку виконувати міграцію таблиць без зовнішніх ключів. В інакшому випадку користувачу виведеться повідомлення про неможливість створення таблиці та міграція моделей, що залишились, не продовжиться. Тому варто уважно продумувати структуру бази даних та послідовність виконання міграцій.

Наступним етапом є заповнення тестовими даними новоствореної бази даних, що надасть нам більш зручне тестування веб-додатку. Для цього ми використаєм механізм сідів. На етапі завершення розробки ми також використаємо даний механізм для очистки системи від тестових даних. Приклад використання механізму сідів зображений на рис. 3.5. Проаналізувавши рисунок можемо сказати, що для створення тестових ролів та користувачів використовуються об’єкти userManager та

roleManager. Перш ніж створювати користувача чи роль виконується перевірка на наявність їх в базі даних, у випадку відсутності останніх – створюються відповідна роль та користувач. Заповнення тестовими даними відбувається окремо за замовчуванням, однак, цей процес можна налаштувати, щоб механізм сідів запускався відразу ж після міграції.

```
public static void Seed(UserManager<User> usrMng, RoleManager<IdentityRole> rlMng)
{
    if (rlMng.FindByName("Адмін") == null)
        rlMng.Create(new IdentityRole("Адмін"));

    var adm = usrMng.FindByEmail(AdminEmail);
    if (adm == null)
    {
        var admUsr = new User
        {
            Email = AdminEmail,
            Name = "Ілля",
            Surname = "Костючик",
            UserName = "Illia",
            EmailConfirmed = true,
            RegistrationDate = DateTime.Now
        };

        usrMng.Create(admUsr, AdmPass);
        usrMng.AddToRole(admUsr, "Адмін");
    }
}
```

Рисунок 3.5 – Приклад використання сідів

В додатку Б зображена структура бази даних. Розгляд структури розпочнемо з таблиці Users, яка містить всю необхідну персональну інформацію про користувачів системи. Вона складається з наступних колонок: ім’я, прізвище, електронна скринька користувача, підтвердження електронної скриньки, дата реєстрації, пароль, телефонний номер, пітвердження телефонного номеру та кількість невдалих спроб входу. Варто зазначити, що дана таблиця не залежить від інших таблиць, тому першою створюється під час міграції, а також всі поля є необхідними для заповнення, окрім телефонного номеру.

Для зберігання додаткових даних про користувача використовуэться таблиця UserClaims. Ці дані, зазвичай, додатково використовуються під час авторизації користувача, наприклад коли ми маємо намір обмежити доступ до ресурсів за віком. Таблиця скадається з ідентифікатора клейма, ідентифікатора користувача, типу та значення клейма. Всі поля обов’язкові для заповнення, а також поле ідентифікатор користувача містить обмеження цілісності зовнішній ключ на таблицю користувачів.

Таблиця UserLogins призначена для зберігання інформації про логіни користувачів використовуючи зовнішні сервіси, такі як: Google, Facebook тощо. Вона містить такі поля, як логін зовнішнього провайдера, ключ провайдера, ім’я провайдера та ідентифікатор користувача, який виступає в ролі зовнішнього ключа на таблицю користувачів, тобто дана таблиця залежить від таблиці користувачів, і тому не може створюватись першою при міграції. Всі поля є необхідними для заповнення.

Таблиця UserTokens містить токени користувачів, що генеруються після успішної автентифікації останніх. Дана таблиця має наступні поля: ідентифікатор користувача, логін провайдера, ім'я та хеш токен користувача, і всі з них є необхідними для заповнення. Як і попередня таблиця – залежить від таблиці користувачів.

Таблиця Roles призначена для зберігання ролів користувачів. Вона складається ідентифікаторі ролі, звичайної та нормалізованої назви, що є необов’язковою для заповнення. Оскільки за концепцією системи користувач може мати багато ролей та навпаки – роль може мати багато користувачів, то дана таблиця повинна бути зв’язана з таблицею користувачів зв’язком багато до багатьох. Але оскільки система управління базами даних фізично не має зв’язка такого типу, то для його реалізації була створена проміжна таблиця UserRoles, яка складається з двох полів: ідентифікатор користувача та ідентифікатор ролі. Дані поля мають обмеження зовнішнього ключа на таблиці користувачів та ролів відповідно. Кожен рядок складається з унікальної комбінації ідентифікаторів, які можуть зіставити одній ролі багато користувачів та навпаки, таким чином ми утворюємо зв’язок багато до багатьох. Всі поля в проміжній таблиця є обов’язковими.

Однією з важливих таблиць є таблиця Tasks, яка призначена для зберігання складних задач. Вона містить наступні поля: ідентифікатор, тип, назва задачі, назва тригера, інтервал виконання, стан задачі, дата створення, дата та час останнього запуску та завершення, дані, що необхідні для виконання задачі, а також ідентифікатор користувача. В даній таблиці необов’язковими полями для заповнення являється дата та час останнього запуску, оскільки задача може бути лише створена та не зареєстрована в планувальнику для виконання. Поле ідентифікатор користувача

має обмеження цілісності зовнішнього користувача на таблицю користувачів, тобто створення таблиці задач здійснюється після таблиці користувачів. Від таблиці задач залежать таблиці, що містять показники різних датчиків, а саме: HumiditiesData, TemperaturesData, LightsData, розглянемо більш детально кожну з них.

Таблиця HumiditiesData використовується для збереження показників отриманих з датчика вологості повітря. Кожен показник складається з таких полів, як: ідентифікатор, рівень вологості, одиниця вимірювання вологості, дата виміру та ідентифікатор складної задачі, яка завантажила даних показник. Всі поля є обов’язковими для заповнення.

За зберігання даних отриманих з датчика температури повітря відповідає таблиця TemperaturesData, яка містить наступні колонки: ідентифікатор показника, рівень температури, одиниця вимірювання температури, дата та час вимірювання і відповідно ідентифікатор задачі, який представляє собою зовнішній ключ на таблицю складних задач. Одиниця виміру температури може приймати значення цельсій або фаренгейт, за замовчуванням – цельсій.

Останньою таблицею в структурі бази даних є таблиця LightsData, яка містить показники, отримані з датчика виміру освітленності. Таблиця складається з наступних колонок, а саме: ідентифікатора, дата виміру, показника освітлення, поле, яке позначає чи був увімкнений світлодіод та, як і у попередніх таблиць, зовнішнього ключа на таблицю складних задач.

3.3 Опис виконання складних задач

Однією з найбільш основних функцій веб-орієнтованої системи аналізу показників робототехнічних датчиків є збір та обробка даних показників, що здійснюється за рахунок створення, планування та виконання складних задач. Структурна схема складних задач наведена в додатку Г.

Розглянемо більш детально кожний компонент системи. Розпочнемо з SchedulerCreator, який використовується для створення інстанса планувальника та задання конфігураційних параметрів. Основними з них є ім’я планувальника, інтервал

перевірки бази даних на наявність задач, які необхідно запустити, кількість потоків доступна для використання, тощо. Оскільки, як було згадано раніше, виконання складних задач було винесено в окремий фізичний процес для кращого масштабування, використовуючи підхід “Remoting client and server”, безпосередньо на сервері створюється інстанс планувальника з параметрами, які роблять його “відкритим об’єктом”, а саме: протокол, порт, назва підключення, і клієнт, в даному випадку веб-додаток, використовує ці параметри для доступу до планувальника та реєстрації задач.

Наступним елементом являється Scheduler він же ж планувальник. Він відповідає за зчитування метаданих про задачі з бази даних, та на основі отриманої інформації створює їхні фізичні об’єкти в таблиці виконання і ставить об’єкт в очікування на запуск. Окрім того, він виступає головним інтерфейсом для збереження нових задач, модифікацію існуючих, видалення, а також містить методи, які примусово зупиняють або запускають задачі. Для того, щоб додатково виконати логіку на ряд подій, таких як: початок та/або закінчення виконання задачі, в планувальнику реєструються об’єкти TriggerListener та TaskListener. Відмінність між ними полягає в ряд методів, наприклад: TriggerListener містить метод, який на основі переданих аргументів, може не запустити задачу на виконання, в той час як TaskListener – метод, що запускається після успішного завершення задачі, наприклад: здійснювати логування даних та результат задачі.

Наступним елементом являється Tasks, який представляє окрему задачу і містить загальну інформацію. Для сховища використовується база даних, оскільки на відміну від оперативної пам’яті, це дасть нам можливість відновити роботу всіх запланованих задач після перезавантаження або аварійного завершення сервера. Основними компонентами задачі є TaskTrigger та TaskDetail.

Компонент TaskTrigger є тригером задачі, який містить необхідні метадані про дату та час запуску, інтервал запуску, тип та ідентифікатор задачі. Він створюється TaskTriggerCreator, який підтримує паттерн “fluent interface”, що дозволяє помістити створення інстанса тригера з усіма параметрами в одну конструкцію. Кожна окрема задача може мати кілька тригерів, наприклад, коли ми бажаємо надсилати сповіщення

для різних часових зон. На рис. 3.7 наведено базовий зразок створення тригера, на ньому можна помітити, що тригер матиме назву SimplTrigger та належитиме до групи Default. Також, використовуючи крон вираз */5 * * * *, ми задаємо інтервал виконання кожні п’ять хвилин.

Компонент TaskDetail містить параметри налаштування для задачі, включаючи її стан, перелік інструкцій, ідентифікатор, а також словник значень, що використовується для зберігання даних задачі для подальшого їх використання під час наступного запусків. Проаналізувавши приклад створення даного компонента на рис. 3.7, можемо сказати, що він створюється за допомогою TaskDetailCreator, при цьому вказуємо тип задачі “MainTask”, ідентифікатор задачі “SimplTask” та група “Default” до якої він належитиме. Також за домопогою метода SetData, ми ініціалізуєм задачу початковими значеннями у вигляді словника.

Варто зауважити, що задачі поділяються на два типи – основні та другорядні. Задачі основного типу призначені для завантаження та аналізу даних в даний момент часу. Окрім цього, на основі отриманих показників, генерують дані для відповідних графіків та діаграм, що були описані в попередніх розділах. За замовчуванням, система містить три задачі даного типу: отримання даних від датчика температури повітря, вологості повітря та рівня освітленості. Варто зазначити, що мінімальний інтервал запуску задачі даного типу становить п’ять тисяч мілісекунд, по замовчуванню – десять тисяч, оскільки це запобігає надмірним запитам до бази даних. Задачі основного типу не можуть бути створені та видалені користувачем з системи.

Другорядний тип задач на відмінну від основного створюється користувачами, дозволяє збирати дані з датчиків в межах зазначений проміжках часу, а також записує зібрані дані в базу даних. Це дозволить переглядати показники за певний проміжок часу та керувати ними в модулі «Керування зібраними показниками». Керування даним типом задач дозволено лише авторизованим користувачам.

Для налаштування тригерів використовуються крон вирази. Крон вираз представляє собой рядок, що складається з семи підвиразів, що відокремлені пробілом та описують окремі деталі розкладу. Підвирази послідовно представляють:

– секунди

- хвилини
- годину
- день місяця
- місяць
- день тижня
- рік (необов’язкове для заповнення)

Розглянемо основні правила створення крон-виразів. Підвирази замість атомарних значень можуть містити діапазони значент. Наприклад, поле дня тижня “wed”, що означає середа, можна замінити на “wed-fri”, що значить від середи по п’ятницю. Символ підстановки “*” використовується для задання будь якого можливого значення в підвиразі, тобто у полі «місяць» символ “*” просто означатиме кожного місяця.

Кожне поле має власний набір дійсних значень, наприклад для секунд це – від нуля до п’ятдесят дев’ять, для годин – від нуля до двадцяти трьох. Для визначення приросту до значент використовується символ “/” Наприклад, якщо у полі “хвилини” поставити “0/15”, це означатиме – кожні 15 хвилин, починаючи з нульової хвилини. Для вказівки “без конкретного значення” застосовується символ підстановки "?", що дозволений лише для полів дня тижня та дня місяця. Це корисно, коли вам потрібно вказати щось в одному з двох полів, але не в іншому [27].

Прикладом cron-виразу є 0 0 12 ? * СР – що означає «кожної середи о 12:00».

3.4 Оптимізація HTTP-запитів

Для оптимізації HTTP-запитів веб-орієнтованої системи аналізу показників робототехнічних датчиків в розумному будинку перш за все були використані техніки бандлінгу та мініфікації. Розглянемо та опишемо детально кожну з них.

Техніка бандлінгу застосовується для зниженням затримки при передачі клієнту за рахунок більш ефективної організації файлів веб-ресурсів, зменшення кількості запитів до сервера та зменшуючи розмір запитуваних ресурсів таких як: стилі, скрипти тощо. Наприклад на рис. 3.8 зображено завантаження ресурсів без

техніки бандлінгу. Для аналізу завантаження ресурсів ми використовуємо Chrome Dev Tools, який являється одним з найкращих інструментів для налагодження, аналізу та тестування веб-сторінок і безпосередньо вбудований в браузер Chrome. За допомогою нього потенційний розробник може переглядати та змінювати DOM елемента без перезавантаження сторінки; наладжувати файли скриптів; інспектувати мережеву активність; оптимізовувати швидкодію веб-додатку тощо.

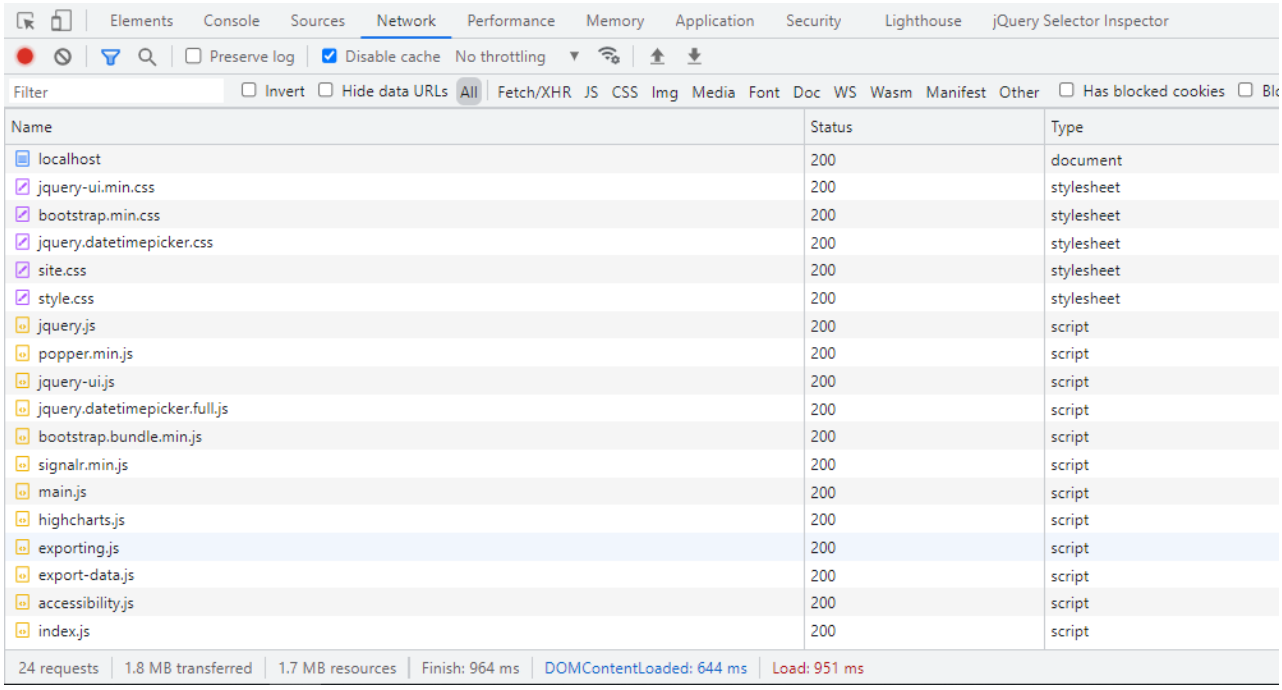


Рисунок 3.8 – Завантаження ресурсів без бандлінгу

На вище наведеному рисунку можна помітити, що при завантажені голової сторінки створюються двадцять чотири запити до сервера для завантаження стилів та скриптів. Всього було завантажено приблизно два мегабайти за дев’ятсот шістдесят чотири секунди.

Тепер створимо окремий бандл, нехай він називатиметься “jquery-all”, який складатиметься з базових файлів бібліотеки для візуалізації віджетів, а саме: jquery.js, popper.min.js, jquery-ui.js та jquery.datetimepicker.full.js.

Результат продемонстрований на рис. 3.9. Проаналізувавши вище наведений рисунок з попереднім, можемо з упевненістю сказати, що кількість запитів запитів зменшилась до двадцяти одного за рахунок бандлу “jquery-all”, який в собі вібрав скрипти з основних файлів бібліотеки jquery: jquery.js, popper.min.js, jquery-ui.js та

jquery.datetimepicker.full.js. Як результат час завантаження ресурсів зменшився до шістсот двадцять одної. Також, беручи до уваги, що був створений лише один бандл, створення інших ще більш оптимізує час завантаження ресурсів. Тепер наведемо опис мініфакації.

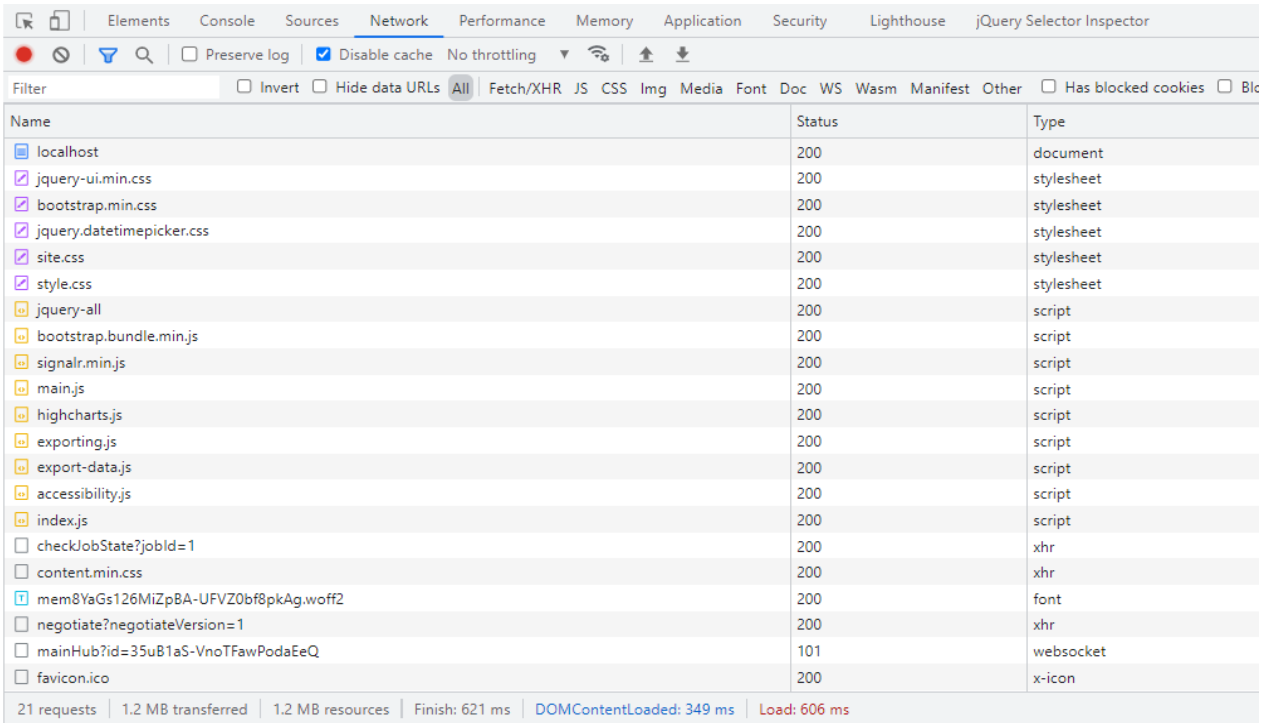


Рисунок 3.9 – Завантаження ресурсів з застосуванням бандлінгу

Одним з головних моментів техніки бандлів є концепція мініфікації, головна суть якої полягає в тому, що при відкритті веб-додатку, клієнту завантажується не повна, а мініфікована версія стилів та скриптів. Це додатково покращує продуктивність, оскільки мініфіковна версія потребує меншого обсягу для передачі, особливо файлів з великим вмістом. До того ж, в режимі налагодження веб-додаток надає звичайну версію, тому що це пришвидшує виявлення та виправлення помилок розробником, а в режимі випуску – мініфіковану.

На рис. 3.10 продемонстрований результат результат мініфікації файлів скриптів та стилів. Як можемо побачити, час завантаження ресурсів зменшився на сімдесят мілісекунд. В масштабованих високо навантажених системах, де файли скриптів та стилі є громіздкими, мініфікація ресурсів може зменшити час завантаження сторінок до кількох секунд.

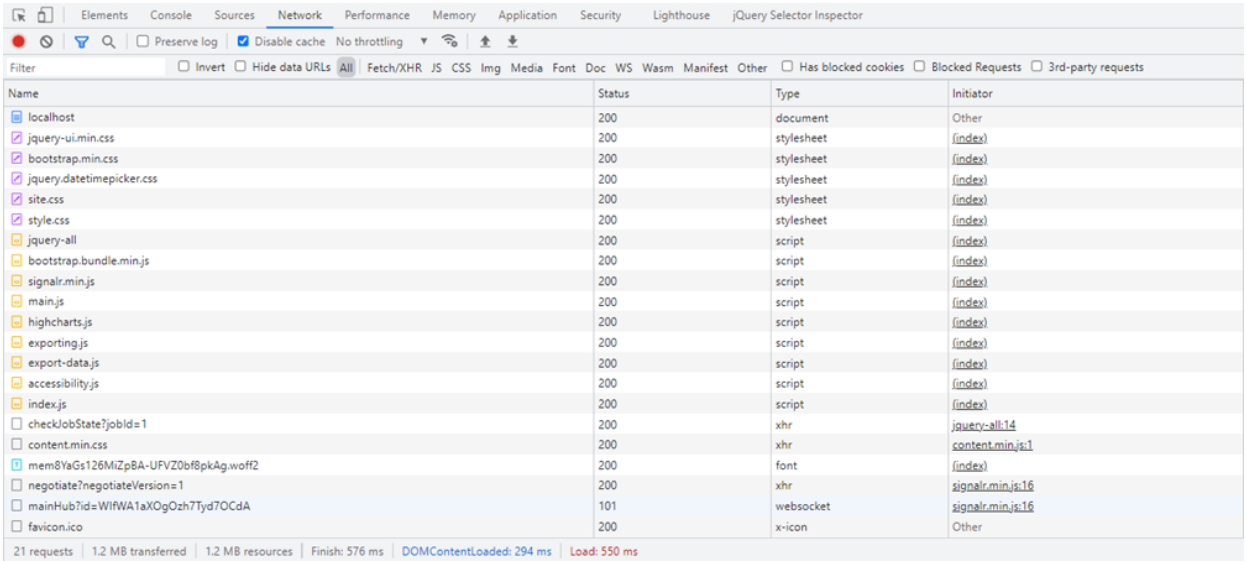


Рисунок 3.10 – Результат мініфакції веб-ресурсів

Таким чином застосування технік бандлінгу та мініфікації дозволяє підвищити ефективність та продуктивність веб-додатків, особливо тих, які мають громіздкі файлів веб-ресурсів

До того ж, разом з вище описаними техніками оптимізації HTTP-запитів використовується кешування веб-ресурсів як на стороні браузера так і на стороні сервера. Блок-схема алгоритму кешування веб-ресурсів наведена в додатку Д.

Висновок до розділу 3

В даному розділі було здійснено опис веб-орієнтованої системи аналізу показників робототехнічних датчиків в розумному будинку. Спочатку була описана структура системи, яка складається з вісім основних модулів, а саме: «Авторизації користувачів», «Керування користувачами», «Відправки сповіщень», «Завантаження показників в реальному часу», «Виконання складних задач», «Збереження діаграм та графіків», «Керування складними задачами» та «Керування показниками». Під час опису структури, було описано принцип взаємодії клієнта та сервера в режимі реального часу, що досягається за рахунок встановлення двонаправленого зв’язку, зокрема з використанням протоколу WebSocket. До того ж, виконання складних задач

було винесено в окремий процес, що дозволить підвищити масштабування додатку та продуктивність.

Далі була розглянута та описана структура бази даних. База даних складається з дев’яти головних таблиць: LightsData, TemperaturesData, HumiditiesData, Tasks, Users, UserClaims UserLogins, UserTokens, Roles та однієї проміжної: UserRoles. Найбільш головними являється таблиця задач – Tasks, таблиця користувачів – Users та таблиця показників LightsData, TemperaturesData, HumiditiesData тощо. До того ж, було описано міграцію моделей за допомогою спеціальних команд EntityFramework Core, що являє собою клас в якому здійснюється опис полів сутностей, їхніх типів, обмеженнями цілісності та зовнішніми ключами. Разом з цим був описаний механізм сідів, що використовується для заповнення бази даних тестовими даними, які будуть використовуватись для тестування додатку. Механізм сідів також використовуватимесь на кінцевому етапі розробки програмного забезпечення для очищення тестових даних.

В третьому підрозділі був описаний один з найбільш основних механізмів системи – виконання складних задач. Складні задачі відповідають за збір, аналіз та збереження показників, що були отримані з датчиків. Вони поділяються на два типи – основні та другорядні. Різниця між ними полягає в тому, що основні – завантажують, оброблюють дані в режимі реального часу, при цьому дані не зберігаються в базу даних, в той час як другорядні – завантажують та зберігають показники в базу даних на зазначених проміжках часу.

В останньому підрозділі описувались техніки оптимізації HTTP-запитів, а саме – бандлінг та мініфікація. Бандлінг призначений для об’єдання декілька файлів скриптів або стилів в один файл, що в результаті дозволить зменшити кількість запитів до сервера, при цьому зменшивши час завантаження веб-сторінок. Гарною практикою являється застосування бандлінгу разом з мініфікаєю. Мініфікація полягає в мінімізації вмісту файлів веб ресурсів, включаючи стисканні імен змінних, видалення коментарів та зайвих пробілів, що зменшить розмір файлу необхідний для завантаження.

4 СТВОРЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Створення моделей бази даних за допомогою ORM

Створена нами структура бази даних, складається з десяти таблиць, з них дев’ять є основними та одна проміжна. Зважаючи на це, нам необхідно буде створити моделі кількістю дев’ять штук для мамінгу на кожну таблицю. Для прикладу створення моделі, візьмемо таблицю показників температури TemperaturesData, оскільки, крім простих полів, містить індекси, обмеження цілісності даних та зовнішній ключ.

Приклад створення моделі зображений на рис. 4.1.

```
[Table("TemperaturesData")]
public class TemperatureData
{
    public int Id { get; set; }
    [Required]
    [Display(Name = "Дата")]
    public DateTime Date { get; set; }

    [ForeignKey(nameof(Task))]
    [Display(Name = "Задача")]
    public int TaskId { get; set; }
    public Task Task { get; set; }

    public string Temp { get; set; }

    [Required]
    [Display(Name = "Температура повітря")]
    public double Temperature { get; set; }

    [Required]
    [Display(Name = "Одиниця вимірювання")]
    public TemperatureUnit TemperatureUnit { get; set; }
}
```

Рисунок 4.1 – Модель показника температури

На вище наведеному рисунку, можна помітити, що модель сутності описується за допомогою звичайного класу, який містить шість властивостей та атрибути аннотації. Розглянемо їх більш детально.

За замовчування, серед всіх членів класу, лише властивості будуть мапитись в таблицю бази даних, тобто звичайні поля ігноруватимуться. Головною умовою мапінгу властивостей є наявність модифікатора public та відкритість сетерра. Також важливу роль відіграють атрибути аннотації, які використовуються для застосування

певних обмежень, вимог до даних та розміщуються поверх властивостей, в даній моделі використовуються такі атрибути як:

– Required – застосовується для позначення поля, яке є обов'язковим для заповнення. Для коректного збереження даних ми використовуємо його для таких властивостей, як: дата отримання, ідентифікатор задачі, значення показника та для одиниці вимірювання;

– Display – використовується для відображення назви поля, що буде зрозумілим користувачу в браузері. Даним атрибутом ми позначаємо ті ж самі властивості, що мають атрибут Required;

– ForeignKey – зазвичай, використовується над зовнішніми ключами для позначення навігаційної властивості. Оскільки модель має лише один зовнішній зв'язок, а саме на модель задач, тому ми ним позначаєм лише зовнішній ключ на модель задач;

На рисунку можна помітити атрибут, що застосовується не до властивостів, а до самого класу, а саме Table. Перш ніж описати його, зазначимо, що ORM Entity Framework Core автоматично генерує назви таблиць за наступною логікою – назва таблиці є назвою множини сутності, наприклад, якщо у нас модель сутності називається Role, ORM згенерує Roles. Однак в поточному випадку генерація назви відбувається некоректно, тому ми використовуємо атрибут Table, для застосування власного коректного імені таблиці – TemperaturesData.

Для описання більш складної логіки зіставлення між моделями та таблицями, а саме створення індексів, складних первинних ключів, зв'язку багато до багатьох, тощо, використаємо підхід вільний програмний інтерфейс додатку (Fluent API) (рис. 4.2). Для застосування даного підходу ми перевизначаємо метод OnModelCreating, який приймає об'єкт типу ModelBuilder, за допомогою якого здійснюється опис. Для сутності показників температури повітря, ми вказуємо ігнорувати властивість Temp, оскільки вона містить тимчасові дані, яких немає сенсу зберігати в базу даних. Також за допомогою метода IndexerProperty, ми створюємо індекс для пришвидшення пошуку за властивостями: Task та Id. Крім того, ми здійснюємо опис зв'язку один до багатьох між сутностями складних задач та показниками температури, тобто одна

задача може мати багато показників температури, а також встановлюємо назву зовнішнього ключа FK_TempData_To_Task. Останньою дією є виклик базової імплементації даного методу, яка створить вище описану логіку для базовий моделей сутностів.

```
protected override void OnModelCreating(ModelBuilder builder)
{
    builder.Entity<TemperatureData>().Ignore(p => p.TempDescription);

    builder.Entity<TemperatureData>()
        .HasOne(p => p.Task)
        .WithMany()
        .HasConstraintName("FK_TempData_To_Task");

    builder.Entity<TemperatureData>().IndexerProperty<Task>("Task");

    builder.Entity<User>().IndexerProperty<int>("Id");

    base.OnModelCreating(builder);
}
```

Рисунок 4.2 – Застосування Fluent API

До того ж, варто зауважити, що за замовчуванням Entity Framework Core створює обмеження цілісності первинний ключ та унікальний ключ для властивості Id. Для тошо, щоб перевизначити дану поведінку, необхідно використати метод

При розробці моделей дуже важливо продумати та подбати про створення індексів. Індекс – це структура, що застосовується до таблиці та представлення, яка підвищує продуктивність пошуку даних. Індекс складається з одного або кількох стовпців таблиці або представлення. Дані ключі утворюють структуру бінарного дерева, що використовується системою управління базами даних для швидкого й ефективного знаходження рядка або рядків, пов’язані із значеннями ключа. SQL Server містить два типи індексів – кластеризований та некластиризований. Різниця між ними полягає в тому, що на пелюстковому рівні кластеризований індекс безпосередньо містить записи, які є впорядкованими, в той час як некластеризований – покажчики на записи в кластеризованому індексі. Тому таблиця може мати один кластеризований індекс та багато некластеризованих. Перш ніж створювати індекси необхідно здійснити аналіз та виділити запити, які найчастіше будуть використовуватись системою.

Одним з найбільш використовуючих записів є пошук користувача за назвою його електронної скриньки, наприклад: при авторизації ми шукаємо користувача за його e-mail, і якщо він знайдений здійснюємо перевірку на збіг паролів або потрібно знайти задачі, що використовуються даним користувачем. До того ж, варто враховувати, що система може мати безліч користувачів і відсутність індексу на полі електронної скриньки може негативно вплинути на продуктивність додатку. Тому доцільно буде додати індекс до поля Email в таблиці користувачів (рис. 4.3).

```
builder.Entity<User>().IndexerProperty<string>("Email");
```

Рисунок 4.3 – Зразок додавання індексу до поля Email

Беручи до уваги, що система може мати лише два типи привілегіїв – адміністратор та користувач, додавання індексів для виконання запитів до таблиці привілегіїв немає сенсу. Враховуючи факт, що кількість зібраних показників для будь з яких датчиків, а саме: температури повітря, вологості повітря, світла, тощо є досить великою, тому реалізація пошуку за певними параметрами на сторінках показників є доцільним. Тому було прийнято створити індекси для полів дата отримання та значення показника в відповідних таблицях, що значно пришвидшить швидкість отримання даних при фільтрації та сортуванні.

Також варто зауважити, що ORM Entity Framework Core, за замовчуванням, створює індекси для поля первинного ключа, а саме – ідентифікатора сутності. Тобто вибірка всіх даних для кожної таблиці являється відразу ж оптимізованою.

4.2 Розробка серверної частини

Для виконання базових операцій в веб-додатку спочатку необхідно пройти реєстрацію або, якщо вже зареєстровані, то ввійти в ситему. За відображення форм заповнення даних, серверну валідацію та подальшу їх обробку відповідає контроллер AccountController (рис 4.4).

```
public class AccountController : Controller
{
    #region Register;
    [HttpGet]
    public IActionResult Registration()
        => View();

    [HttpPost]
    public async Task<IActionResult> Registration(RegisterViewModel model)
    {
        if (ModelState.IsValid)
        {
            var ur = new User
            {
                Email = model.Email,
                UserName = model.Email,
                Name = model.Name,
                Surname = model.Surname,
                RegistrationDate = DateTime.Now
            };
            var res = await _urMng.CreateAsync(ur, model.Password);
            if (res.Succeeded)
            {
                await _urMng.AddToRoleAsync(ur, "Користувач");
                var cd = await _urMng.GenerateEmailConfirmationTokenAsync(ur);
                var clbkUrl = Url.Action(
                    "ConfirmEmail",
                    "Account",
                    new { userId = ur.Id, cd },
                    protocol: HttpContext.Request.Scheme);
                var emServ = new EmailService();
                await emServ.SendEmailAsync(
                    model.Email,
                    "Підтвердження реєстрації",
                    $"Підтвердіть реєстрацію: <a href='{clbkUrl}'>Підтвердити</a>");
                return View("RegisterConfirmation");
            }
            else
                foreach (var er in res.Errors)
                    ModelState.AddModelError(string.Empty, er.Description);
        }
        return View(model);
    }
    #endregion;
```

Рисунок 4.4 – Контроллер авторизації користувачів

Розглянемо логіку реєстрації користувачів. За обробку натискання кнопки Реєстрації відповідає метод Registration з атрибутом HttpGet, який принімає звичайний Get запит та повертає користувачу Razor представлення форми. Після заповнення форми, дані відправляють з Post запитом та передаються у метод Registration. На початку метода здійснюється валідація даних, далі створюємо користувача та зберігаємо в базу даних, якщо збереження завершилось успішно – надаємо йому права користувача. Останнім етапом реєстрації є відправка листа з підтвердженням електронної пошти та інформування користувача про це шляхом відправлення відповідної Razor форми (рис. 4.5).

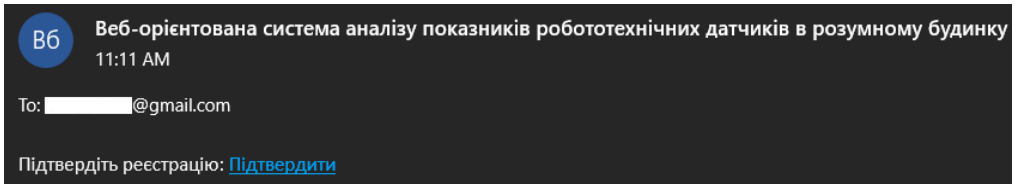


Рисунок 4.5 – Лист підтвердження електронної пошти

Після натискання на Підтвердити буде створений Get запит на метод ConfirmationEmail (рис. 4.6).

```
[HttpGet]
public async Task<IActionResult> ConfirmEmail(string urId, string cd)
{
    if (urId == null || cd == null)
        return View("Error");
    var ur = await _urMng.FindByIdAsync(urId);
    var res = await _urMng.ConfirmEmailAsync(ur, cd);
    return res.Success
        ? RedirectToAction("Login", "Account")
        : View("Error");
}
```

Рисунок 4.6 – Обробка підтвердження реєстрації

Даний метод принімає ідентифікатор користувача та унікальний хеш код, що був згенерований при реєстрації. Далі відбувається пошук користувача та перевірка на збіг хеш кодів. Після успішного завершення користувач перенаправляється на сторінку входу, в інакшому випадку – відображається повідомлення про помилку. Блок-схема алгоритму реєстрації користувачів наведена в додатку Е. Окрім реєстрації та підтвердження email контроллер також містить логіку для входження в систему, зміну пароля тощо.

Після успішної реєстрації користувач перенаправляється на головну сторінку, яка складається з панелі табів, кожна з яких містить аналіз та візуалізацію показників, отримані з відповідних датчиків. В попередніх розділах вказувалось, що саме за цю функціональність відповідають складні задачі основного типу. За цю логіку відповідає метод Execute класа MainTask, що зображений на рис.4.6.

На початку виконання методу здійснюється виклик методу GetDataIndicator, який відповідає за отримання показника датчика в даний момент часу. Далі після отримання показника за допомогою метода SetUpAverageIndicator здійснюється аналіз даних з використанням попередніх показників, в результаті ми отримуємо середнє значення за годину та за добу. Після аналізу викликаються методи SetUpMainPlot, SetUpSecPlot та SetUpThirdPlot, які на основі переданих аргументів, генерують дані для графіків та діаграм, що візуалізують зміну показника в реальному часі, результати в межах поточної години та за добу відповідно. Наприкінці методу

ми отримуємо об’єкт hubContext, який відповідає за спілкування з браузером в реальному часі, та за допомогою його методів надсилаєм всім підключеним користувачам необхідні дані, які будуть використані для оновлення попередніх та графіків (рис. 4.7).

```
public async Task Execute(ITaskExecutionContext context)
{
    using (var scope = _serviceScopeFactory.CreateScope())
    {
        var dataIndicator = GetDataIndicator();
        var taskData = context.TaskDetail.TaskDataMap;
        if (taskData.Keys.Contains("CurrentLevel"))
            taskData["CurrentLevel"] = dataIndicator.Level;
        else
            taskData.Add("CurrentLevel", dataIndicator.Level);
        // Set up an average indicator of data
        var avgDataForHour = SetUpAverageIndicator(taskData, UnitOfTime.Hour, 1, dataIndicator.Level);
        var avgDataForDay = SetUpAverageIndicator(taskData, UnitOfTime.Day, 1, dataIndicator.Level);
        var timeInSeconds = (int)context.Trigger.TaskDataMap["TimeInSeconds"];
        if (!_curTimeInSeconds.HasValue)
            _curTimeInSeconds = timeInSeconds;
        bool isTriggerChanged = false;
        if (_curTimeInSeconds != timeInSeconds)
        {
            _curTimeInSeconds = timeInSeconds;
            isTriggerChanged = true;
        }
        // Set up data for plots
        var mainPlotData = SetUpMainPlot(taskData, dataIndicator, 10, timeInSeconds, isTriggerChanged);
        var secChartData = SetUpSecPlot(taskData, dataIndicator, 60, 10);
        var thirdChartData = SetUpThirdPlot(taskData, dataIndicator, 24, 1);
        var hubContext = scope.ServiceProvider.GetRequiredService<IHubContext<MainHub>>();
        await hubContext.Clients.All.SendAsync("ReceiveLightData",
            dataIndicator,
            avgDataForHour,
            avgDataForDay,
            mainPlotData,
            secChartData,
            thirdChartData);
    }
}
```

Рисунок 4.7 – Логіка виконання складної задачі

Для управління складними задачами використовується контроллер ManagementController, який знаходиться на рис. 4.8. При переході на сторінку управління викликається метод Index, який спочатку перевіряє привілегії поточного користувача, і якщо користувач має привілегії адміністратора то з бази даних витягаються всі наявні складні задачі, в інакшому випадку лише ті задачі які належать до звичайного користувача. Далі на основі отриманих даних, ми обчислюємо кількість усіх наявних задач, ті які запущені, зупинені та повністю завершені. Наприкінці повертаємо Razor представлення, яке буде перетворене в звичайну HTML сторінку з даними та таблицею задач.

```
public class ManagementController : Controller
{
    #region CRUD
    public async Task<IActionResult> Index()
    {
        var role = User.Claims.FirstOrDefault(c => c.Type == ClaimsIdentity.DefaultRoleClaimType).Value;

        IQueryable<Task> tasks = _context.Tasks.Include(j => j.User);
        if (role == "Користувач")
        {
            var userId = _urMngr.GetUserId(User);
            tasks = tasks.Where(j => j.UserId == userId);
        }
        var tasksList = await tasks.ToListAsync();

        ViewBag.AllTasksCount = tasks.Count();
        ViewBag.RunningTasksCount = tasks.Where(j => j.TaskState == TaskState.Running).Count();
        ViewBag.StoppedTasksCount = tasks.Where(j => j.TaskState == TaskState.Stopped).Count();
        ViewBag.CompletedTasksCount = tasks.Where(j => j.TaskState == TaskState.End).Count();
        return View(tasksList);
    }

    public async Task<IActionResult> Details(int? id)
    {
        if (id == null)
            return NotFound();
        var task = await _context.Tasks
            .Include(j => j.User)
            .FirstOrDefaultAsync(m => m.Id == id);
        if (task == null)
            return NotFound();
        var indicators = await _context.TemprsData.Where(l => l.TaskId == task.Id).ToListAsync();
        ViewBag.CountOfIndicators = indicators.Count;
        ViewBag.AvgLevel = indicators.Average(i => i.Level).ToString("0.00");
        var avgLevelList = indicators.GroupBy(l => l.Date.Date, (key, data) => data.Average(d => d.Level));
        ViewBag.AvgLevelPerDay = avgLevelList.Average().ToString("0.00");
        return View(task);
    }
```

Рисунок 4.8 – Контроллер керування складними задачами

Окрім цього контроллер також містить такі методи як: Create, Edit, Details та Delete. За створення окремої задачі відповідає метод Create. Він приймає Post запит з наступними даними: назва, група, інформацію про дату та час початку виконання задачі, прапорець, що вказує чи повинна задача бути запущена в даний момент. Після отримання даних відбувається їхня валідація та попередня обробка, в разі успішного завершення відбувається запис новоствореної задачі в базу даних і відправка клієнту відповідь про успішне створення з HTTP кодом 200 та перенаправлення на сторінку управління. Якщо валідація завершилась некоректно, то користувачу надсилаються повідомлення з помилками. Окрім того він здійснює валідацію за допомогою anti-forgery token для протидії підробці міжсайтових запитів.

Для редагування існуючих складних задач використовується метод Edit. Як і метод Create приймає Post запит та перед виконанням здійснює валідацію запиту за допомогою anti-forgery token для протидії підробці міжсайтових запитів. Стосовно аргументів приймає ідентифікатор задачі, що редагується та список властивотей з оновленими даними. Далі здійснюється валідація отриманих даних, якщо результат негативний – клієнту повертається список всіх помилок щодо введених ним даних.

Якщо результат валідації позитивний, то створюється запит на пошук задачі за її ідентифікатором. Після успішного пошуку відбувається оновлення необхідних параметрів та збереження оновленої задачі до бази даних. Наприкінці клієнту надсилається відповідь з HTTP кодом 200 про успішне редагування та перенаправлення на головну сторінку. Всі ці дії відбуваються в межах однієї транзакції для збереження цілісності даних.

Метод Details призначений для відображення детальної інформації про окрему задачу. Він принімає ідентифікатор задачі та здійснює її пошук в базі даних. Якщо задача існує то відбувається пошук всіх показників, що були отримані нею та обчислюється їх кількість, загальне середнє значення та середнє значення протягом в залежності від дати отримання. У випадку якщо задачі з даним ідентифікатором не знайдено, то клієнту відправляється відповідь з відповідним статусом, що задачі не існує.

За видалення задачі відповідає метод Delete. Як і попередній метод принімає унікальний ідентифікатор, за яким відбуваєть пошук на наявність задачі в базі даних. Якщо задача існує то клієнту відправляється форма з підтвердженням видалення, після натискання кнопки підтвердити, то задача видаляється з бази даних і клієнту надсилається повідомлення про успішне видалення. Якщо задача з переданим ідентифікатором не була знайдена, то надсилається відповідне повідомлення.

Подібним чином реалізовані й інші контролери для управління показниками, користувачами тощо. Вони базуються на чотирьох вище описаних методах та доповнені додатковими методами для реалізації специфічної функціональності.

4.3 Розробка клієнтської частини

Після створення серверної логіки основних компонентів, відбувається розробка їхньої клієнтської логіки. Візьмемо для прикладу створення форми реєстрації користувачів (рис. 4.9).

```
<div class="container md-5">
  <div class="rw">
      <form method="post" controller="Account" action="Register">
        <h4>Введіть, будь ласка, реєстраційні дані</h4>
        <hr />
        <div validation="All" class="text-red"></div>
        <div class="form-group">
          <label for="Email"></label><sup> *</sup>
          <input for="Email" class="form-control" placeholder="E-mail..." />
        </div>
        <div class="form-group">
          <label for="Name"></label><sup> *</sup>
          <input for="Name" class="form-control" placeholder="Ім'я..." />
        </div>
        <div class="form-group">
          <label for="Surname"></label><sup> *</sup>
          <input for="Surname" class="form-control" placeholder="Прізвище..." />
        </div>
        <div class="form-group">
          <label>Введіть свій номер телефону:</label>
          <input class="form-control" placeholder="Номер телефону..." />
        </div>
        <div class="form-group">
          <label for="Password"></label><sup> *</sup>
          <input for="Password" class="form-control" placeholder="Пароль..." />
        </div>
        <div class="form-group">
          <label for="PasswordConfirm"></label><sup> *</sup>
          <input for="PasswordConfirm" class="form-control" placeholder="Повторіть пароль..." />
        </div>
        <button type="submit" class="btn btn-primary">Зареєструватись</button>
      </form>
  </div>
</div>
```

Рисунок 4.9 – Клієнтська логіка реалізації реєстраційної форми

На вище наведеному рисунку зображений фрагмент Razor предствлення – файл з розширенням .cshtml, який навідміну від звичайного html документа, що складається лише з html тегів, він також містить синтаксис Razor – спеціальні конструкції, які дозволяють переходити від коду html до c#. Представлення може містити всі стандартні елементи розмітки html, а також підключати стилі та скрипти. Під час виконання вони компілюються у збірки і при нагоді використовуються для генерації html-сторінок, які у випадку відповіді HTTP запиту відправляються клієнту у вигляді звичайної розмітки html-сторінки.

Проаналізувавши створення форми, можемо помітити, що спочатку йде створення bootstrap контейнера за допомогою класу .container, використання якого дозволить створити адаптивний контент. Кожен контейнер складається з набору рядків, які мають клас .row, кожен рядок, в свою чергу, складається з набору колонок, які призначені для зберігання певного фрагменту даних, в поточному випадку це йде форма реєстрації. На тегу форми можна помітити атрибут method, який позначає тип запиту, що буде використовуватись при відправці даних, а саме – post. Окрім цього тег також містить атрибути, що починаються з префікса asp-, які ще називаються тег-хелперами та використовуються для спрощення генерації html контенту та атрибутів.

Наприклад в тегу asp-controller замість вказування абсолютного шляху до контроллера ми вказуємо лише ім’я.

Власне форма починається з блоку, який містить тег-хелпер asp-validation-summary=”All”, та призначений для відображення помилок валідації даних. Далі йдуть шість блоків, а саме: блок для введення електронної адреси, прізвища, імені, номера телефону, пароля та підтвердження пароля. Кожен блок складається з тегу label, який вказує параметр для введення та input, що є полем для введення. Останнім елементом є кнопка з типом submit, що реєструє її як елемент відправки даних. На рис. 4.10 продемонстрований результат генерації вище описаної форми.

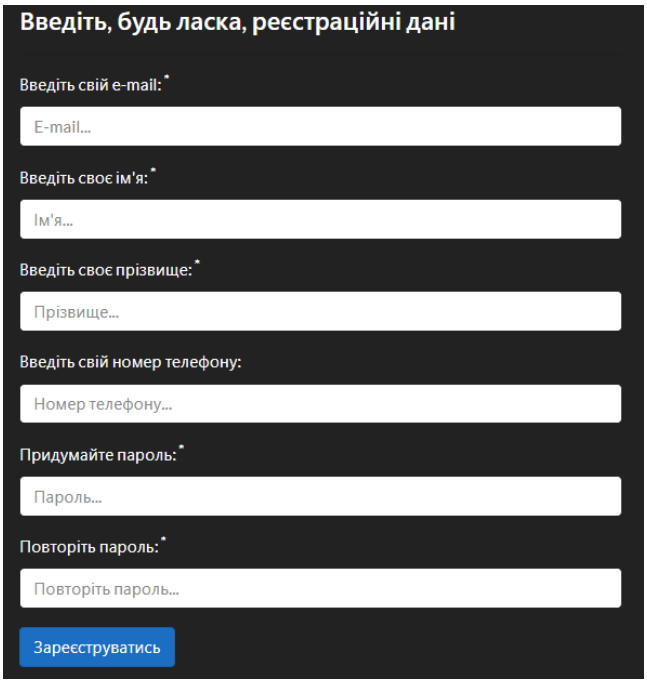


Рисунок 4.10 – Форма реєстрації користувачів

На рисунку можна помітити символ “*”, який вказує, що поле є обов’язковим для заповнення.

Кожне представлення в проєкті містить загальні компоненти, що присутні в інших представлень, наприклад, це може бути шапка веб-додатку, футер, тощо. Тому набагато доцільніше буде створити спільний шаблон, в результаті чого при зміні спільних елементів достатньо буде лише внести зміни в шаблон, не змінюючи при цьому кожне представлення. Такі шаблони також називаються майстер сторінками.

Майстер сторінки, як правило, використовуються для відтворення уніфікованого виду сайту. Тобто, це ті ж самі представлення, які містять в собі інші представлення. Власне, вони дозволяють значно зменшити час розробника, оскільки, наприклад: можна описати спільну логіку панелі вкладок для всіх представлень, а також підключити загальні скрипти та стилі, в результаті чого не доведеться на окремому представленні прописувати шлях до файла, а потім при необхідності його змінювати. Вставлення інших представлень в шаблон відбувається за допомогою спеціальних тегів. Приклад створення майстер сторінки зображений на рис. 4.11.

```
<header>
  <nav class="nav nav-exp-sm nav-toggle-sm nav-dark bg-main">
    <div class="container">
      <a class="navbar-br">SmartWeb</a>
      <button class="navbar-toggler" type="button" data-toggle="collapse" data-target=".navbar-collapse" aria-controls="navbarSupportedContent"
              aria-lbl="Togle nav">
        <span class="navbar-toggler-icon"></span>
      </button>
      <div class="nav-collapse collapse d-sm-inline-flex flex-sm-row-reverse">
        <partial name="_LoginPartial" />
        <ul class="nav-nav flex-grow-1">
          <li class="nav-item">
            <a class="nav-link" controller="Home" action="Index">Головна сторінка</a>
          </li>
          @if (Usr.IsLogined)
          {
            <li class="nav-item dropdown">
              <a class="nav-link dropdown-toggle" data-toggle="dropdown" id="indicators">Управління</a>
              <div class="dropdown-menu" area-labelledby="themes">
                <a class="dropdown-item" controller="Management" action="Index">Задачами</a>
                <a class="dropdown-item" controller="LightDatas" action="Index">Показниками</a>
                @if (User.IsInRole("Адміністратор"))
                {
                  <a class="dropdown-item" controller="Users" action="Index">Користувачами</a>
                }
              </div>
            </li>
          }
        </ul>
      </div>
    </div>
  </nav>
</header>
```

Рисунок 4.11 – Створення шапки проєкту

Проаналізувавши вище наведений рисунок, можемо сказати, що шапка містить тег nav з ключовими класами navbar та navbar-expand-sm, які роблять шапку адаптивною до зміни ширини екрану. Шапка починається з елементу, що містить ім’я проєкту, при натисканні якого – перенаправляє користувача на головну сторінку. Далі йде елемент з прямим посиланням на головну сторінку. Наступним у нас є випадаючий список, що відображається у випадку авторизації користувача. Список містить посилання на управління складними задачами, показниками та користувачами. Останнє посилання доступне лише для користувачів з привілегіями адміністратор. Останнім елементом є тег partial, який додатково підключає посилання на персональний кабінет та вихід з системи для авторизованих користувачів, для

неавторизованих – посилання на вхід та реєстрацію в системі. Результат створення шапки зображений на рис. 4.12.

Рисунок 4.12 – Шапка веб-додатку

Можна помітити, що ми авторизувались в системі з привілегіями адміністратор, оскільки є посилання на управління користувачами.

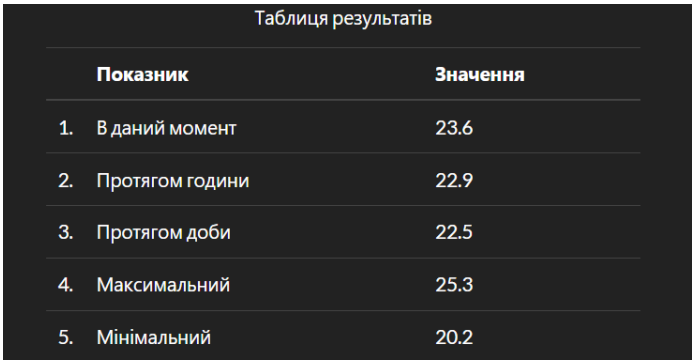
Тепер опишемо розробку клієнтської частини головної сторінки. Головна сторінка складається з трьох табів кожен з яких містить аналіз та візуалізацію показників з датчиків температури повітря, вологості повітря та освітленності. Оскільки кожен з табів містить однакові клієнтські компоненти, тому розглянемо лише таб температури повітря. Власне, таб починається з таблиці, код якої зображений на рис. 4.13.

```
<p class="col-12 text-center">
    Таблиця результатів
</p>
<table class="table w-50">
  <thead>
    <tr>
      <th style="width: 5%"></th>
      <th>Показник</th>
      <th>Значення</th>
    </tr>
  </thead>
  <tbody>
    <tr>
        <td>1.</td><td>В даний момент</td><td id="curLevel">0</td>
    </tr>
    <tr>
        <td>2.</td><td>Протягом години</td><td id="avgLevelForHour">0</td>
    </tr>
    <tr>
        <td>3.</td><td>Протягом доби</td><td id="avgLevelForDay">0</td>
    </tr>
    <tr>
        <td>4.</td><td>Максимальний</td><td id="maxLevelForDay">0</td>
    </tr>
    <tr>
        <td>3.</td><td>Мінімальний</td><td id="minLevelForDay">0</td>
    </tr>
  </tbody>
</table>
```

Рисунок 4.13 – Створення таблиці аналізу показників

Власне, перед таблицею розташований її заголовок. Таблиця складається з п’яти рядків та трьох стовпців. Перший рядок містить значення показника в

теперішній момент часу, другий – середнє значення протягом години, третій – середнє значення протягом доби, четвертий – максимальне значення протягом доби та п'ятий – мінімальне значення протягом доби. Результат створення зображений на рис. 4.14.



Таблиця результатів

	Показник	Значення
1.	В даний момент	23.6
2.	Протягом години	22.9
3.	Протягом доби	22.5
4.	Максимальний	25.3
5.	Мінімальний	20.2

Рисунок 4.14 – Таблиця аналізу показників

Проаналізувавши результати, можемо сказати, що значення температури в даний момент дорівнює 23.6, протягом години – 22.9, протягом доби – 22.5, максимальне за добу – 25.3, мінімальне за добу – 20.2.

Після таблиці результатів йде розділення таба на три підтаба. Кожен з них містить графіки та діаграми візуалізації відповідних даних. Розглянемо логіку візуалізації графіку зміни показника в реальному часі (рис. 4.15).

```
function setUpMainPlot(plotData) {
    let mainChart = null;
    const seriesData = plotData.seriesData.map((data, i) => {
        return { x: i, y: data.level === undefined ? data.Level : data.level};
    });
    const xCategories = plotData.xCategories.map(dateStr => new Date(dateStr).toLocaleTimeString());
    const levels = seriesData.map(d => d.y);
    let maxSeriesData = Math.max(...levels);
    mainChart = Highcharts.chart('mainChart', {
        chart: {
            type: 'line'
        },
        title: {
            text: plotData.plotName
        },
        xAxis: {
            categories: xCategories,
            min: 0,
            max: xCategories.length - 1
        },
        yAxis: {
            title: { text: 'Значення показника' },
            min: 0,
            max: maxSeriesData
        },
        series: [{
            name: 'Зміна показника в реальному часу',
            data: seriesData
        }]
    });
}
```

Рисунок 4.15 – Логіка візуалізації показників в реальному часі

Логіка візуалізації показників в реальному часі знаходиться в методі setUpMainPlot, який принімає об’єкт типу PlotData. Даним об'єкт містить дві властивості: xCategories – масив значень, що будуть відображені на вісі абсцис, seriesData – масив значень вісі ординат, що мапиться в масив об’єктів типу Point, які в результаті будуть додані на графік. Перед створенням графіку обчислюється максимальне значення вісі ординат.

Власне за побудову графіків та діаграм відповідає об’єкт highcharts, який в залежності від переданих параметрів здійснює відповідну візуалізації. Мінімально необхідними параметрами є: селектор тегу, в якому буде розташований графік (‘mainChart’); тип діаграми – лінійна чи стовпчаста (‘line’); назва діаграми (‘Зміна показника’); мінімальне, максимальне та масив значень для вісі абсцис; мінімальне, максимальне значення для вісі ординат, а також її назва (‘Значення показника’); останнім необхідним параметром є масив об’єктів, що містить назву графіку та масив об’єктів типу Point. Результат створення зображений на рис. 4.16.

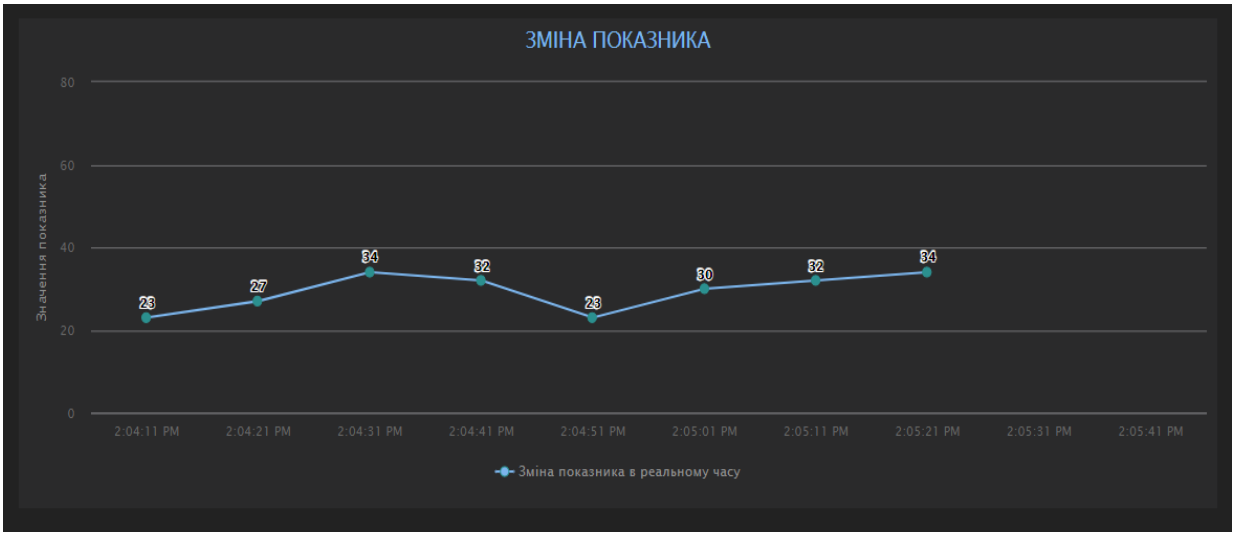


Рис. 4.17 – Візуалізація зміни показника температури

На вище наведеному графіку, можемо сказати, що графік складається з восьми точок, тобто було зібрано вісім показників температури повітря за вісімдесят секунд. Максимальна можлива кількість точок – десять, при цьому інтервал між поділками

вісь абсцис – десять секунд. У випадку перевищенні останньої точки – вісь абсцис матиме оновленні дані, а графік міститиме одну точку.

Логіка візуалізації стовпчастої діаграми подібна до логіки візуалізації звичайного графіку, однак для параметра типу діаграми необхідно передати ‘chart’ замість ‘line’. Поглянемо на стовпчасту діаграму середніх показників за одну годину (рис. 4.18).

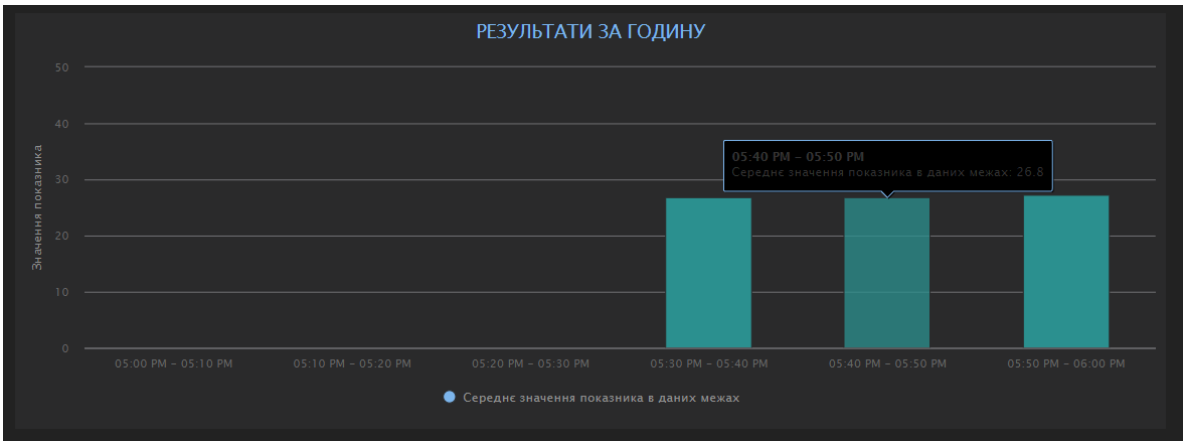


Рисунок 4.18 – Діаграма середніх показників температури за годину

Здійснивши аналіз діаграми, можемо помітити, що діаграми містить три останні стовпця кожен з яких візуалізує середнє значення в межах десяти хвилин. Перші три стовпця є пустими, що значить, що сервер додатку був запущений в другій половині години. При бажанні дізнатись точне середнє значення, необхідно навести курсор на відповідний стовпець. Наприклад, при наведенні на другий стовпець середнє значення температури дорівнює 26.8.

Далі опишемо логіку розробки управління складними задачами. Кожна сторінка управління, в тому й числі показниками та користувачами, складається з двох основних блоків.

Перший блок розпочинається з відповідного заголовку. Далі йдуть посилання на форму створення окремої задачі та зручний перехід до управління показниками. Після цього розташовується таблиця, яка надає швидку інформацію про загальну кількість задач, кількість задач що виконуються, виконаних та кількість задач що ні разу виконувались (рис. 4.19).

```
<div id="fst-clock" class="d-flex flex-column">
    <h2 class="text-center mt-5">Управління задачами</h2>
    <div class="col-12">
      <a asp-action="Create" class="mt-2">Створити</a>|<a asp-controller="Datas" class="mt-1">Перейти до показників</a>
    </div>
    <div class="col-12">
      <table class="table text-center" style="margin: 25px auto 0; max-width: 500px">
          <tr>
            <th>Кількість</th><th>Значення</th>
          </tr>
          <tr>
            <td>Загальна</td><td>@ViewBag.AllTasksCount</td>
          </tr>
          <tr>
            <td>Виконаних</td><td>@ViewBag.CompletedTasksCount</td>
          </tr>
          <tr>
            <td>Зупинених</td><td>@ViewBag.StoppedTasksCount</td>
          </tr>
          <tr>
            <td>Нових</td><td>@ViewBag.CompletedTasksCount</td>
          </tr>
      </table>
    </div>
</div>
```

Рисунок 4.19 – Логіка створення першого блоку сторінки управління

Варто зауважити декілька, моментів. Блок складається з контейнера з ідентифікатором “fst-block” та класами, що вирівнюють весь вміст по центру. Значення для таблиці знаходяться в об’єкті ViewBag, що використовується для передачі даних між контроллером та представленням. Вигляд першого блоку зображений на рис. 4.20.

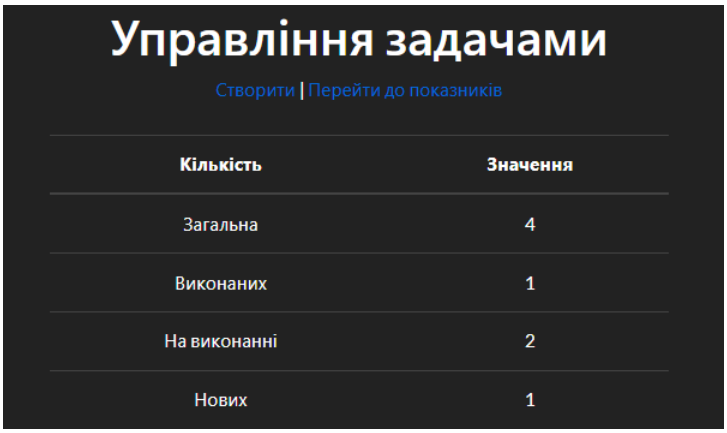


Рисунок 4.20 – Перший блок сторінки управління

Проаналізувавши таблицю, можемо сказати, що система містить лише чотири складні задачі, з яких виконаних лише одна, ті, що виконуються – дві, та одна нова – тобто ще ні разу не запускалась.

Другий блок складається з таблиці, яка містить перелік всіх задач з головною інформації про кожну з них (рис. 4.21). Як і будь яка інша таблиця вона

розпочинається з тега caption, що містить заголовок таблиці. Далі йде побулова шапки таблиці, для цього ми використовуємо html хелпер DisplayNameFor, який повертає локалізовану назву колонки. Після шапки йде побудова тіла таблиці, для цього ми використовуємо цикл foreach, який перебирає модель типу колекції складних задач та почергово створює рядок з відповідними значеннями властивості. Для отримання відповідного значення використовується html хелпер DisplayFor, який навідміну від простого виводу значення, здійснює форматування дат та чисел в залежності регіональних налаштувань на комп'ютері користувача. До того ж, можна помітити умовні конструкції, наприклад: одна з них перевіряє чи має задача задані межі виконання, якщо так – то відбувається їх показ користувачу, в інакшому випадку – відображається символ нескінченності. Також в кінці кожного рядка створюються посилання на операції над задачами: редагування, перегляд детальної інформації, видалення, показ показників, що були зібрані ними.

```
<table class="table table-striped table-hover text-center" style="max-width: 1250px">
    <caption style="caption-side: top; font-size: 24px; text-align: center">Таблиця задач</caption>
    <tr>
        <th>@Html.DisplayNameFor(model => model.TaskName)</th>
        <th>@Html.DisplayNameFor(model => model.TaskState)</th>
        <th>Дата та час виконання</th>
        <th>@Html.DisplayNameFor(model => model.TimeInSeconds)</th>
        <th colspan="4"></th>
    </tr>
    @foreach (var task in Model)
    {
        <tr>
            <td>@Html.DisplayFor(modelItem => task.JobName)</td>
            <td>@Html.DisplayFor(modelItem => task.JobState)</td>
            <td>
                @if (task.StartDate.HasValue)
                {
                    @Html.DisplayFor(modelItem => task.StartDate)
                    @: -
                    @Html.DisplayFor(modelItem => task.SecondDate)
                }
                else
                    @:&#8734;
            </td>
            <td>@Html.DisplayFor(modelItem => task.TimeInSeconds)</td>
            <td><a asp-action="Edit" asp-route-id="@task.Id">Редагувати</a></td>
            <td><a asp-action="Details" asp-route-id="@task.Id">Деталі</a></td>
            <td><a asp-action="Delete" asp-route-id="@task.Id" class="@(task.IsNotRemovable ? "disabled" : "")">Видалити</a></td>
            @if (item.TriggerName != "MainTriggerData")
                <td><a asp-action="Index" asp-controller="Datas" asp-route-processId="@task.Id">Зібрані показники</a></td>
        </tr>
    }
</table>
```

Рисунок 4.21 – Логіка створення другого блоку сторінки управління

Результат побудови другого блоку сторінки для управління задачами зображений на рис. 4.22.

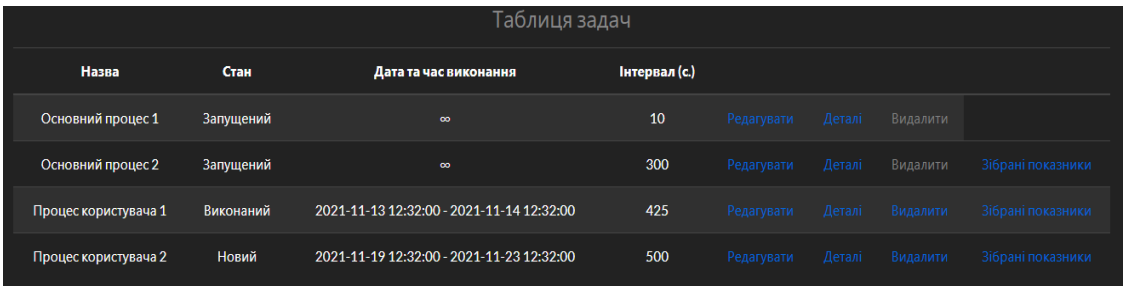


Рисунок 4.22 – Другий блок сторінки управління

Дивлячись на таблицю, бачимо, що всього в системі є чотири задачі для збіру показників температури. Перші дві є системними задачами, тобто у них час виконання необмежений та їх неможливо видалити. Останні задачі були створені безпосередньо користувачем, одна з них вже завершилась, друга – ні разу не запускалась.

Тепер опишемо логіку редагування окремої задачі (рис. 4.23). Варто зазначити, що подібним чином реалізована й логіка редагування, створення та видалення задач, показників, користувачів, тощо.

```
<div class="container mt-5">
  <h4>Редагування процесу</h4>
  <form asp-action="Edit">
    <div asp-validation-summary="ModelOnly" class="text-danger"></div>
    <input type="hidden" asp-for="Id" />
    <div class="form-group">
      <label asp-for="TaskName" class="control-label"></label>
      <input asp-for="TaskName" class="form-control" />
    </div>
    <div class="form-group">
      <label asp-for="GroupName" class="control-label"></label>
      <input asp-for="GroupName" class="form-control" />
    </div>
    <div class="form-group">
      <label asp-for="TaskType" class="control-label">Тип даних</label>
      <select asp-for="TaskType" class="custom-select">
        <option selected>Температура</option>
        <option>Вологість повітря</option>
        <option>Освітлення</option>
      </select>
    </div>
    <div class="form-group">
      <label asp-for="StartDate" class="control-label"></label>
      @if (Model.TaskState == TaskState.End)
        <input asp-for="StartDate" class="form-control date-picker" disabled />
      else
        <input asp-for="StartDate" class="form-control date-picker" />
    </div>
    <div class="form-group">
      <label asp-for="SecondDate" class="control-label"></label>
      @if (Model.TaskState == TaskState.End)
        <input asp-for="SecondDate" class="form-control date-picker" disabled />
      else
        <input asp-for="SecondDate" class="form-control date-picker" />
    </div>
    <div class="form-group">
      <label asp-for="TimeInSeconds" class="control-label"></label>
      @if (Model.TaskState == TaskState.End)
        <select asp-for="TimeInSeconds" class="form-control" asp-items="ViewBag.SecondTaskSelectListItem" disabled></select>
      else
        <select asp-for="TimeInSeconds" class="form-control" asp-items="ViewBag.SecondTaskSelectListItem"></select>
    </div>
    <input type="submit" value="Зберегти" class="btn btn-primary" />
  </form>
  <a asp-action="Index">Повернутися</a>
</div>
```

Рисунок 4.23 – Логіка редагування задачі

Перед редагування параметрів задачі знаходиться її заголовок, що містить назву операції. Власне, наступним елементом йде форма, при заповненні якої та відправці даних створюється post запит, який відправить результати форми на сервер для подальшої обробки. До того ж, всі елементи форми є адаптивними до змін, тобто кінцевому користувачу буде легко редагувати як на персональному комп'ютера так і на мобільних пристроях чи планшетах. Першим елентом форми є контейнер, який відображає помилки при невдалій валідації даних. Наступним йде прихований елемент, що містить ідентифікатор задачі, який використовуватиметься відповідним методом контроллера для пошуку та оновлення об'єкта. Далі, безпосередньо, йдуть шість блоків редагування з класом form-group, які складаються з назви та поля для редагування відповідного параметра.

Перший блок відповідає за редагування назви задачі; другий – групи; третій – містить випадаючий список з типами задач, а саме: збору показників температури повітря, вологості повітря та освітлення; четвертий – дата та час запуску задачі, варто зауважити, що якщо задача вже була виконана, то дане поле є недоступним; п'ятий – аналогічний попередньому, однак для дати та часу закінчення задачі; шостий – містить випадаючий список опцій повторення виконання, якщо задача вже виконана – недоступний. Останнім елементом форми є кнопка відправки даних. Після форми розташовано посилання на зручне повернення до сторінки управління задачами.

Результат створення представлення для редагування задачі зображений на рис. 4.24. Дивлячись на форму можемо сказати, що нам доступні всі поля редагування, що значить – задача ще не завершилась. Також, варто додати, що немає необхідності вводити дату початку/кінця виконання вручну, оскільки при натисканні на відповідне поле, буде відображений віджет, що надасть зручно вибирати потрібну нам дату та час. Якщо оновлення даних завершилось умпішно, буде надіслано відповідне сповіщення.

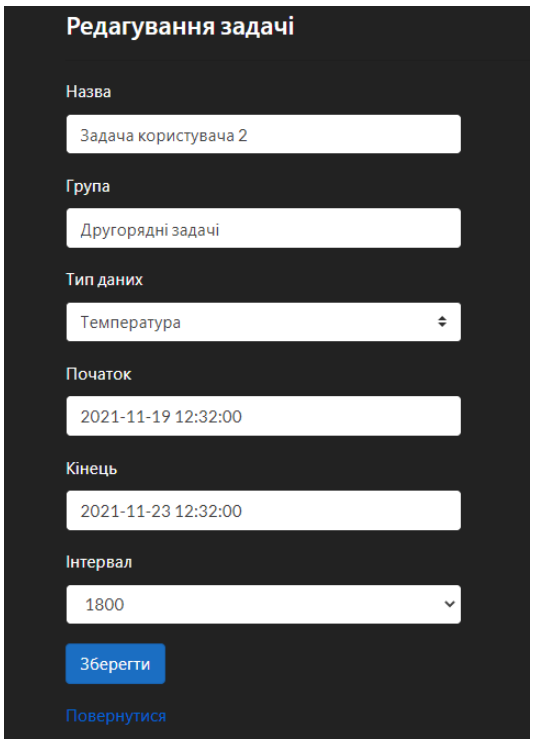


Рисунок 4.24 – Сторінка редагування задачі

Варто зауважити, що всі сторінки управління мають подібну функціональність описану вище, окрім сторінки управління показниками. Другий блок даної сторінки містить багато-функціональну таблицю (рис. 4.25), яка дозволить швидко фільтрувати, групувати та сортувати показники. Блок-схема алгоритму фільтрації показників наведена в додатку Ж. Це було зроблено з метою швидкого перегляду необхідних нам показників серед чисельної кількості.

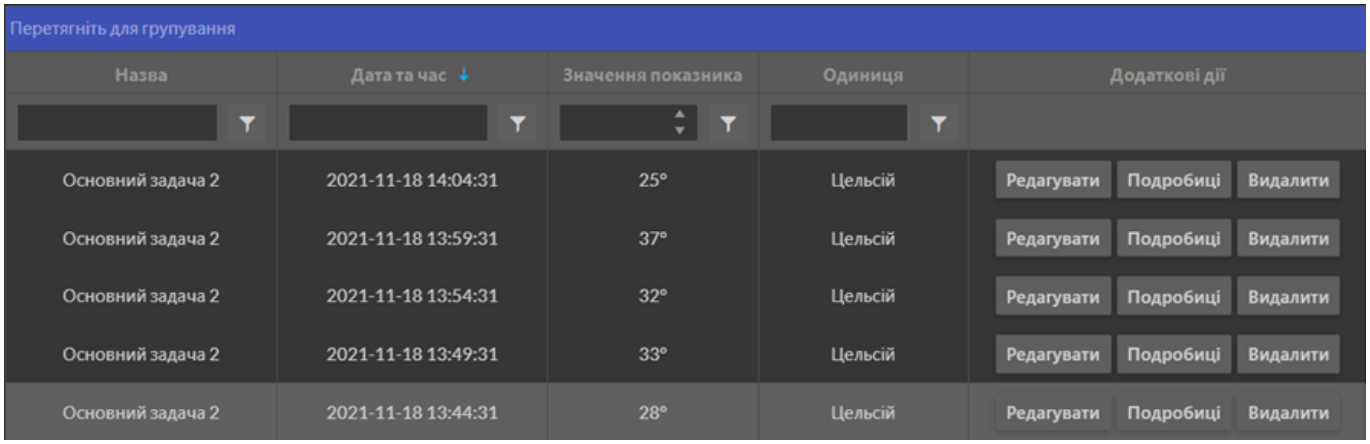


Рисунок 4.25 – Багато-функціональна таблиця

Рисунки інтерфейсу веб-додатку наведені в додатку И.

4.4 Розробка юніт тестів

На кінцевому етапі розробки важливо переконатися, що веб-додаток функціонує коректно і в ньому немає помилок. Існують чимало механізмів для тестування та перевірки додатку, ми використаємо механізм юніт-тестів. Юніт-тести призначені для автоматичного та швидкого тестування окремих компонентів додатку, незалежно від решти компонентів. Також вони несуть такі переваги, як написання слабопов'язаних компонентів згідно принципів SOLID, оскільки для тестування компонентів незалежно один від одного, необхідно, щоб вони були слабко пов'язані.

Як правило, для розробки юніт-тестів використовують спеціальні фреймворки, в даному випадку – xUnit.net, оскільки являється найбільш популярним та сумісним для роботи з .NET. Кожен юніт тест має модель AAA (Arrange-Act-Assert):

– Arrange – ініціалізація початкових дани для запуску тесту;
– Act – виконання логіки перевірки;
– Assert – верифікація отриманих результатів з очікуваними;

Розглянемо зразок написання юніт-тесту для перевірки кількості користувачів, що повертається методом Index контроллера для управління користувачами в системі (рис. 4.26).

```
[Fact]
public void IndexRetViewResWithUsrs()
{
    // Arrange
    var mck = new Mock<IUserRepository>();
    mck.Setup(repo => repo.GetAll()).Returns(GetTestUsers());
    var cntrl = new UsersController(mck.Object);

    // Act
    var result = cntrl.Index();

    // Assert
    var viewRes = Assert.IsType<ViewResult>(result);
    var mdl = Assert.IsAssignableFrom<IEnumerable<User>>(
        viewRes.Model);
    Assert.Equal(GetTestUsers().Count, mdl.Count());
}
```

Рисунок 4.26 – Unit-test для перевірки кількості користувачів

На рисунку помітно, що метод помічається атрибутом Fact, який використовується для позначення юніт-теста. Кожен тест повинен мати унікальну та зрозумілу назву.

Тест розпочинається з секції Arrange, яка ініціалізує мок об’єкт репозиторія користувачів та передає в конструктор контроллера. Мок об’єкти застосовуються для імітації роботи складних об'єктів, в поточному випадку – імітація повернення користувачів з бази даних, оскільки ми насамперед перевіряємо коректність обробки вже повернутих користувачів в методі контроллера. В секції Act, ми запускаєм метод контроллера та отримуємо результати. Наступна секція Assert, безпосередньо, здійснює перевірку отриманих результатів, а саме: тип повернутого результату – очікуємо тип представлення, тип моделі – очікуємо тип колекції користувачів та перевіряємо отриману кількість користувачів з очікуваною. Як можна помітити, перевірка результатів відбувається за допомогою методів класа Assert, якщо перевірка в одному з методів повернула негативний результат, виконання юніт-теста завершується невдало з описом відповідної помилки.

Результат виконання зображений на рис. 4.27.

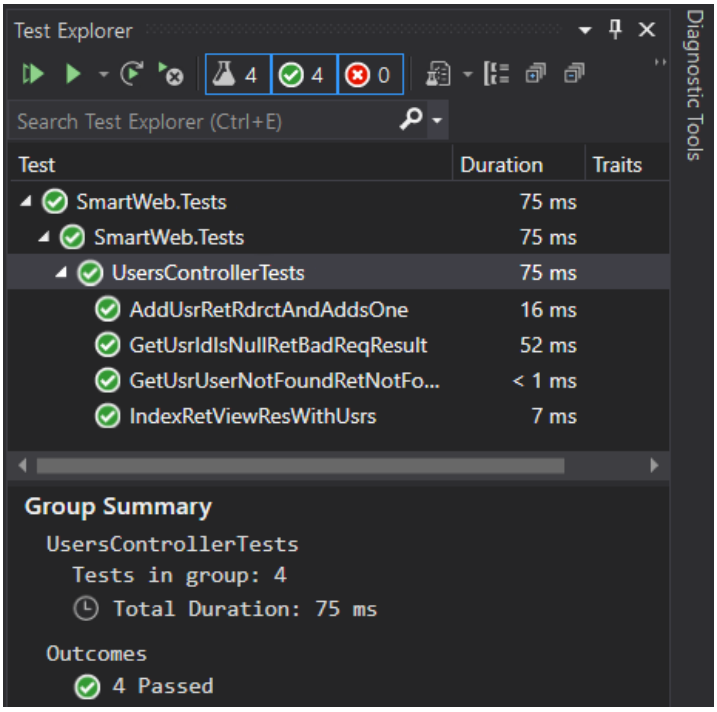


Рисунок 4.27 – Результат запуску тестів

Проаналізувавши вище наведений рисунок, можемо сказати, що всі тести виконались успішно. Для виконання та перегляду результатів ми використовуємо спеціальне вікно Test Explorer в Visual Studio. Зверху відображається список тестів та результат виконання. Знизу розташовується панель Group Summary, яка містить загальну інформація про кількість виконання юніт тестів, час виконання тощо. Якщо тест завершився невдало то поряд з його назвою з’являється відповідний статус. При натисненні на нього в Group Summary з’явиться детальна інформація про причину невиконання тесту, очікуване та отримане значення та розташування де виникла дана помилка (рис. 4.28).

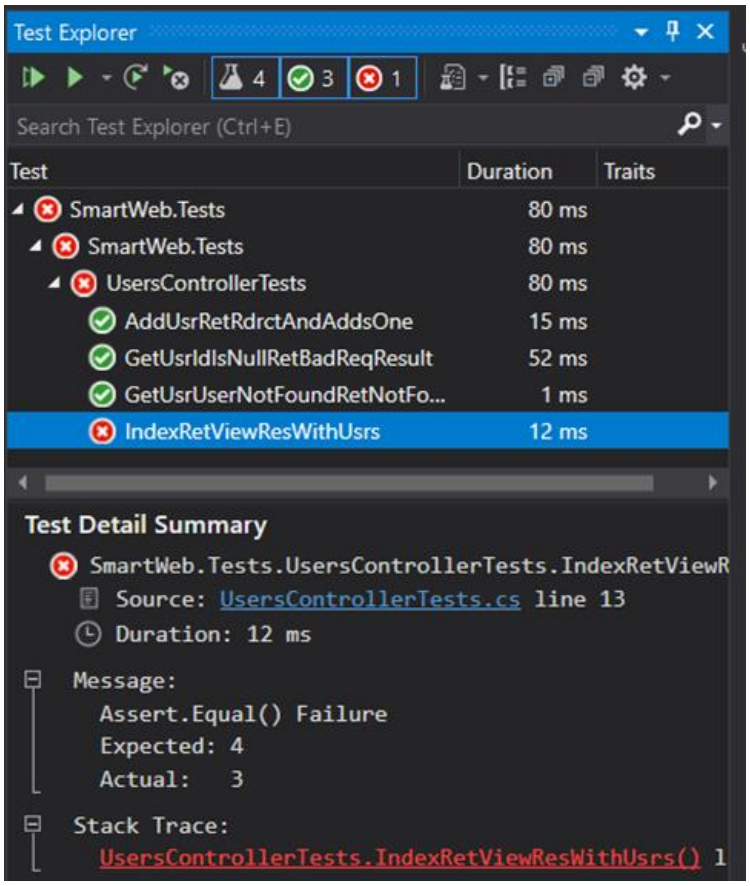


Рисунок 4.28 – Інформація про невдалий тест

Аналогічні юніт тести були написані для всіх компонентів веб-додатку та були виконані. Компоненти, тести яких завершились невдало, були виправлені та було здійснено повторне їх тестування. Даний процес виконувався для всіх методів, поки всі тести не завершились коректно.

Висновок до розділу 4

В даному розділі було описано створення веб-орієнтованої системи аналізу показників робототехнічних датчиків в розумному будинку. На початку було описано розробку моделей для роботи з базою даних. Модель сутності являє собою звичайний клас, що описується за допомогою властивостей та атрибутів аннотації. За замовчуванням, в базу даних будуть мапитись лише публічні властивості з модифікаторм public для сеттера. Для опису більш складної логіки зіставлення між моделями та таблицями, наприклад: створення первинних ключів, зв'язків багато до багатьох, і т.д. використовується підхід Fluent API. Окрім того, був здійснений аналіз всіх можливих запитів до бази даних та визначено ті таблиці та поля, котрі використовуються найчастіше, і на основі отриманої інформації було створено індекси для відповідних полів таблиць.

Наступним кроком став опис розробки серверної частини основних компонентів системи. За відображення форми для заповнення даних для входу та реєстрації, їх серверну валідацію та подальшу обробку відповідає контроллер AccountController. Власне, при заповненні реєстраційних даних створюється post запит на метод контроллера. Перед обробкою даних, відбувається серверна валідація, у випадку коректного завершення валідації, відбувається запис користувача в базу даних та відправка листа для підтвердження на вказану ним адресу. При некоректній валідації відображаються відповідні повідомлення. Для управління певним функціоналом (складними задачами, користувачами) використовуються CRUD контроллери. Кожен з яких містить чотири основних метода: create – містить логіку створення об'єкта; edit – призначений для редагування об'єкта; details – повертає детальну інформацію про певний об'єкт; delete – видаляє об'єкт з системи. Кожен з контроллерів, окрім вище описаних методів, також містять додаткові методи для реалізації специфічної функціональності.

Далі була описана клієнтська логіка створення основних компонентів. Для створення компонентів використовуються представлення, які навідміну від звичайних html документів, також містять синтаксис Razor – спеціальні конструкції,

які дозволяють переходити від коду html до c# та надають додаткові можливості. За відображення окремого компонента відповідає система сіток bootstrap, в результаті чого досягається гнучкість та адавтивність вмісту сторінку, що робить її зручною для перегляду незалежно від ширини екрана. Окрім цього було описано створення графіків та діаграм для візуалізації певних даних. За їх створення відповідає об'єкт highcharts, обов'язковими аргументами якого є: селектор тегу, де відображати графік; тип діаграми; її назва; мінімальне, максимальне та масив значень для вісі абсцис та ординат; масив об'єктів типу point. Для створення таблиць, котрі містять основну інформацію про об'єкти, використовуються нативні теги, а також цикл foreach, який перебирає модель типу колекції об'єктів та почергово створює рядки таблиці.

Останнім кроком був опис розробки юніт тестів, котрі застосовуються для швидкого тестування окремих компонентів додатку, незалежно від решти компонентів. Юніт тест створюється за моделлю AAA (Arrange-Act-Assert), де arrange відповідає за ініціалізацію початкових даних; act – за виконання логіки перевірки; assert – верифікую отримані результати. Для створення тестів використовувався фреймворк xUnit.net, який має накращу сумісність для роботи з .NET.

5 РОЗРОБЛЕННЯ СТАРТАП-ПРОЄКТУ

Останнім етапом є здійснення маркетингово аналізу стартап-проєкту для формулювання стратегії та перспектив перспектив під час впровадження на зовнішньому ринку стартапів.

5.1 Опис ідеї проєкту

Основна ідея стартап-проєкту полягає в створенні веб-орієнтованої системи аналізу і візуалізації даних в режимі реального часу та керування задачами для збору показників на певних проміжках часу в розумному будинку. Одними з основних чинників системи є масштабованість, адаптивність, швидкість та точність завантаження показників датчиків в режимі реального часу. Як тільки будуть досягнуті дані чинники система зможе досягнути високих результатів аналізу та обробці даних при користуванні декількома користувачами одночасно.

Наступним кроком є аналіз і розширений опис головних переваг та напрямків використання системи користувачами (табл 5.1).

Таблиця 5.1 – Опис та аналіз ідеї стартап-проєкту

Мета	Сфери використання	Переваги для клієнта
Створення системи аналізу, обробки та керування задачами для збору даних на визначених проміжках часу	Використання у власній оселі для зручного аналізу та спостереження за зовнішніми умовами в будинку	Кінцеві користувачі одержують програмне забезпечення, яке дає змогу переглядати та здійснювати аналіз стану в приміщенні в реальному часі та на визначених інтервалах часу
	Застосування в офісних приміщень та на підприємствах	

Після цього потрібно сформулювати головні недоліки та переваги в порівняння з існуючими рішеннями на ринку. Головними конкурентами являються Home

Assistant та openHAB. В таблиці 5.2 наведемо результати аналізу та порівняння створеної системи з конкуруючими.

Таблиця 5.2 – Визначення сильних, слабких та нейтральних характеристик ідеї проєкту

№	Технічні характеристики	Існуючі аналоги			W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій додаток	Home Assistant	openHAB			
1	Адаптивність та зручність інтерфейсу	+	+	+		+	
2	Масштабованість додатку	+	+	-		+	
3	Швидкість завантаження та обробки інформації	+	-	-			Використання кешування та мінімізації веб-ресурсів для підвищення швидкодії операцій
4	Керування збором та аналізом показників на визначених межах дат	+	-	-			Створення задач користувачами для керування збором на необхідному проміжку часу

№	Технічні характеристики	Існуючі аналоги			W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій додаток	Home Assistant	openHAB			
5	Багатоплатформеність	+	-	-			Розробка система з сучасними технікам відображення веб вмісту незалежно від розміру екрану.

На вище наведеній таблиці можемо помітити, що створена система має численні переваги в порівняння з головними існуючими рішеннями.

5.2 Технічний аудит ідеї проєкту

Після цього нам потрібно здійснити аналіз альтернативних технологічних засобів за допомогою яких буде реалізовуватись ідея стартап-проєкту. Беручи до уваги те, що на сьогоднішній час будь який веб-додаток реалізовується за допомогою таких інструментів як HTML і CSS, то для проведення аналізу виберемо технології для реалізії серверної частини додатку (табл. 5.3).

Таблиця 5.3 – Технологічна здійсненість ідеї проєкту

№	Ідея	Засоби реалізації	Наявність	Доступність
1	Реалізація додатку аналізу та обробки даних в режимі реального часу	Мова програмування C#	Доступна в наявності	Доступна
2		Мова програмування Python		

№	Ідея проєкту	Технології її реалізації	Наявність технологій	Доступність технологій
3	та керування збором даних на інтервалах часу	Мова програмування JavaScript		

Після проведення аналізу можливих технологій для створення продукту, можемо сказати, що для технічної реалізації ідеї проєкту нам доступно декілька різних технологій, з них в кінцевому результаті було вибрано першу технологію з використанням мови програмування C#.

5.3 Аналіз ринкових можливостей запуску стартап-проєкту

Для успішного впровадження стартапа на зовнішній ринок, необхідно проаналізувати всі доступні можливості ринку. Спочатку проаналізуємо його пепередню характеристику (табл 5.4).

Таблиця 5.4 – Попередня характеристика потенційного ринку

№	Найменування	Опис
1	Кількість головних гравців, од	2 (виробники систем пристроїв інтернет речей)
2	Загальний обсяг продаж, грн/ум.од	~123 000 грн/ум.од
3	Динаміка потенційного ринку	Зростає
4	Наявність обмежень для входу	Необхідний доступ до мережі інтернет
5	Специфічні вимоги до сертифікації і стандартизації	Відсутні
6	Середня норма рентабельності в галузі (або по ринку), %	100%

Здійснивши аналіз попередньої характеристики, можемо сказати, що сучасний ринок для впровадження стартапа проєкту є привабливим для реалізації проєкту, оскільки маємо мінімальну кількість основних конкурентів, динаміка ринку поступово зростає та спостерігається хороша рентабельність.

Наступним кроком в аналізу ринкових можливостей є визначення та характеристика цільової аудиторії (табл. 5.5).

Таблиця 5.5 – Характеристика потенційних клієнтів стартап-проєкту

№	Запити	Цільові клієнти	Різниця у поведінці різних цільових клієнтів	Головні вимоги клієнтів до продукту
1	Запит в зручному перегляді та аналізі основних умов в приміщенні	Власники квартир та багатоповерхових будинків. До того ж, система доступна для використання в офісних приміщеннях	Користувачі мають намір віддалено переглядати стан в оселі в режимі реального часу та в потрібний момент часу	Головними вимогами є зручність користування на мобільних пристроях та швидкодія оновлення даних

Виконавши аналіз потенційних клієнтів ми визначили проблему, яка формує ринок, їхні вимоги, зацікавленість та відмінність між різними групами і тому, можемо підсумувати, що стартап є привабливим як для власників осель, так і багатоповерхових будинків і офісних приміщень.

Подальший крок заключається в аналізу факторів можливостей, що посприяють впровадження стартапа на ринок (табл. 5.6).

Таблиця 5.6 – Фактори можливостей

№	Фактор	Зміст	Відповідь
1	Інтеграція іноземного економічного простору	Поширення продукту на іноземних покупців	Здійснення локалізації веб-інтерфейсу та запуск реклами
2	Надсилання сповіщень користувачам	Можливість миттєво надсилати сповіщення при ненормативних показниках відповідних датчиків	Використання протоколів передачі даних та підключення хмарних сервісів для програної системи

Далі визначаємо можливі фактори загроз, які здатні вплинути на продаж створеного продукту (табл. 5.7).

Таблиця 5.7 – Фактори загроз

№ п/п	Фактор	Зміст	Відповідь
1	Кіберзагроза	Проникнення на сервер шкідливого програмного забезпечення, яке здатне зупинити систему	Поступовий аудит рівнів безпеки
2	Появи нових конкурентів	В процесі або після виходу існує ймовірність появи конкурентів	Поступове впровадження унікальних функціональних можливостей

№	Фактор	Зміст	Відповідь
3	Виникнення системних помилок	Під час користування додатком існує ймовірність виникнення специфічних помилок	Здійснення логування в усіх можливих місцях виконання дій в системі

Наступною дією йде виконання ступеневого аналізу конкуренції (табл. 5.8).

Таблиця 5.8 – Ступеневий аналізу конкуренції на ринку

Властивості конкуруючого середовища	Опис даної характеристики	Відповідь компанії (дії компанії, щоб бути конкуруючоспроможною)
1. Тип конкуренції: чиста	Запропонування товару, якому не має конкурентів	Пропонування унікальних функціональних можливостей
2. Рівень конкурентної боротьби: міжнаціональний	Дана система орієнтована на продаж по всьому світу	Впровадження локалізації відповідно до мови країни
3. Галузева ознака: внутрішньогалузева	Систему слід використовувати для спостереженням за показниками датчиками	Підвищення швидкодії та точності отримання даних
4. За видами товарів: товарно-видова	Основна функція – аналіз даних на визначених проміжка часу	Шанс запланувати обробку даних в потрібний момент часу
5. За характером конкурентних переваг: нецінова	Система підтримує інтеграцію для інших платформ	Можливість іншим платформам використовувати програмний інтерфейс

Властивості конкуруючого середовища	Опис даної характеристики	Відповідь компанії (дії компанії, щоб бути конкуруючоспроможною)
6. За інтенсивністю: не марочна	Алгоритми та функції обробки даних в реальному часу позитивно відрізняються від конкурентів	Створення та поступовий розвиток бренду, що досягне високої популярності

Далі нам потрібно проаналізувати конкурентність за М. Портером (табл. 5.9).

Таблиця 5.9 – Аналіз конкуренції в галузі за М. Портером

	Головні конкуренти в сфері	Потенційні опоненти	Провайдери	Покупці	Товари-замінники
Складові	Розробники пристроїв інтернет речей для розумного будинку	Інші розробники подібних систем	Постачальники повинні мати кваліфікацію в конфігурації серверу	Клієнти мають вміти використовувати засоби зворотнього зв’язку	Пристрої інтернет речей
Висновки	Необхідно досягти високої швидкодії та точність обробки даних	За рахунок гнучкої масштабованості можна стати конкуренстоспроможним	Відсутні	Відсутні	Відсутній намір клієнтів розбиратися з системою

На основі проведеного аналізу, можемо підсумувати, що система, не зважаючи на наявність конкурентів, залишатиметься конкурентоспроможною, оскільки останні мають обмежений набір функціональних можливостей. До того ж, під час створення стартапа, перш за все, необхідно брати до уваги побажання користувачів та створення функціоналу, що відсутній у головних конкурентах.

Здійснимо обґрунтування факторів конкурентоспроможності створеного додатку (табл. 5.10).

Таблиця 5.10 – Обґрунтування факторів конкурентоспроможності

№	Фактор	Опис (перелік умов, що роблять фактор для порівняння аналогів істотним)
1	Автоматизація процесу обробки даних	Основні конкуренти подібних систем не мають змогу створити аналіз та обробку даних на визначених інтервалах часу
2	Швидкодія	За рахунок кешування та мінімізації веб-ресурсів, зменшується час очікування даних з сервера
3	Адаптивність	Можливість адаптувати елементи веб-інтерфейсу під будь які розміри пристрою

Далі потрібно виконати порівняльний аналіз сторін додатку (табл. 5.11).

Таблиця 5.11 – Порівняльний аналіз сильних та слабких сторін системи

№	Фактор	Бали 1-20	Рейтинг аналогів в порівнянні з Home In Smart World						
			–3	–2	–1	0	+1	+2	+3
1	Автоматизація процесу обробки даних	11			+				
2	Швидкодія	16		+					
3	Адаптивність	12			+	*			

Далі виконаємо SWOT аналіз (табл. 5.12).

Таблиця 5.12 – SWOT аналіз

Сильна сторона: – Автоматизаціія процесу обробки даних – Висока швидкість обробки та передачі даних – Мультиплатформеність	Слабка сторона: – Поява конкурентів на ринку – Слабка довіра користувачів до нових рішень
Перспективи: – Масштабувати продукт під побажання клієнта	Загрози: – Недостатність фінансування

На основі SWOT-аналізу необхідно скласти альтернативи ринкової поведінки стартап проєкту (табл. 5.13).

Таблиця 5.13 – Альтернативи ринкового впровадження стартап проєкту

№	Альтернативні умови (приблизний перелік дій) поведінки на ринку	Шанс одержання ресурсів	Терміни впровадження
1	Надання системи в пробний період користування	Висока ймовірність (85%)	1-2 місяці
2	Презентація на певних публічних ресурсах	Середня ймовірність (50%)	1 місяць

Підсумувавши результати виконаного аналізу, можемо сказати, що необхідно надавати систему в пробний період користування та презентувати на публічних ресурсах для впровадження стартапу на ринок.

5.4 Розробка ринкової стратегії стартап-проєкту

Наступним етапом є розробка ринкової стратегії для охоплення ринку. Для цього спочатку необхідно визначитись з цільовими клієнтами (табл. 5.14).

Таблиця 5.14 – Вибір цільових груп потенційних споживачів

№	Опис націленої групи користувачів	Бажання клієнтів сприйняти рішення	Орієнтовний попит в межах потенційних клієнтів (груп)	Потужність конкуренції на ринку	Простота входу на ринок
1	Застосування в приватних помешкань та будинках	Готовність має бути високою, оскільки дана система покращує її аналоги	Високий попит, оскільки конкуруючі системи мають недоліки	Враховуючи те, що система має нові функції, конкуренція не очікується	Готовність використання продукту підприємствами та власниками осель
2	Використання в офісних та підприємницьких приміщень				
Головні клієнти: власники осель, багатоповерхові будівлі та офісні приміщення					

Після попереднього аналізу було визначено, що цільовим використанням є житлові та підприємницькі приміщення. Враховуючи це, було обрано стратегію масового маркетингу. Подальший крок полягає в створення базової стратегії розвитку (табл. 5.15).

Таблиця 5.15 – Визначення базової стратегії розвитку

№	Вибране альтернативне рішення розвитку додатку	Стратегія виходу ринку	Головні конкурентоспроможні позиції базуючись на обраній альтернативи	Базова стратегія розвитку*
1	Ринок: приватні оселі та офісні приміщення	Пропозиція моніторингу за станом в приміщеннях в визначені інтервали часу	Зниження плати порівняно з аналогами, захопленість зацікавленими користувачами,	Диференціації

Наступною дією являється визначення базової стратегії конкуруючої поведінки (табл. 5.16).

Таблиця 5.16 – Визначення базової стратегії конкурентної поведінки

№	Чи є створене рішення «першопрохідцем»?	Чи має на меті компанія залучати існуючих клієнтів у конкурентів чи шукати нових?	Чи має на меті компанія копіювати головний функціонал продукту конкурента, та який?	Стратегія конкуруючої поведінки*
1	Ні, оскільки існують аналоги з меншою функціональністю	Система матиме як і нових користувачів так і конкурентів	Система копією основні характеристики з додаванням нових	Заняття конкурентної ніші

Останньою дією є визначення стратегії позиціонування (табл. 5.17).

Таблиця 5.17 – Визначення стратегії позиціонування

№	Вимоги до продукту у націлених клієнтів	Базова стратегія розвитку	Ключові переваги позиції власного створеного продукту	Вибір асоціацій, які мають сформулювати комплексне рішення власного проекту (три основних)
1	Швидкодія, висока фунціональність та простота використання	Диференціації	Унікальні функціональні можливості, вдосконалення проєктування інтерфейсів	Чудова швидкість обробки даних Зручність і стабільність використання Підтримка клієнта

На основі проведеного аналізу було прийнято рішення вибрата стратегію заняття конкурентної ніші та диференціації.

5.5 Розробка маркетингової програми стартап-проєкту

Першим кроком при розробці маркетингової програми є визначення переваг потенційного товару (табл. 5.18).

Таблиця 5.18 – Визначення переваг концепції товару

№	Потреба	Перевага, яка надається товаром	Основні переваги над аналогами
1	Спостереження за станом оселі або приміщення підприємства	Спостереження за станом приміщення в режимі реального часу	Простота, доступність та зручність в користування та висока плату за використання в головних конкурентах

В наступній дії нам потрібно визначити опис рівнів моделі товару (табл. 5.19).

Таблиця 5.19 – Опис трьох рівнів моделі товару

<table>
<tr><td>Рівні товару</td><td colspan="3">Сутність та складові</td></tr>
<tr><td>I. Товар за задумом</td><td colspan="3">Надання користувачам можливість легко спостерігати за станом приміщення в реальному часі, здійснювати аналіз отриманих показників та створювати задачі для збору даних в певних проміжках часу</td></tr>
<tr><td rowspan="5">II. Товар у реальному виконанні</td><td>Властивості/характеристики</td><td>М/Нм</td><td>Вр/Тх /Тл/Е/Ор</td></tr>
<tr><td>1. Простота в використанні
2. Швидкість обробки та завантаження даних
3. Вартість</td><td>М
М
Грн.</td><td>Тх
Тх
Низька</td></tr>
<tr><td colspan="3">Якість: Надійність та точність аналізу показників</td></tr>
<tr><td colspan="3">Пакування: надання ліцензії</td></tr>
<tr><td colspan="3">Марка: HSW – Home in Smart World</td></tr>
<tr><td>III. Товар із підкріпленням</td><td colspan="3">Надання гарантії терміном 1 рік на безкоштовне усунення можливих недоліків</td></tr>
<tr><td colspan="4">За рахунок отримання патенту на програмний код та реєстрації торгової марки, товар матиме захист від копіювання</td></tr>
</table>

Подальшою дією є визначення меж встановлення вартості (табл. 5.20).

Таблиця 5.20 – Визначення меж встановлення вартості

№	Вартість цін на замінники	Вартість цін на аналоги	Рівень доходів націленої групи клієнтів	Межі встановлення вартості на товар
1	-	Рівень цін варіюється від вибраного пакету	Від 25 тис. грн.	Продаж: 750$ - 1000$ Підтримка: 100$ в місяць

Подальший крок полягає в формуванні системи збуту (табл. 5.21).

Таблиця 5.21 – Формування системи збуту

№	Специфіка купівельної поведінки основної групи користувачів	Функції збуту, які зобов'язується здійснювати постачальник продукту	Глибина канала	Оптимальна система
1	Власники підприємств та осель, що прагнуть спостерігати за станому будинку в будь який момент часу	Функції ліцензії	Однорівнева глибина для певних користувачів	Традиційна

Останнім крок полягає у визначенні концепції маркетингових комунікацій (табл. 5.22).

Таблиця 5.22 – Концепція маркетингових комунікацій

№	Специфіка діяння націлених користувачів	Засоби контактування, які використовують користувачі	Головні позиції, вибрані для розміщення	Мета рекламної кампанії	Задум для рекламного звернення
1	Придбання продукту та технічна підтримка від компанії	Електронна скринька та мобільний телефон	Простота, функціональність та підтримка	Привернути уваги клієнтів до функцій продукту	Класичний

В даному пункті був здійснений опис моделі товару, визначені межі встановлення вартості, сформована система збуту та переваги товару, і враховуючи опис було створено створено ринкову програму.

Висновок до розділу 5

На початку даного розділу був здійснений опис основної ідеї стартап-проєкту, і в підсумку можемо сказати, що головна ідея проєкту полягає в полягає в створенні веб-орієнтованої системи обробки та візуалізації даних про стан приміщення в режимі реального часу в розумному будинку. Далі було визначено технологічний аудит стартапа, в результаті чого для розробки додатку була вибрана мова програмування C#.

Наступною дією став аналіз ринкових можливостей запуску стартапу, де спочатку було проведено ступеневий аналізу конкуренції на ринку так і за М. Портером, де в підсумку було визначено, що система залишатиметься конкурентноспроможною, оскільки головні конкуренти містять обмежені функції в використанні. Наступним став SWOT-аналіз, на основі якого, було визначено, що для успішного впровадження на ринок необхідно надавати систему в пробний період користування.

Наприкінці було розроблено ринкову та маркетиногову стратегію проєкту, які полягають у визначенні та формуванні концепції товару, його збуту, вартості тощо. та здійснено аналіз усіх пройдених етапів. В кінцевому підсумку можемо сказати, що даний проєкт може бути проваджений на ринок та посісти свою ринкову нішу.

ВИСНОВОК

В ході роботи над магістерською дисертацією було розроблено веб-орієнтовану систему аналізу показників робототехнічних датчиків в розумному будинку, яка здійснює аналіз показників температури, вологості повітря та рівня освітлення в розумному будинку.

На першому етапі роботи був здійснений опис предметної області. Проведено аналіз найпопулярніших існуючих рішень для отримання інформації про їхні переваги та недолі. Першою системою стала платформа «Home Assistant». Серед головних переваг варто виділити швидкодію та безкоштовне використання. Недоліками платформи є неповнота документації та складність розуміння на початку користування. Другою системою стала openHAB, що надає API для інтеграції в інші системи та підтримує кросплатформеність. Однак основними недоліками є повільний розвиток та складний процес встановлення. Останньою системою для розгляду став сервіс «Azure IoT», який надає просту інтеграцію з будь-якими існуючими системами та чудову масштабованість. Головним недоліком сервісу є висока плата за використання. Наприкінці даного етапу була поставлена мета дисертації та описано основні кроки для її досягення.

Наступним кроком став аналіз сучасних серверних та клієнтських технологій для розробки веб-додатку. В результаті для розробки серверної сторони був вибраний стек .NET. В ролі мови програмування був вибраний C#. Для абстрагування від бази даних був вибраний Entity Framework Core. Для створення бази даних – SQL Server, оскільки її досить легко інтегрувати EF-Core. Для створення веб додатку вибраний ASP.NET Core, оскільки він надає нативні інструменти для написання клієнтської логіки. Для ведення робіт на клієнтській стороні серед нативних інструментів були вибрані: мова розмітки HTML, каскадна таблиця стилів CSS та мова програмування JavaScript. Для пришвидшення розробки був використаний css-препроцесор Less, UI-фреймворк Bootstrap та клієнтська бібіліотека jQuery.

Було здійснено опис структури системи. Структура системи містить вісім основних модулів: «Авторизації користувачів», «Керування користувачами»,

«Відправки сповіщень», «Завантаження показників в реальному часу», «Виконання складних задач», «Збереження діаграм та графіків», «Керування складними задачами» та «Керування показниками». Була описана структура бази даних, яка складається з десяти таблиц. Головними таблицями стали Users, Tasks, HumiditiesData, TemperaturesData та LightsData. До того ж, було описано міграцію моделей та механізм сідів, що використовується для заповнення бази даних тестовими даними. Було описано виконання складних задач та визначено їхні типи – основні та другорядні. Також був здійснений опис технік оптимізації HTTP-запитів, в результаті чого зменшилась кількість запитів до сервера та розмір веб ресурсів.

Подальший крок полягав в описі розробки системи. Спочатку була описано розробку моделей для роботи з БД, де модель сутності представляє звичайний клас, що описується за допомогою властивостей та атрибутів аннотації. Далі описувалась серверна логіка розробки основних компонентів системи. Варто зазначити, що для управління певним функціоналом використовуються CRUD контроллери. Кожен з яких містить чотири основних метода: create, edit, details та delete. Окрім того, кожен з них містять додаткові методи для реалізації специфічної функціональності. Для розробки клієнтської логіка основних компонентів використовувались представлення, які навідміну від звичайних html документів, дозволяють переходити від html до c#, в результаті – отримуємо додаткові можливості. За відображення окремого компонента відповідає система сіток bootstrap, яка надає адавтивність вмісту сторінку. Наприкінці було описано розробку юніт тестів, що використовуються для швидкого тестування окремих компонентів додатку.

Останнім кроком роботи було розроблення стартап-проєкту для створеного додатку, в кінцевому підсумку була визначена стратегія впровадження додатку на ринок та потеційні клієнти.

Результатом виконаної роботи є веб-додаток, який застосовується для спостереження за станом в приміщенні оселі або підприємства. Одними з основних функцій є створення складних задач для збору і аналізу даних на заданих інтервалах часу та надсилання сповіщень при перевищенні останніми нормативних значень через засоби комунікації.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Визначення розумного будинку – URL: https://www.investopedia.com/terms/s/smart-home.asp

2. Первое знакомство с Home Assistant – URL: https://habr.com/ru/post/471822/

3. Опис платформи openHAB – URL: https://habr.com/ru/post/485848/

4. Azure Services IoT – URL: https://www.predicagroup.com/blog/azure-services-iot/

5. Мова програмування C# – URL: https://whatis.techtarget.com/definition/C-Sharp

6. Введення в платформу .NET 5 – URL: https://devblogs.microsoft.com/dotnet/introducing-net-5/

7. C# 7.0. Справочник. Полное описание языка, 7-е издание / Джозеф Албахари, Бен Албахари – Диалектика, 2016 – 1088с

8. C# 7 и .NET Core. Кросс-платформенная разработка для профессионалов | Прайс Марк: Пер. с англ. – СПб.: “Питер”, 2018 – 640с.

9. Платформа з відкритим вихідним кодом .NET – URL: https://auth0.com/blog/what-is-dotnet-platform-overview/

10. ORM-інструмент Entity Framework – URL: https://www.tutorialspoint.com/entity_framework/entity_framework_overview.htm

11. Mark J. Price C# 10 and .NET 6 – Modern Cross-Platform Development: Build apps, websites, and services with ASP.NET Core 6, Blazor, and EF Core 6 using Visual Studio 2022 and Visual Studio Code, 6th Edition: Packt Publishing; 6th Edition, 2021 – 824c.

12. Адам Фримен Entity Framework Core для ASP.NET Core MVC, 2-е изд.: Пер. с англ. – СПб.: ООО “Диалектика”, 2020. – 624 с. : ил. Парал. тит. англ.

13. Веб-фреймворк ASP.NET Core – URL: https://www.tutorialsteacher.com/core/aspnet-core-introduction

14. Middleware in ASP.NET Core – URL: https://wakeupandcode.com/middleware-in-asp-net-core/

15. Andrew Lock ASP.NET Core in Action, Second Edition: Publishing house “Manning”, 2021. – 832c.

16. Система управління базами даних SQL Server – URL: https://www.sqlservertutorial.net/getting-started/what-is-sql-server/

17. Алас Болье Изучаем SQL. Генерация, выборка и обработка данных. 3-е издание: Пер. с англ. – СПб. : ООО “Диалектика”, 2021. – 402 с.

18. Документація по мові запитів T-sql – URL: https://docs.microsoft.com/en-us/sql/t-sql/language-reference?view=sql-server-ver15

19. Каскадна таблиця стилів CSS – URL: https://www.tutorialspoint.com/css/what_is_css.htm

20. Опис CSS синтаксу – URL: https://www.tutorialspoint.com/css/css_syntax.htm

21. CSS-препроцесор Less – URL: https://www.tutorialspoint.com/less/less_overview.htm

22. Мова програмування JavaScript – URL: https://www.tutorialspoint.com/javascript/index.htm

23. Офіційна документація по JavaScript – URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide

24. Порівняння UI-фреймворків – URL: https://www.geeksforgeeks.org/bootstrap-vs-material-ui/

25. Хмарна платформа Microsoft Azure – URL: https://www.tutorialspoint.com/microsoft_azure/cloud_computing_overview.htm

26. Создание статического веб-приложения HTML в Azure – URL: https://docs.microsoft.com/ru-ru/azure/app-service/quickstart-html

27. Опис основних можливостей Cron expression – URL: https://en.wikipedia.org/wiki/Cron