Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мережних технологій факультету інформатики

РОЗРОБКА WEB-ЗАСТОСУВАННЯ ДЛЯ МОНІТОРИНГУ SSL-СЕРТИФІКАТІВ СЕРВЕРА ТА ЇХ ОНОВЛЕННЯ

Текстова частина до курсової роботи
за спеціальністю «Прикладна математика», 113

Керівник курсової роботи
ст. в.
Сініцина Р. Б.

_____

Виконав студент
Огир В. Д.

_____

Київ 2021

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мережних технологій факультету інформатики

ЗАТВЕРДЖУЮ
Зав. кафедри мережних технологій,
проф., д. ф.-м. н. Г. І. Малашонок
_____ (підпис)

“____” _____2020 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу

студента Огира Вадима Дмитровича факультету інформатики 3 курсу
Тема: Розробка WEB-застосування для моніторингу SSL-сертифікатів сервера та їх оновлення

Зміст ТЧ до курсової роботи:

Індивідуальне завдання
Календарний план
Зміст
Перелік умовних позначень
Вступ
Розділ 1: Теоретичні відомості і постановка завдання
Розділ 2: Розробка застосування
Розділ 3: Тестування застосування
Висновки
Перелік використаних джерел

Дата видачі “___” _____ 2020 р. Керівник _____
(підпис)

Завдання отримав _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН ВИКОНАННЯ КУРСОВОЇ РОБОТИ

Тема: Розробка WEB-застосування для моніторингу SSL-сертифікатів сервера та їх оновлення

Календарний план виконання роботи:

№ п/п	Назва етапу курсового проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу	16.10.2020	
2.	Ознайомлення з існуючою інформацією за темою курсової роботи	17.10.2020	
3.	Проектування бекенду застосунку	25.11.2020	
3.	Реалізація парсингу SSL-сертифікатів	30.12.2020	
4.	Початок розробки бекенду застосунку	02.01.2021	
5.	Подання проміжної версії практичної частини	11.01.2021	
6.	Початок розробки фронтенду застосунку	01.03.2021	
7.	Початок розробки системи планування фонових задач і віддаленого виконання скриптів	20.04.2021	
8.	Аналіз практичної частини, її корегування	01.05.2021	
9.	Початок написання теоретичної частини	05.05.2021	
10.	Подання проміжної версії теоретичної частини	11.05.2021	
11.	Остаточне завершення написання теоретичної частини роботи	14.05.2021	
12.	Створення презентації	15.05.2021	
13.	Захист курсової роботи	20.05.2021	

Студент Огир В. Д.

Керівник Сініцина Р. Б.

“____” _____

ЗМІСТ

ПЕРЕЛІК ТЕРМІНІВ ТА УМОВНИХ ПОЗНАЧЕНЬ

- **HTTP (HyperText Transfer Protocol) –** протокол передачі даних прикладного рівня
- **FTP (File Transfer Protocol) –** протокол передачі файлів
- **SMTP, IMAP, POP** – протоколи передачі даних, призначені для передачі електронних листів
- **SSL/TLS** – криптографічні (захищені) протоколи передачі даних
- **HTTPS, SMTPS, IMAPS, POPS** – захищені версії відповідних протоколів, що працюють у зв'язці з SSL/TLS
- **TCP (Transmission Control Protocol)** – протокол передачі даних в мережі Інтернет, що виконує функції транспортного рівня моделі OSI
- **SSL-сертифікат –** вид цифрового сертифікату, що підтверджує володіння власником веб-серверу доменного імені, з яким він пов'язаний. Необхідний для встановлення захищеного з'єднання
- **Wildcard SSL-сертифікат –** SSL-сертифікат, дія якого розповсюджується на тільки на сам домен, а і на всі його піддомени
- **Endpoint –** кінцева адреса веб-ресурсу, на яку можуть бути відправлені HTTP-запити
- **ORM (Object-Relational Mapper) –** механізм або технологія конвертування даних між системами, які працюють з даними різного типу. Зазвичай використовується для роботи з базами даних
- **DOM (Document Object Model) –** програмний інтерфейс, що дозволяє програмам та скриптам отримати доступ до вмісту HTML-документу
- **API (Application Public Interface) –** публічний інтерфейс застосунку
- **SSH (Secure Shell) –** протокол прикладного рівня, який призначений для здійснення віддаленого керування сервером
- **Bash-команда –** команда для виконання у середовищі Unix Shell
- **Cron –** утиліта, призначена для часового планування виконання задач

ВСТУП

У сучасному інформаційному світі безпека має критичне значення, оскільки все більше програм працюють з особистими даними користувачів, які необхідно надійно захищати від зловмисників. І, оскільки робота сучасних програм, зокрема, веб-ресурсів, засновується на обміні інформацією через мережу Інтернет, були розроблені спеціальні протоколи і методи захисту з'єднань між користувачем і сервером. Найпоширенішим з цих протоколів є HTTP, а з недавнього часу – його захищене розширення – HTTPS.

Для того, щоб мати можливість використовувати HTTPS, на сервері має бути встановлений унікальний SSL-сертифікат, який і забезпечує безпечність з'єднання. Такі сертифікати підписуються деякими авторизованими центрами сертифікації, які перш ніж підписати сертифікат, мають впевнитись, що його отримувач – саме той, про кого йдеться у сертифікаті.

Проте є значний недолік: задля безпеки такі сертифікати мають обмежений термін дії – від трьох місяців до кількох років. І для того, щоб не залишати користувачів веб-ресурсу або додатку незахищеними, необхідно слідкувати за терміном дії сертифікатів і вчасно їх оновлювати. Саме для цього існують спеціальні сервіси для моніторингу і, інколи, автоматизованого оновлення SSL-сертифікатів.

Мета курсової роботи – дослідити існуючі сервіси для моніторингу і оновлення SSL-сертифікатів, визначити їх переваги і недоліки та створити універсальне застосування, яке дасть можливість системним адміністраторам зручно моніторити сертифікати їх серверів і автоматично їх оновлювати.

Завдання курсової роботи – розробити веб-застосування для моніторингу SSL-сертифікатів різних типів серверів: веб-серверів, поштових серверів, файлових серверів, а також для їх автоматизованого оновлення, якщо таку можливість надає центр сертифікації, або ж сповіщення про завершення терміну дії сертифікатів.

Об’єктом дослідження є розробка веб-застосунку з використанням фреймворку ASP.NET Core і бібліотеки React.JS.

Предметом дослідження є сучасні системи моніторингу веб-ресурсів з можливістю автоматизованого виконання наперед визначених команд.

У рамках курсової роботи було створено застосування для автоматизованого моніторингу і оновлення SSL-сертифікатів сервера. Воно дозволяє моніторити вибрані сертифікати, сповіщати адміністратора у разі закінчення терміну їх придатності, а також автоматизовано оновлювати сертифікати, виконавши деякий набір команд, які може вказати користувач. Це застосування може використовуватись як в особистих цілях, так і на підприємстві.

Курсова робота складається з трьох розділів, які містять підрозділи, висновків, списку використаної літератури та додатків.

Перший розділ містить теоретичні відомості, висвітлює актуальність теми, а також у ньому розглянуто аналоги розробленого застосунку і виділено їх переваги і недоліки.

У другому розділі описано розробку застосування. Описано алгоритм парсингу SSL-сертифікату веб-сервера, обґрунтовано обрані патерни проектування програмних продуктів, а також вибір технологій і бібліотек.

Третій розділ присвячений тестуванню реалізованого застосунку на реальному прикладі.

РОЗДІЛ 1

1.1 Аналіз сучасного стану питання та обґрунтування теми

Станом на сьогодні майже всі веб-ресурси в публічному інтернеті працюють за захищеним протоколом HTTPS, або віддають йому перевагу над звичайним HTTP. Проте, ще 7 років тому, за даними Google [1], лише 50% всіх ресурсів використовували захищене з'єднання. Ця тенденція має просту причину: все більше веб-сайтів працюють з особистими даними відвідувачів, що зобов'язує веб-ресурси належним чином ці дані обробляти і захищати.

Раніше, якщо необізнаний користувач використовував незахищений веб-ресурс, він навіть не здогадувався про це. Але тепер всі веб-браузери явно дають зрозуміти користувачу, що він у небезпеці, якщо користується веб-сайтом без захищеного з'єднання, що може дуже сильно відобразитись на репутації ресурсу.

Такий захист надає шифрування з'єднання, яке забезпечується протоколом HTTPS, що працює лише за наявності так-званих SSL-сертифікатів, які є унікальними для кожного веб-ресурсу, або навіть для окремих його частин, для надійнішого захисту. Ці сертифікати підтверджуються авторизованими сервісами, що називаються центрами сертифікації, а також мають нетривалий термін дії, що робить необхідним регулярну завчасну заміну цих сертифікатів на нові. Гаяти час при оновленні сертифікату, термін дії якого підбігає кінця, вкрай небажано, адже недійсний сертифікат браузери сприймають навіть гірше, ніж його відсутність.

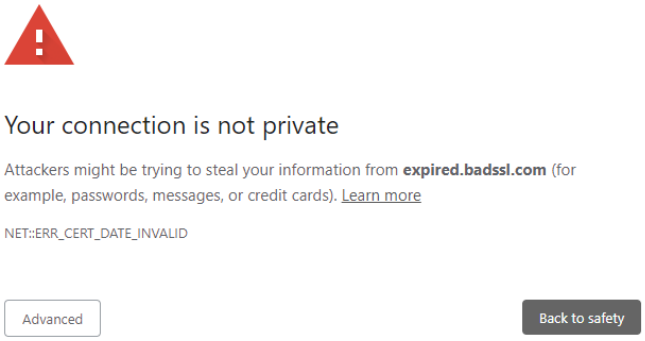

Рисунок 1.1 – Реакція браузера на веб-сайт з недійсним сертифікатом

І навіть один день затримки оновлення сертифікату може обернутись для веб-сайту великими репутаційними і фінансовими втратами. Отож було розроблено багато сервісів для моніторингу SSL-сертифікатів.

Важливість сервісів для автоматизованого моніторингу і оновлення SSL-сертифікатів можна зрозуміти за даними Use Case діаграмами, на яких зображено процедуру ручного моніторингу системним адміністратором SSL-сертифікату і його оновлення, а також автоматизованого – за допомогою реалізованого застосунку:

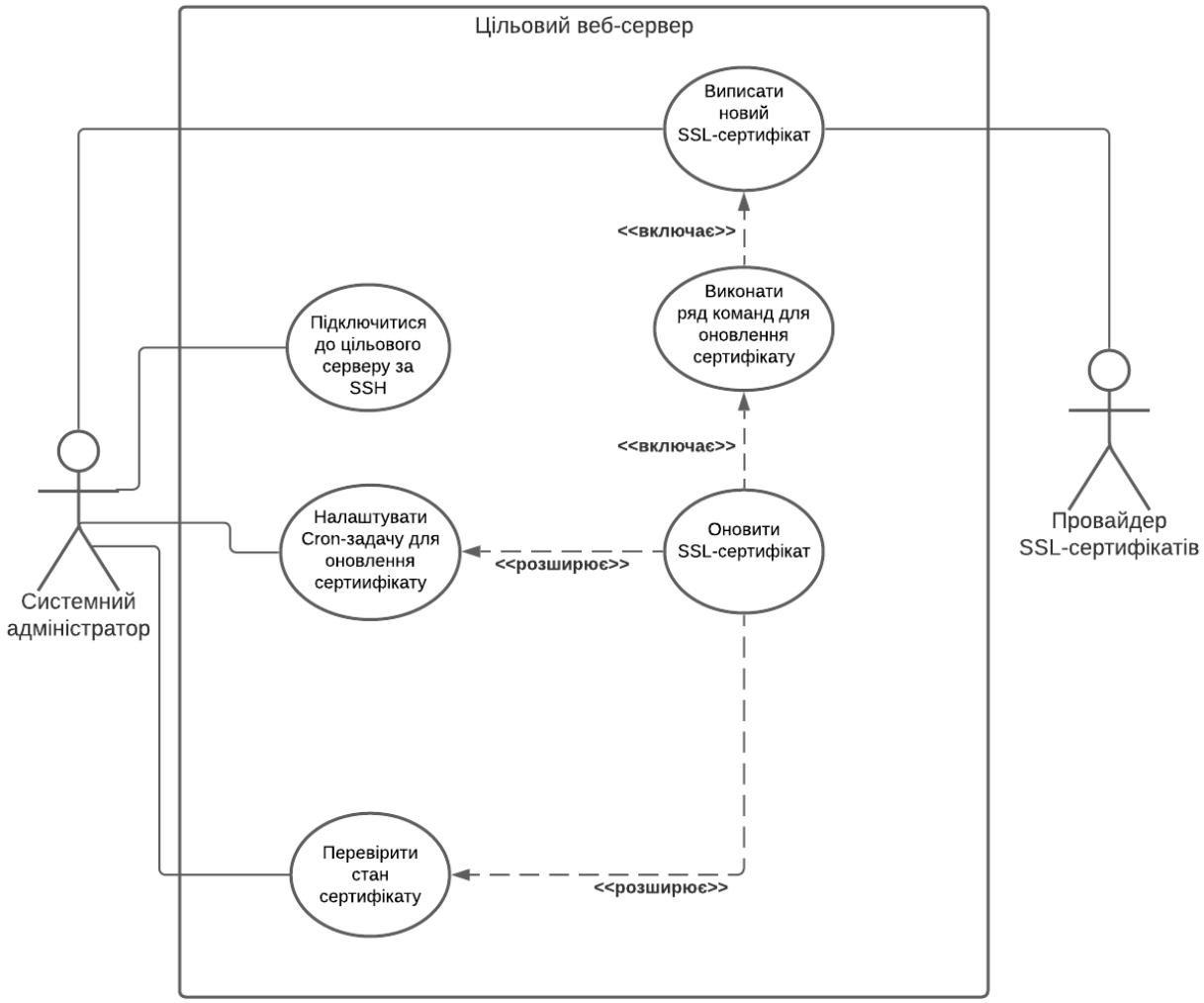

Рисунок 1.2 – Схема дій системного адміністратора для ручного моніторингу і оновлення SSL-сертифікатів

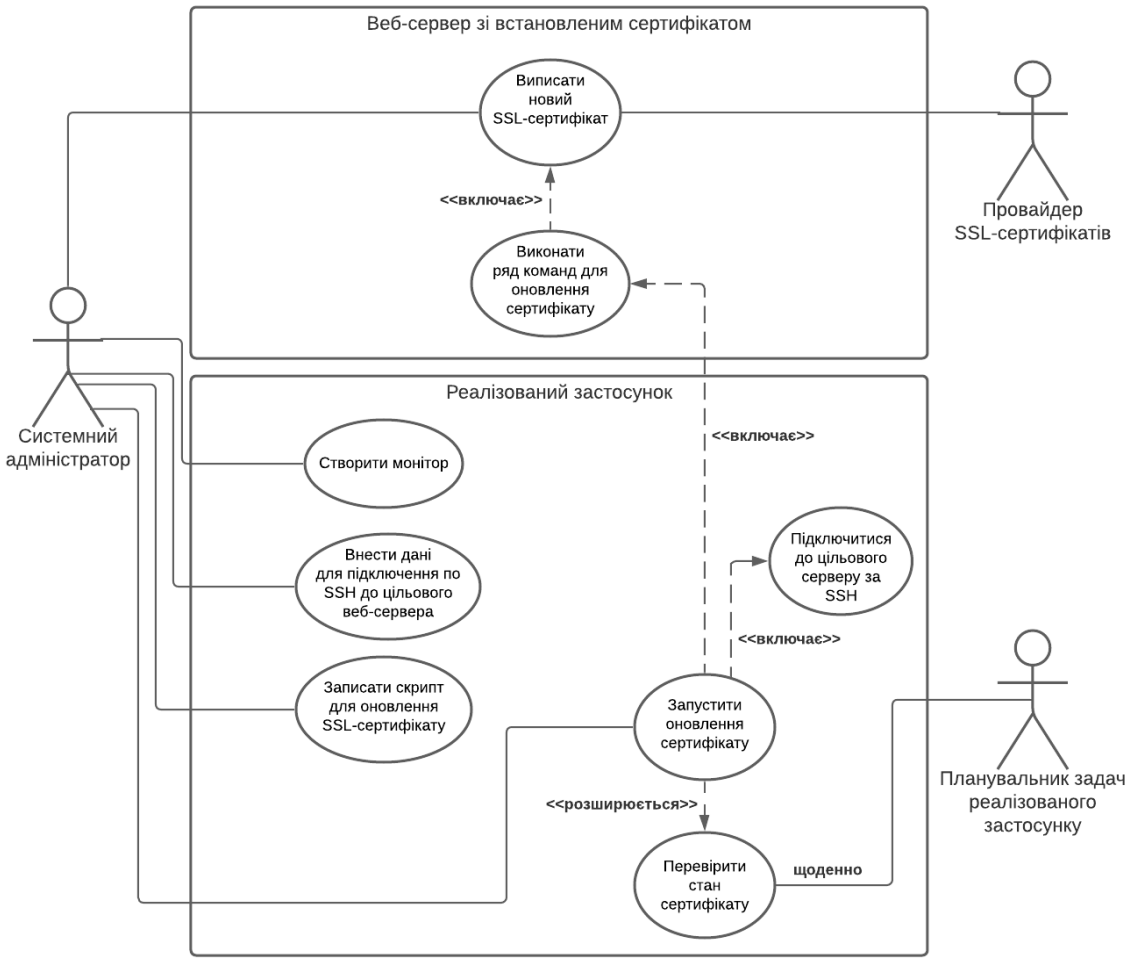

Рисунок 1.3 – Схема дій системного адміністратора для налаштування автоматизованого моніторингу і оновлення SSL-сертифікатів з використанням реалізованого застосунку

Таким чином, без використання реалізованого застосунку, системний адміністратор мав би сам створити і запланувати повторювану задачу (за допомогою Cron або іншого планувальника) для оновлення сертифікату на цільовому сервері, тобто запланувати його оновлення. До того ж, таке рішення не надає жодних сповіщень про виняткові ситуації (як-от неочікувана зміна сертифікату, або ж невдала спроба оновлення). З використанням розробленого застосунку цей процес значно спрощується, оскільки системному адміністратору необхідно лише створити монітор (тобто вказати адресу веб-ресурсу, до якого прив’язаний SSL-сертифікат), а також вказати, як підключитись до цільового серверу і які команди виконати для оновлення. Роботу з планування і виконання

вказаних дій застосунок зробить самостійно, при цьому також надаючи сповіщення на електронну пошту користувача про результати виконання цих дій. Важливо зазначити, що застосунок не створює Cron-задачі (або ж подібні) для оновлення сертифікату на цільовому сервері, а лише виконує вказані користувачем команди, коли настане час оновлення сертифікату.

1.2 Теоретичні відомості

1.2.1 Протокол HTTP

Для того, щоб зрозуміти, як працює захищене з'єднання і для чого необхідні SSL-сертифікати, пропоную розглянути це на прикладі одного з протоколів передачі даних – HTTP. Основним його призначенням є передача веб-сторінок, проте цей протокол використовують і для передачі інших форматів даних. Протокол працює за схемою «запит-відповідь», і для ідентифікації ресурсів використовує глобальні ідентифікатори – URI. Також варто зазначити, що протокол HTTP не має стану, тобто відсутнє збереження стану між двома різними парами «запит-відповідь».

Структура пари «запит-відповідь» така:

- стартовий рядок, що складається з ідентифікатору методу запиту, а також цільового endpoint'у
- заголовки
- тіло повідомлення, що містить дані запиту, результат виконання або опис помилки, якщо запит не виконано успішно

Протокол HTTP має кілька методів запиту, найбільш вживаними з яких є:

- «GET» - здебільшого використовується для отримання інформації
- «POST» - використовується для відправки даних або створення нових, також використовується для ініціалізації певних команд на сервері
- «PUT» - часто використовується для редагування існуючих даних
- «DELETE» - видаляє існуючі дані

- «OPTIONS» - повертає методи HTTP, які підтримує цільовий endpoint

1.2.2 Захищені протоколи передачі даних

Перед тим, як дізнатися про принцип роботи HTTPS, необхідно зрозуміти, що таке SSL. Отже, SSL (або більш новий TLS) – криптографічний протокол, який шифрує дані, що пересилаються з його використанням. Шифрування відбувається за допомогою пари приватного і публічного ключів – унікального набору символів визначеної довжини, які використовуються у хеш-функції, яка власне шифрує дані.

Алгоритм роботи SSL:

1. Клієнт підключається до сервера
2. Сервер відсилає клієнту цифровий сертифікат (SSL-сертифікат), що містить необхідні для встановлення з’єднання дані і публічний ключ
3. Клієнт шифрує випадкове число за допомогою отриманого публічного ключа і відправляє на сервер результат
4. За допомогою приватного ключа сервер розшифровує отримане число
5. З випадкового числа обидві сторони з’єднання створюють ключі для шифрування даних, які будуть передаватись між сторонами

Таким чином, сутностям ззовні з’єднання відомий лише публічний ключ сервера, чого недостатньо для успішного розшифрування даних, що передаються між клієнтом і сервером.

Отже, протокол HTTPS – це той самий HTTP, що працює всередині протоколу SSL. Насправді, за таким же принципом працюють і всі інші захищені версії протоколів передачі даних: FTPS, IMAPS, SMTPS тощо.

1.3 Огляд існуючих рішень

Рішень для моніторингу веб-ресурсів існує безліч, проте дуже багато з них зосереджено саме на моніторингу контенту веб-сайтів, наприклад. Тож для

порівняння і аналізу було вибрано два найбільш відомих сервіси моніторингу саме технічної складової веб-ресурсів: доменного імені, SSL-сертифікатів, швидкодії і т. д.

Oh Dear! [2]

Це платний сервіс для комплексного моніторингу веб-сайтів, який включає моніторинг продуктивності сайту (швидкість обробки запитів), середній час безвідмовної роботи, загальну перевірку працездатності всіх можливих сторінок веб-сайту, а також моніторинг його SSL-сертифікату. Також сервіс дозволяє запланувати періодичну перевірку endpoint'ів: для цього він робить запит на вказаний endpoint і деякий час чекає відповідь.

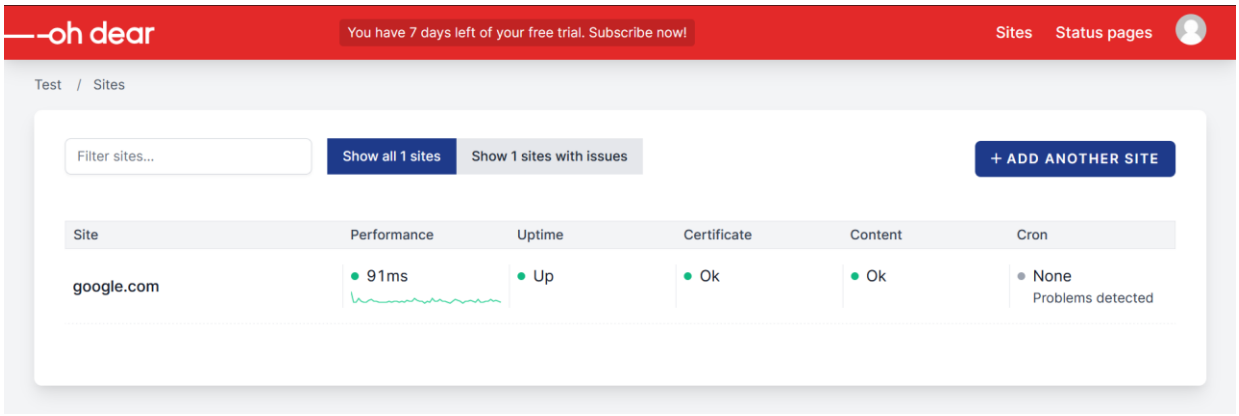

Рисунок 1.4 – Приклад монітору в Oh Dear!

Переваги цього сервісу: широкий функціонал для комплексного моніторингу веб-сайтів, гнучкі налаштування запитів для перевірки роботи сайту, інформативне логування, визначення всіх піддоменів, на які розповсюджується SSL-сертифікат, а також зручний інтерфейс.

Проте, в рамках теми курсової роботи, цей сервіс має значні недоліки: можливо моніторити лише веб-сайти (немає підтримки різних протоколів і типів серверів: FTP, SMTP, IMAP, POP), а також відсутня можливість виконання визначених користувачем команд на цільовому сервері для оновлення SSL-сертифікатів.

Site24x7 [3]

Це платний сервіс для ще більш комплексного моніторингу не тільки веб-сайтів, а і веб-серверів різних типів (HTTP, FTP, поштові сервери, DNS, NTP), хмарних рішень, віртуальних машин. Це універсальний інструмент, який застосовують великі компанії для моніторингу своїх рішень. Окремої уваги заслуговує розділ «Security» - моніторинг захищеності веб-ресурсів, який включає перевірку SSL-сертифікатів, доменних імен, змін в контенті веб-сайту та навіть репутації бренду.

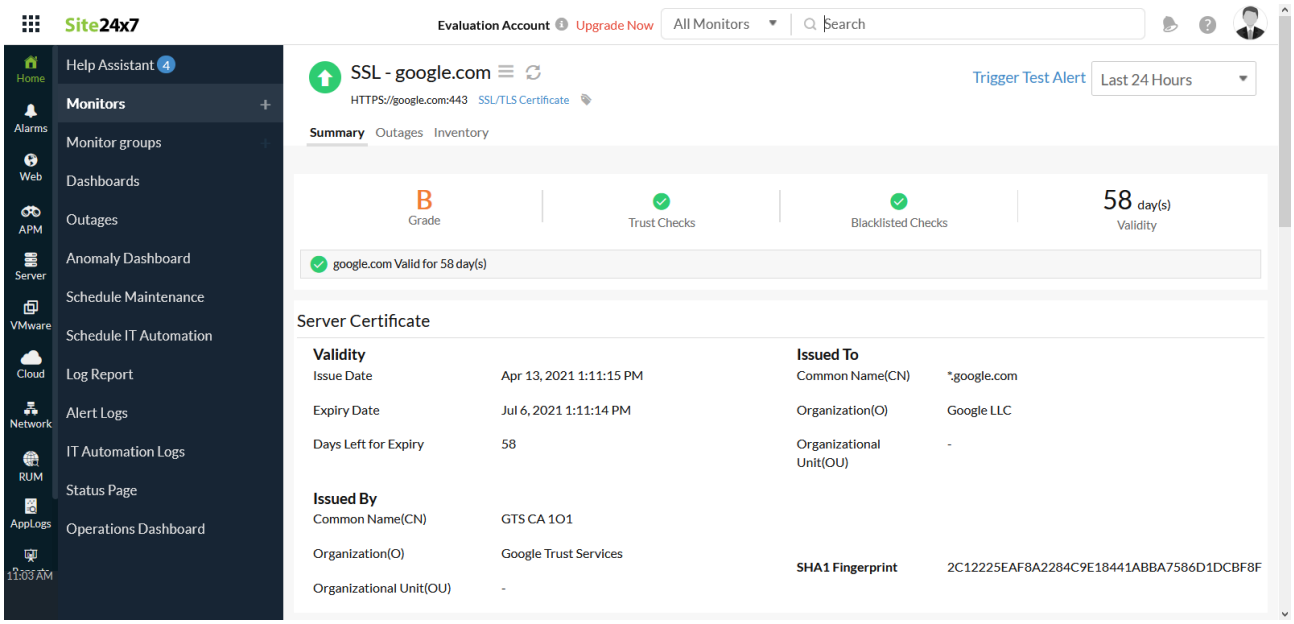

Рисунок 1.5 – Приклад монітору в Site24x7

З переваг даного сервісу можна виділити підтримку будь-яких типів серверів, багату функціональність і велику гнучкість налаштування.

З недоліків, знову ж, сервіс, на жаль, не підтримує автоматизоване виконання визначених користувачем команд на віддаленому сервері, а також має складний інтерфейс.

Обидва ці сервіси досить потужні у тому, для чого вони призначені. Проте, у цих сервісів, як і у всіх інших, відсутня можливість якимось чином оновити SSL-сертифікат, що моніториться, хоч це і можливо зробити не для всіх сертифікатів.

1.4 Постановка задачі

Отож проаналізувавши існуючі рішення для моніторингу веб-ресурсів, було поставлено такі завдання курсової роботи:

- Визначення необхідних функціональних можливостей застосування
- Дослідження технологій та засобів розробки, які відповідають вимогам застосування та у повній мірі дозволяють реалізувати необхідні функції
- Реалізація застосування для автоматизованого моніторингу і оновлення SSL-сертифікатів сервера. Врахувавши вищеперераховані існуючі рішення, було висунуто такі вимоги:
    - Можливість додавання сертифікату для моніторингу за доменним іменем (хостом) і портом (для моніторингу різних типів веб-серверів), або протоколом
    - Щоденна автоматична перевірка всіх доданих сертифікатів, щоб переконатись, що їх термін дії ще не вичерпаний
    - Сповіщення на пошту про сертифікати, термін дії яких підходить до кінця, а також, якщо налаштоване автоматичне оновлення, сповіщення про успішне або невдале оновлення
    - Можливість налаштувати оновлення сертифікату: надати дані для підключення до цільового сервера по SSH, а також набір команд для виконання
    - Можливість налаштувати автоматичне оновлення, при цьому не виключаючи можливість ручного запуску оновлення в будь-який час
    - Детальне логування виконання наданого користувачем скрипту для оновлення сертифікату

РОЗДІЛ 2

2.1 Обґрунтування вибраних технологій

На сьогодні існує кілька найпопулярніших мов програмування і фреймворків для розробки серверної частини застосунків з клієнт-серверною архітектурою: C#, Java, Python, JavaScript, і ASP.NET Core, Spring, Django, NodeJS відповідно. Тож для реалізації бекенду застосунку було прийнято рішення використати платформу .NET від компанії Microsoft, а саме мову програмування C# і фреймворк ASP.NET Core, оскільки, за даними офіційних тестів, фреймворк має найбільшу швидкодію серед усіх зазначених [4]. Також такий вибір було зроблено спираючись на власний досвід розробки з використанням даного фреймворку. Для фреймворку ASP.NET Core також розроблено досить багато інструментів. Один з них – ORM EntityFramework Core, що дозволяє працювати з базою даних в застосунку на рівні абстрактних команд без використання мови запитів SQL. Також цей фреймворк надає кілька підходів до створення баз даних і їх підтримки, найзручнішим з яких є «Code First» - він надає можливість створити сутності «в коді», а потім за цими даними автоматично створити базу даних.

Також варто зазначити, що фреймворк ASP.NET Core пропонує кілька підходів для створення застосунку: Web MVC і Web API. Між цими підходами є кардинальна різниця. Архітектура застосунку, реалізованого за першим підходом, є дуже схожою на патерн програмування MVC, тобто бекенд працює як з даними, так і з представленням. Для цього фреймворк ASP.NET Core надає рушій для створення сторінок представлення – Razor Web Pages. Натомість, застосунок, реалізований за підходом Web API, працює лише з даними, повністю віддаючи відповідальність за реалізацію представлення розробнику. Отже, був обраний саме підхід Web API, оскільки підхід Web MVC вважається застарілим і не є зручним у сучасних реаліях, коли більшість веб-застосунків є односторінковими.

Для реалізації представлення, тобто фронтенду застосунку, була вибрана бібліотека з відкритими вихідними кодами ReactJS від компанії Facebook. З-поміж інших бібліотек для створення односторінкових веб-застосунків вона виділяється зручністю використання і швидкодією, які здобуваються завдяки технології JSX (JavaScript XML), що дозволяє використовувати мову розмітки HTML напряму в коді JavaScript, а також компонентному підходу, що дозволяє виділяти повторювані частини застосунку в окремі компоненти, які можна використовувати повторно. Також робота бібліотеки ReactJS заснована на технології Virtual DOM, що дозволяє працювати з віртуальним деревом компонентів DOM замість реального, завдяки чому зростає швидкодія застосунку. І нарешті, бібліотека ReactJS має дуже велику спільноту розробників, завдяки чому існує багато бібліотек компонентів, деякі з яких навіть підтримують і використовують всесвітньо відомі компанії.

У якості СКБД була вибрана PostgreSQL з відкритим вихідним кодом. Її перевагами є: зручність використання, яка забезпечується потужними додатками (з графічним інтерфейсом – pgAdmin, і консольний застосунок – PSQL), а також швидкодія. Варто зазначити, що використана база даних є реляційною. Також на цей вибір вплинув наявний досвід розробки з використанням даної СКБД і наявність налаштованих інструментів для резервного копіювання баз даних.

2.2 Розробка бекенду застосунку

Бекенд – це та частина застосунку, яка працює на сервері, містить бізнес-логіку, взаємодіє з базою даних і реагує на запити користувача з клієнтської частини (фронтенду). І як вже було описано вище, обрана реалізація застосунку з використанням клієнт-серверної архітектури, тобто клієнт (фронтенд) і сервер (бекенд) повністю розділені і зв’язуються між собою за допомогою запитів, реалізованих за архітектурою REST API.

2.2.1 Архітектурні особливості бекенду застосунку

В основу структури бекенду лягла так звана «Чиста архітектура» Роберта Мартіна [5]. Це загальна архітектура для побудови додатків, яка дозволяє розділити ключові компоненти системи на окремі рівні таким чином, щоб кожен рівень залежав лише від тих, які йому насправді необхідні для роботи.

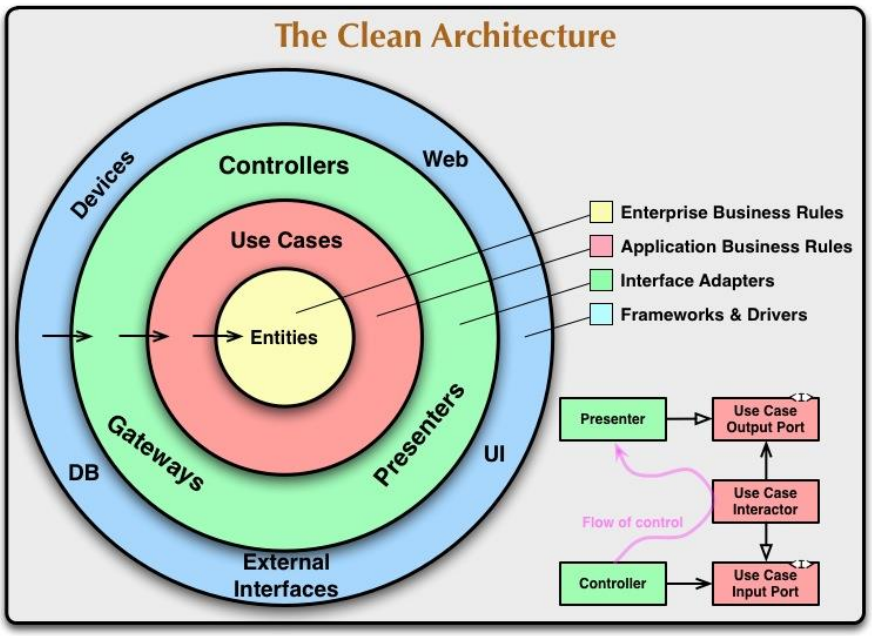

Рисунок 2.1 – Діаграма «Чистої архітектури» Роберта Мартіна

Отже, на цій діаграмі зображено кілька рівнів, які взаємодіють між собою:

- Entities – найважливіший рівень, що містить опис сутностей системи

- Use Cases – рівень, що залежить від Entities, і містить бізнес-логіку застосунку

- Controllers/Presenters/Gateways – рівень, що слугує «мостом» між рівнем UI і рівнем бізнес-логіки

- Web/UI/DB – рівень, на якому працює клієнтська частина застосунку – UI або Web, що відсилає до контролерів команди користувача і працює з даними, які отримав від попереднього рівня. Також на цьому рівні відбувається робота з базою даних, адже це окрема від застосунку сутність, яка лише зберігає дані

Велика перевага цієї архітектури в тому, що вона виключає, до прикладу, пряму взаємодію рівня UI і бізнес-логіки застосунку, що у сучасних застосунках може призвести до швидкого зниження якості коду і темпів розробки.

Таким чином, для реалізованого застосунку було створено таку структуру проектів:

- Domain – проект, що містить лише опис сутностей
- Application – проект, що містить обробники бізнес-запитів користувача, а також бізнес-логіку застосунку
- API – проект, що містить налаштування застосунку, а також контролери, які прийматимуть запити від клієнтської частини
- Persistence – проект, що містить налаштування бази даних
- Infrastructure – додатковий проект, що містить реалізацію і опис логіки, пов’язаної з безпекою застосунку, тобто аутентифікацію і авторизацію

Маємо таку діаграму залежностей цих проектів:

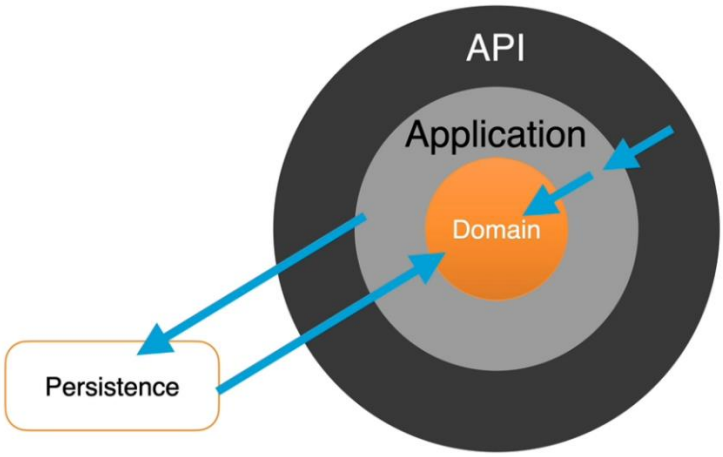

Рисунок 2.2 – Діаграма залежностей проектів бекенду застосунку

І завдяки транзитивності, проект API, наприклад, буде в змозі використовувати сутності, що визначені в проекті Domain, хоча прямої залежності між цими проектами немає.

Далі розглянемо взаємодію лише проектів API і Application. Оскільки необхідно досягти повного розмежування відповідальностей між цими

проектами, розглянемо кілька патернів проектування, які використані у реалізації застосунку.

Перший використаний патерн – CQRS [6] – Command Query Responsibility Sharing – патерн проектування, який має на меті розділення логіки на запити (запити, які призначені лише для отримання даних) і команди (запити, результатом яких є створення нових даних, редагування існуючих даних, видалення даних або ж запуск якогось процесу).

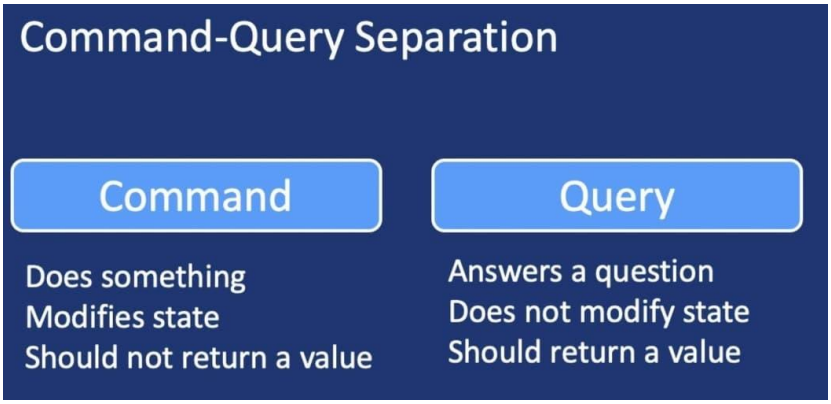

Рисунок 2.3 – Розділення логіки на команди і запити за патерном CQRS

Цей патерн також є корисним у тому випадку, коли застосунок використовує різні бази даних, окремо оптимізовані для читання існуючої інформації і запису нової, оскільки дозволяє чітко визначити цикл роботи з даними для кожного запиту.

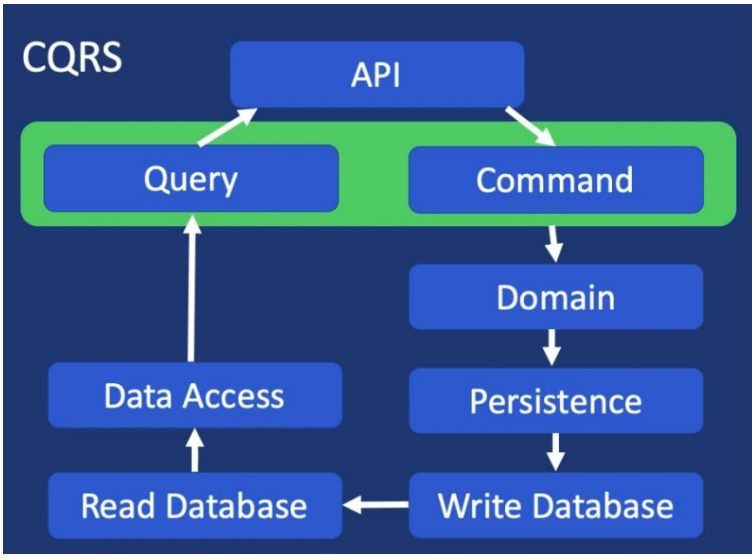

Рисунок 2.4 – Цикл роботи з даними з двома БД за патерном CQRS

Другий патерн – Mediator [7] – поведінковий патерн проектування, який дозволяє розділити логіку отримання запиту і його обробки. Його суть в тому, що отримавши запит, формується деякий об’єкт з інформацією запиту і відправляється до обробника цього запиту, який в свою чергу після його виконання повертає об’єкт, що містить результат виконання запиту.

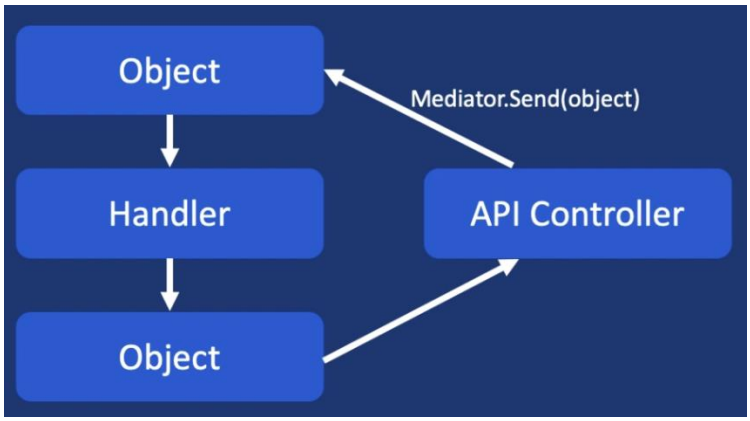

Рисунок 2.5 – Приклад обробки запиту за патерном Mediator

Саме цей патерн дозволяє зручно розділити відповідальність проектів API і Application реалізованого застосунку.

2.2.2 Опис сутностей

Для роботи з SSL-сертифікатами і збереження їх в базу даних недостатньо мати саму сутність «Сертифікат», оскільки для роботи застосунку, а точніше, наприклад, для реалізації оновлення сертифікатів, вимагається збереження додаткових даних. Отож було прийнято рішення створити додаткову сутність «Монітор», яка має посилання на сертифікат, що моніториться.

Складові сутності «Монітор»:

- Id – ідентифікаційний код (первинний ключ в базі даних)
- DisplayName – ім’я монітору для зручної ідентифікації
- DomainName – доменне ім’я, за яким було зроблено запит для отримання сертифікату
- Port – порт веб-сервера

- CreationDate – дата створення монітору
- Certificate – об’єкт SSL-сертифікату (в базу даних записується посилання на нього – зовнішній ключ)
- AutoRenewalEnabled – чи увімкнене автоматичне оновлення сертифікату
- LastCheckDate – дата останньої перевірки сертифікату
- Дані SSH цільового серверу для оновлення сертифікату (Hostname, Port, Username, PrivateKey, Password)
- SshConfigured – чи було налаштовано доступ по SSH
- RenewalScript – набір команд для оновлення сертифікату
- RenewalConfigured – чи було налаштовано оновлення сертифікату
- IsInRenewalQueue – чи знаходиться сертифікат в черзі на оновлення
- IsRenewing – чи оновлюється сертифікат в даний момент часу
- LastRenewalDate – дата останньої спроби оновлення
- LastRenewalLogs – результат виконання команд для оновлення сертифікату
- WasRenewalSuccessful – чи була остання спроба оновлення сертифікату успішною
- RenewalErrorCode – якщо остання спроба оновлення сертифікату була безуспішною, містить код помилки
- LastSshConnectionCheckDate – остання дата перевірки з’єднання SSH

Сутність також включає службові поля, як до прикладу – посилання на сутність користувача, що володіє даним монітором. Також варто зазначити, що кожен монітор посилається на унікальний сертифікат. Звісно, це не є ефективно, оскільки багато моніторів можуть моніторити один і той самий сертифікат, проте цьому є пояснення. Справа в тому, що це рішення простіше з точки зору реалізації, а в перспективі, навіть краще у швидкодії. Оскільки у застосунку передбачена можливість оновлення сертифікатів, тобто їх зміну, у випадку одного сертифікату для всіх моніторів необхідно було б змінювати Id

сертифікату у всіх моніторах, що не є зручно. Також, що більш важливо, при видаленні монітору на даний момент каскадно видаляється і сертифікат, на який посилається монітор. У випадку з одним сертифікатом для багатьох моніторів механізм їх видалення було б реалізувати значно складніше.

Наступна сутність описує здобутий SSL-сертифікат. Її складові:

- Id – ідентифікаційний код (первинний ключ в базі даних)
- SubjectCommonName – назва суб’єкту сертифікату або доменне ім’я, на яке виписаний сертифікат
- SubjectOrganization – назва організації, на яку виписаний SSL-сертифікат
- IssuerCommonName – назва ресурсу, що виписав сертифікат
- IssuerOrganization – назва організації, що виписала сертифікат
- ValidFrom – дата початку терміну дії сертифікату
- ValidTo – дата кінця терміну дії сертифікату
- Version – номер версії SSL-сертифікату
- SerialNumber – унікальний серійний номер SSL-сертифікату

Остання сутність – AppUser – є розширенням наданої фреймворком ASP.NET Core сутності User (користувач), яка містить необхідні дані про користувача застосунку, як-то його поштову адресу або ім’я. Складові сутності AppUser:

- DisplayName – повне ім’я користувача
- RegistrationDate – дата реєстрації користувача
- Monitors – колекція моніторів, які створив даний користувач (в базі даних цього поля немає, натомість у кожного монітора є зовнішній ключ, що містить ідентифікаційний код користувача)
- NotificationsEmail – поштова адреса для отримання сповіщень
- NotifyAboutCertificateChange – чи сповіщати користувача про неочікувану зміну сертифікату в моніторі
- ExpiryNotificationThresholdDays – номер доби до закінчення терміну дії сертифікату, починаючи з якої система буде надсилати сповіщення

- NotifyAboutExpiryIfRenewalConfigured – чи надсилати користувачу сповіщення про закінчення терміну дії сертифікату, якщо він має налаштоване оновлення
- RenewalThresholdDays – номер доби до закінчення терміну дії сертифікату, починаючи з якої система буде намагатися оновити сертифікат

2.2.3 Поняття сервісу в ASP.NET Core

Для роботи з залежностями в проекті (це особливо важливо для реалізації даного застосунку, оскільки він має специфічну архітектуру з кількох проектів), існує механізм Dependency Injection, що дозволяє зробити об’єкти, що взаємодіють в застосунку, слабко зв’язними. Такі об’єкти зазвичай пов’язані між собою абстракціями – інтерфейсами, що робить систему більш гнучкою і зрозумілою. Для роботи з такими залежностями існують контейнери залежностей – IoC-контейнери (Inversion of Control – інверсія контролю). В сучасній версії фреймворку ASP.NET Core такий контейнер представлений інтерфейсом IServiceProvider, а самі залежності називають сервісами. Таким чином, під час запуску створюється і конфігурується контейнер сервісів, які можна буде використовувати в подальшій роботі. Прикладом сервісу є DataContext – сервіс, що створений для доступу до бази даних, який надає багато можливостей читання, запису і інших дій над даними в базі даних.

Також варто ввести поняття життєвого циклу залежностей (сервісів). Тож в ASP.NET Core існує три типи сервісів:

- Singleton – протягом часу роботи застосунку існує лише один екземпляр сервісу і його ініціалізація відбувається лише один раз
- Scoped – для кожного запиту до застосунку створюється новий екземпляр сервісу
- Transient – при кожному зверненні до сервісу створюється новий його екземпляр

Таким чином, при розробці застосунку можна обирати тип створюваних розробником сервісів. До прикладу, сервіс, що надає доступ до бази даних – DataContext – є виду Scoped, тобто для кожного запиту до застосунку створюється новий екземпляр сервісу.

Великою перевагою сучасних версій фреймворку ASP.NET Core є те, що розробнику надається можливість створення власних сервісів, в які можна вивести якісь загальні дії, що регулярно виконуються під час роботи застосунку.

2.2.4 Реалізація обробки запитів, приклади «запитів» і «команд»

В ASP.NET Core для обробки запитів існують контролери – сутності, які відповідають за окремі частини публічного інтерфейсу застосунку. У створеному застосунку є кілька таких контролерів:

- BaseController – батьківський для всіх інших контролер
- MonitorsController – контролер, що відповідає за роботу з моніторами: створення, отримання, видалення, а також за супутні дії
- UserController – контролер, що відповідає за роботу з користувачами: аутентифікація, реєстрація, робота з ключами доступу

Перш ніж надавати приклад реалізації контролеру, варто згадати, що для обробки запитів реалізований проект Application, а також що запити поділяються на звичайні запити і команди. Тож спочатку розглянемо реалізацію обробника звичайного запиту. Це клас, що має кілька вкладених класів: Query і Handler. Клас Query містить поля, що надаються в якості вхідних параметрів при виконанні запиту. Клас Handler містить реалізацію обробки запиту. Для цього у ньому є конструктор, в якому можна отримати необхідні для обробки запиту сервіси, а також метод Handle, що приймає екземпляр класу Query.

```
public class Details
{
    public class Query : IRequest<MonitorDto>
    {
        public Guid Id { get; set; }
    }

    public class Handler : IRequestHandler<Query, MonitorDto>
    {
        private readonly DataContext _context;
        private readonly IMapper _mapper;

        public Handler(DataContext context, IMapper mapper)
        {
            _context = context;
            _mapper = mapper;
        }

        public async Task<MonitorDto> Handle(Query request,
CancellationToken cancellationToken)
        {
            var monitor = await _context.Monitors
                .Where(m => m.Id == request.Id)
                .Include(m => m.Certificate)
                .SingleOrDefaultAsync();

            if (monitor == null)
                throw new RestException(HttpStatusCode.NotFound,
ErrorType.MonitorNotFound);

            var monitorToReturn = _mapper.Map<Monitor, MonitorDto>(monitor);

            return monitorToReturn;
        }
    }
}
```

Приклад 2.1 – Реалізація обробки запиту на отримання даних монітору

Далі розглянемо реалізацію обробника команди. Це клас, що має кілька вкладених класів: Command, Handler, а також, інколи, CommandValidator – для валідації отриманих в запиті даних. Клас Command містить поля, що надаються в якості вхідних параметрів при виконанні запиту. Клас Handler містить реалізацію обробки запиту. Метод Handle приймає екземпляр класу Command. Варто зауважити, що якщо в класі запиту є вкладений клас валідатора, що реалізує клас AbstractValidator типу команди, обробка запиту навіть не буде викликана, якщо валідація не буде успішною.

```
public class Create
{
    public class Command : IRequest<MonitorDto>
    {
        public string DisplayName { get; set; }
        public string DomainName { get; set; }
        public int Port { get; set; }
    }

    public class Handler : IRequestHandler<Command, MonitorDto>
    {
        private readonly DataContext _context;
        private readonly ICertificateParser _certificateParser;
        private readonly IMapper _mapper;
        private readonly IUserAccessor _userAccessor;

        public Handler(DataContext context, ICertificateParser
certificateParser, IMapper mapper,
            IUserAccessor userAccessor)
        {
            _context = context; _certificateParser = certificateParser;
            _mapper = mapper; _userAccessor = userAccessor;
        }

        public async Task<MonitorDto> Handle(Command request,
CancellationToken cancellationToken)
        {
            var certificate =
_certificateParser.GetCertificate(request.DomainName, request.Port);

            var user = await _context.Users
                .SingleOrDefaultAsync(x =>
x.UserName.Equals(_userAccessor.GetCurrentUsername()));

            var monitor = new Monitor
            {
                User = user, DisplayName = request.DisplayName,
                DomainName = request.DomainName, Port = request.Port,
                CreationDate = DateTime.Now, Certificate = certificate,
                AutoRenewalEnabled = false, LastCheckDate = DateTime.Now
            };

            _context.Monitors.Add(monitor);

            var success = await _context.SaveChangesAsync() > 0;

            if (success)
                return _mapper.Map<Monitor, MonitorDto>(monitor);
            throw new RestException(HttpStatusCode.InternalServerError,
ErrorType.SavingChangesError);
        }
    }
}
```

Приклад 2.2 – Реалізація обробки команди на створення нового монітору

```
public class CommandValidator : AbstractValidator<Command>
{
    public CommandValidator()
    {
        RuleFor(m => m.DisplayName)
            .NotEmpty()
            .Length(2, 30);
        RuleFor(m => m.DomainName)
            .NotEmpty()
            .Length(4, 30)
            .Must(MonitorPathValidators.BeValidDomainName)
            .WithMessage("Please specify a valid domain name without
protocol.");
        RuleFor(m => m.Port)
            .InclusiveBetween(1, 65535)
            .WithMessage("Please specify a valid port.");
    }
}
```

Приклад 2.3 – Реалізація валідатора команди на створення нового монітору

Отже, можемо бачити, що вся необхідна бізнес-логіка використовується саме в обробниках запитів. Таким чином, залишається отримати запит і повернути результат. Саме це відбувається у методах контролерів, які зв'язуються з обробниками запитів через сервіс Mediator, передаючи об'єкт класу відповідного запиту або команди.

```
[HttpGet("{monitorId}")]
[Authorize(Policy = "IsMonitorOwner")]
public async Task<ActionResult<MonitorDto>> Details(Guid monitorId)
{
    return await Mediator.Send(new Details.Query {Id = monitorId});
}
```

Приклад 2.3 – Реалізація методу контролера для отримання даних монітора

```
[HttpPost]
public async Task<ActionResult<MonitorDto>> Create(Create.Command command)
{
    return await Mediator.Send(command);
}
```

Приклад 2.4 – Реалізація методу контролера для створення монітора

2.2.5 Система обробка помилок

Оскільки застосунок, що реалізований в рамках курсової роботи, тісно пов’язаний з системним і мережевим адмініструванням, тобто призначений для системних адміністраторів, було прийнято рішення реалізувати інформативну систему обробки помилок, яка дозволила б користувачам застосунку розуміти причину помилок, що можуть трапитись протягом його роботи.

Фреймворк ASP.NET Core надає власну реалізацію обробки помилок. Таким чином, якщо під час роботи застосунку сталась виняткова ситуація, застосунок продовжує працювати, але фіксує помилку і сповіщає про неї користувача, виконання запиту якого привело до помилки.

Отже, було реалізовано додатковий клас виняткової ситуації – RestException – який відповідає за передбачені помилки, як-то помилки валідації, або ж помилки збереження даних. Такі помилки відрізняються від непередбачених тим, що для них реалізовані спеціальні коди. Ось деякі з них:

- SavingChangesError – помилка збереження даних в базі даних
- MonitorNotFound – монітор за наданими параметрами не знайдено
- RefreshTokenExpired – вичерпано час дії ключа оновлення
- CertificateParsingError – помилка парсингу SSL-сертифікату
- SshConnectionTestingTimeout – перевищена кількість запитів на тестування SSH-з’єднання за одиницю часу

Конструктор даного класу приймає як HTTP-код, так і один з вищезазначених. Це зроблено для того, щоб позначити відповідь на HTTP-запит правильним HTTP-кодом, а також включити в тіло відповіді власний код і, можливо, текстове пояснення помилки.

Список і коди реалізованих помилок продубльований в реалізації фронтенду застосунку. Таким чином, після того, як станеться виняткова ситуація, користувач отримає детальне пояснення помилки. Також цей підхід зручний тим, що тексти помилок можна змінювати в залежності від вибраної користувачем мови.

2.2.6 Реалізація аутентифікації і авторизації користувачів

Реалізований застосунок включає можливість створення користувачів. Тобто застосунком можуть користуватися кілька системних адміністраторів одночасно: створювати монітори, парсити SSL-сертифікати і оновлювати їх.

Варто зазначити, що фреймворк ASP.NET Core надає власну реалізацію аутентифікації і авторизації користувачів, яку можна одразу ж включити до проекту під час його початкового створення. Проте, враховуючи вибрану архітектуру застосунку, було прийнято рішення реалізувати зазначену функціональність самотужки.

Отож, спочатку був модифікований проект Domain, а саме – додана сутність AppUser, яка розширює надану фреймворком сутність User. Також був створений додатковий проект Infrastructure – він містить логіку, пов’язану з аутентифікацією і авторизацією користувачів, а саме: кілька службових сервісів, а також логіку обмеження IsMonitorOwnerRequirement – для можливості доступу користувачів лише до власних моніторів.

Після успішної аутентифікації користувача, система генерує для нього два унікальних ключа: ключ аутентифікації і ключ оновлення. Перший ключ є типу JWT – Json Web Token – стандартна реалізація ключа аутентифікації. Зазвичай ключі цього типу діють дуже короткий проміжок часу – близько 10-15 хвилин. Це означає, що після закінчення терміну дії ключа, користувачу доведеться знову проходити аутентифікацію, що не є зручно. Для цього в систему доданий ще один ключ – ключ оновлення (Refresh Token). Зазвичай термін дії такого ключа становить 30 діб. Він призначений для випуску нових ключів аутентифікації. Таким чином, після завершення терміну дії ключа аутентифікації, автоматично відбувається запит на отримання нового, що значно подовжує сесію користувача. Додатково, подвійний ключ значно посилює захист користувачів і надає можливість заблокувати користувача у разі порушення правил користування застосунком, до прикладу.

2.2.7 Реалізація парсингу SSL-сертифікату сервера

Парсинг SSL-сертифікату – ключова особливість застосунку. Щоб мати можливість моніторити термін дії сертифікату, ми маємо здобути інформацію про нього. Таким чином, необхідно створити сервіс, який буде повертати дані сертифікату за хостом веб-ресурсу, що включає доменне ім’я і порт.

Справа в тім, що платформа .NET не надає прямої можливості отримати SSL-сертифікат цільового ресурсу за його хостом. Єдиний спосіб його отримати – зробити порожній запит до цільового ресурсу і в ході його обробки, а точніше, перевірки дійсності сертифікату, записати інформацію про SSL-сертифікат в окрему змінну.

Тож спочатку парсинг сертифікату було реалізовано з використанням стандартних можливостей платформи .NET – класу HttpClient і його методів. Створюється HTTP-клієнт з отриманими даними підключення, на подію «Валідація SSL-сертифікату» HTTP-клієнта додається власний обробник – він записує об’єкт отриманого сертифікату у змінну і успішно валідує його, оскільки ця перевірка нам не потрібна. Далі встановлюється з’єднання з цільовим ресурсом і виконується порожній запит. Проте, пізніше цей метод парсингу SSL-сертифікату було відкинуто, оскільки він має значний недолік: можливість роботи лише з HTTP-серверами (тобто, неможливо отримати SSL-сертифікат файлового серверу, до прикладу).

Тож було прийнято рішення змінити реалізацію парсингу SSL-сертифікатів за допомогою HTTP-клієнту на більш низькорівневу – за допомогою TCP-клієнту. Алгоритм парсингу SSL-сертифікату принципово не змінився:

1. Створюється TCP-клієнт з отриманими даними підключення
2. Відкривається SSL-потік, до якого доданий власний обробник події «Валідація SSL-сертифікату»
3. Відбувається аутентифікація, під час якої успішно валідується і записується в окрему змінну сертифікат

4. З’єднання закривається

Для парсингу SSL-сертифікатів було створено окремий сервіс типу Singleton, екземпляр якого існує лише один на весь час роботи застосунку.

Наразі, у даної реалізації парсингу SSL-сертифікатів є кілька обмежень:

- заборонені перенаправлення з цільової адреси на іншу
- немає підтримки Cancellation Token’ів, що, при проблемах доступу до цільового веб-ресурсу, створює затримку виконання запиту до одної хвилини

2.2.8 Реалізація SSH-бота для виконання команд на віддаленому сервері

Для реалізації оновлення SSL-сертифікату необхідно мати можливість автоматизовано підключатись до цільового веб-сервера за SSH і виконувати ряд команд, що призведе до створення нового сертифікату. Для цього необхідно реалізувати SSH-клієнт з такими вимогами:

- можливість підключення до веб-сервера за наданими даними SSH (для аутентифікації необхідно мати можливість використовувати як звичайний пароль, так і приватний SSH-ключ)
- можливість виконання консольних команд
- можливість отримати результат виконання команди
- наявність обробки помилок

Також бажано, щоб клієнт підтримував підключення і виконання команд на серверах з різними операційними системами (Windows, Unix).

Всім цим вимогам задовольняє бібліотека Chilkat.SSH, яка і була використана в реалізації даного застосунку.

Для вибраної бібліотеки було створено обгортку, яка включає обробку і створення власних виняткових ситуацій. Ось деякі з них:

- SshConnectionError – помилка підключення до серверу (невірна адреса сервера або SSH-порт)

- SshKeyParsingError – помилка читання або аутентифікації приватного SSH-ключа за паролем
- SshAuthenticationError – помилка аутентифікації за паролем
- SshChannelOpeningError – помилка відкриття каналу для виконання команди
- SshCommandExecutionError – помилка виконання команди
- SshChannelTimeout – вичерпано час очікування результату виконання команди

Також, у даної бібліотеки є деякі особливості, які зумовлені процесом виконання команд. Одна з них полягає в тому, що для виконання кожної команди необхідно відкривати окремий канал сесії в межах одного з'єднання, оскільки SSH-сервер закриває канал після виконання команди.

Алгоритм підключення і виконання команд з використанням Chilkat.SSH такий:

1. Підключення до цільового сервера за наданою адресою і портом
2. Аутентифікація за приватним SSH-ключем з паролем, або ж лише за паролем
3. Відкриття окремого каналу сесії
4. Відправка команди на виконання
5. Закриття каналу сесії
6. Отримання результату виконання команди
7. Закриття з'єднання з сервером

2.2.9 Реалізація виконання фонових задач

2.2.9.1 Обґрунтування вибору системи виконання фонових задач

Планувальник фонових задач – одна з ключових особливостей застосунку. Такий планувальник потрібен для того, щоб мати деяку чергу, в яку будуть надходити дії, які необхідно виконувати в окремому потоці, як-то перевірка SSL-сертифікату сервера, або ж його оновлення. А далі такі дії мали б послідовно

виконуватись в окремому потоці. При цьому, така система має задовольняти ряду вимог:

- збереження стану черги, тобто підтримка коректного продовження роботи після аварійного завершення роботи застосунку
- підтримка логування результатів виконання дій
- підтримка повторюваних задач

Спочатку планувалось створити таку систему за допомогою технології RabbitMQ. RabbitMQ – це брокер повідомлень. Його задача – приймати повідомлення (бінарні дані), зберігати їх у черзі і передавати їх по-черзі споживачу (виконувачу дій). Таким чином, RabbitMQ дозволяє досягти послідовного виконання запланованих дій, при цьому не перевантажуючи систему. Проте, для системи з використанням RabbitMQ необхідно було б також реалізувати деякий постачальник і планувальник дій на боці бекенду, який би планував щоденну перевірку SSL-сертифікатів, а також їх оновлення. Також необхідно було б реалізувати виконувачі цих дій, які б приймали повідомлення від брокера і передавали б у відповідь результат виконання. У такої системи є досить багато недоліків. По-перше, виконувачам задач необхідно використовувати вже реалізовані у застосунку сервіси і сутності, як-от сервіс для парсингу SSL-сертифікату, що значно посилює зв’язність застосунку і зручність розробки. По-друге, для планування повторюваних задач необхідно реалізовувати додаткову систему. По-третє, необхідно самостійно реалізовувати логування виконання дій. І нарешті, така система не буде мати жодного інтерфейсу для ручного керування: не можна, до прикладу, переглянути, які задачі перебувають у черзі, і які – в обробці.

Натомість була знайдена потужна система планування і виконання фонових задач HangFire, яка призначена саме для застосунків на базі платформи .NET. Її переваги:

- зручний інтерфейс додавання задач в чергу
- можливість планувати повторювані задачі

- наявність окремої бази даних для черги виконання задач, завдяки чому система продовжує роботу з необхідного місця, якщо роботу застосунку було перервано, а також слідкує за часом повторюваних задач і запускає їх миттєво у разі запізнення
- наявність зручної панелі керування з графічним інтерфейсом

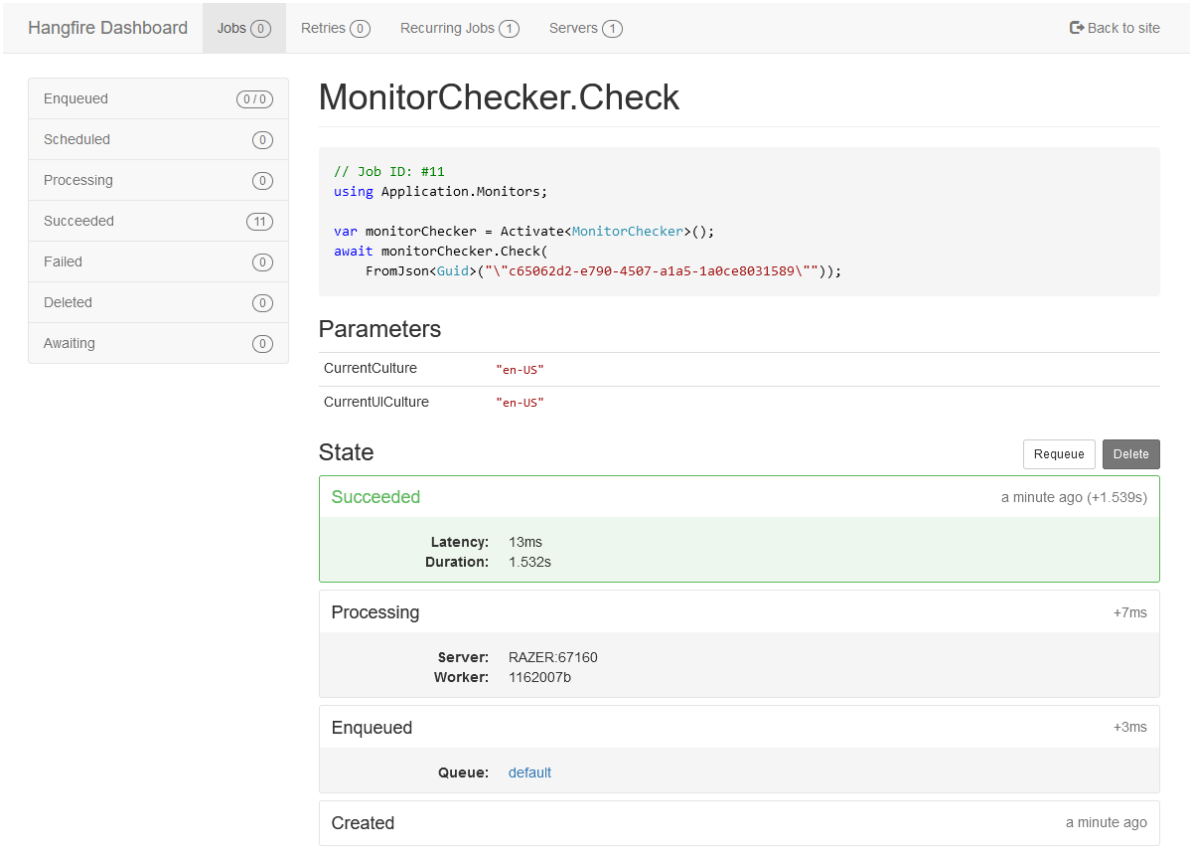

Рисунок 2.6 – Графічний інтерфейс системи фонових задач HangFire

Тож було вирішено використовувати саме цю систему для роботи з фоновими задачами.

2.2.9.2 Реалізація автоматизованої щоденної перевірки сертифікатів

Для перевірки збережених в застосунку SSL-сертифікатів було створено сервіс MonitorChecker, що містить 2 основні методи:

- EnqueueCheck – додає монітор з отриманим ідентифікаційним кодом в чергу для перевірки його сертифікату

- EnqueueCheckAfterRenewal – додає монітор з отриманим ідентифікаційним кодом в чергу для перевірки, чи змінився його сертифікат після оновлення

Обидва ці методи звертаються до сервісу BackgroundJobClient, який надається системою фонових задач, для додавання задач в чергу. Для парсингу SSL-сертифікату веб-ресурсу використовується створений сервіс CertificateParser. Після парсингу новий сертифікат порівнюється зі збереженим попередньо і визначається:

- чи був змінений сертифікат
- чи був вичерпаний його термін дії
- чи настав час оновлення сертифікату

Також, згідно з налаштуваннями користувача, за деякими з цих подій можуть бути відправлені електронні листи-сповіщення користувачу.

Варто зазначити, що перевірка всіх збережених сертифікатів відбувається застосунком автоматично і щоденно. Це реалізовано за допомогою згаданої раніше бібліотеки планування задач HangFire, яка надає можливість планування повторюваних задач всередині застосунку.

2.2.9.3 Реалізація відправлення електронних листів

Оскільки моніторинг має на меті також оперативне повідомлення користувача про успішне або неуспішне виконання дій, було прийняте рішення надсилати такі сповіщення на електронну пошту користувача.

Щоб не реалізовувати власний поштовий сервер, оскільки це є доволі складним процесом, який не гарантує стовідсоткову доставку листів, вирішено скористатися сервісом для відправки електронних листів MailGun. Це один з найпопулярніших сервісів розсилки електронних листів, що гарантує доставку листів отримувачу, і, до того ж, є безкоштовним. Також сервіс має бібліотеки для сучасних мов програмування і фреймворків, які включають і ASP.NET Core.

Тож був реалізований сервіс EmailSender, що має метод для відправки листа, що приймає поштову адресу цільового користувача, тему листа, а також

його вміст. Варто зазначити, що бібліотека сервісу MailGun підтримує кілька можливостей форматування тіла листа:

- простий текст
- HTML
- HTML, створений шаблонними рушіями (Razor Web Pages, Liquid)

Оскільки дизайн застосунку, а тим більше – дизайн електронних листів-сповіщень, не є пріоритетом в даній курсовій роботі, було вибрано другий варіант, а саме – створити кілька простих шаблонів електронних листів для попередньо визначених сценаріїв.

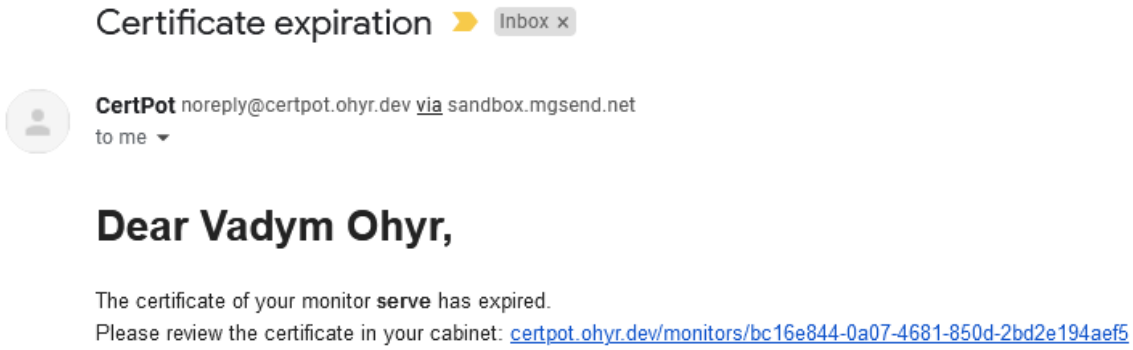

Рисунок 2.7 – Приклад повідомлення про вичерпання терміну дії сертифікату

2.2.9.4 Реалізація оновлення SSL-сертифікатів

Для оновлення SSL-сертифікатів (як ручного, так і автоматичного) було створено сервіс MonitorRenewer, що містить основний метод – EnqueueRenewal, який додає монітор з отриманим ідентифікаційним кодом в чергу для оновлення. Даний сервіс також використовує сервіси BackgroundJobClient і MonitorChecker. Також, у своїй роботі він використовує попередньо описаний SSH-бот, який і виконує команди оновлення сертифікату на цільовому сервері.

Тож алгоритм оновлення сертифікату такий:

1. SSH-бот підключається до цільового серверу за наданими даними SSH
2. Збережений набір команд розділяється на окремі команди
3. По-черзі виконується кожна команда і результат виконання одразу записується в окрему змінну

4. З’єднання закривається
5. В чергу виконання фонових задач додається задача на перевірку оновленого SSL-сертифікату

Якщо ж під час виконання команд виникне виняткова ситуація (виконання команди зайняло дуже багато часу, не вдалося підключитися до сервера тощо), виконання одразу ж зупиниться, в базу даних буде записано інформацію про виконані команди, а також код помилки. Саме для того, щоб в базу даних була записана хоч якась інформація про виконані команди, було вирішено виконувати їх послідовно, а не всі одразу. Таким чином, користувачу буде простіше знайти причину помилки.

2.3 Розробка фронтенду застосунку

2.3.1 Структура проекту

Як вже було зазначено раніше, для реалізації фронтенду застосунку вибрана бібліотека React. Оскільки вона пропонує компонентний підхід для створення користувацького інтерфейсу, доцільно спочатку розробити зручну структуру файлів проекту.

Тож маємо таку структуру каталогів проекту:

- api – містить файли, пов’язані зі створенням запитів до бекенду для відправки і отримання інформації
- components – призначений для файлів компонентів, які є універсальними для всього застосунку, і не містять бізнес-логіки застосунку
- config – містить службові файли конфігурації застосунку
- constants – містить службові константи застосунку
- features – призначений для файлів компонентів, які реалізують бізнес-логіку застосунку, а також для деяких функцій, таких як валідатори даних
- models – містить сутності застосунку, а також реалізацію менеджменту його стану
- screens – призначений для компонентів сторінок застосунку

- styles – містить файли стилів застосунку
- utils – призначений для утилітних функцій, які не містять бізнес-логіки застосунку

2.3.2 Реалізація менеджменту стану застосунку

Для реалізації менеджменту стану застосунку була вибрана бібліотека з відкритим кодом Redux, яка також розроблена компанією Facebook і є найбільш популярною серед сучасних бібліотек менеджменту стану. Її головна перевага полягає у простоті роботи і використання.

Ця бібліотека пропонує декілька основних сутностей [8]:

- action – дія, яка відображає зміну стану
- reducer – функція, яка на вхід отримує action (дію) і змінює стан, засновуючись на дії
- store – сутність, яка містить стан застосунку

Проте, вона має суттєвий недолік – рекомендована реалізація менеджменту стану з використанням цієї бібліотеки вимагає досить великої кількості несуттєвого коду. Для вирішення цієї проблеми існує інша бібліотека – Redux Toolkit, яка є обгорткою для бібліотеки Redux, і до того ж, додає до Redux підтримку асинхронних дій. Замість вищезазначених сутностей, ця бібліотека пропонує одну – slice – сутність, яка суміщає в собі стан, reducer і створені дії.

```
export const createMonitor = createAsyncThunk<IMonitor, INewMonitor>(
  "monitors/createMonitor",
  async (monitor: INewMonitor, { rejectWithValue }) => {
    try {
      const response = await Monitors.create(monitor);
      return response;
    } catch (err) {
      return rejectWithValue(err);
    }
  }
);
```

Приклад 2.4 – Реалізація асинхронної дії для створення нового монітору

```
const monitorsSlice = createSlice({
  name: "monitors",
  initialState,
  reducers: {
    removeMonitor: (state, action: PayloadAction<string>) => {
      state.monitors = state.monitors.filter((m) => m.id !== action.payload);
    },
    switchMonitorAutoRenewal: (state, action: PayloadAction<string>) => {
      const monitor = state.monitors.find((m) => m.id === action.payload);
      if (monitor) {
        monitor.autoRenewalEnabled = !monitor.autoRenewalEnabled;
      }
    },
  },
  extraReducers: (builder) => {
    builder.addCase(fetchMonitors.pending, (state) => {
      state.loading = true;
    });
    builder.addCase(fetchMonitors.fulfilled, (state, { payload }) => {
      state.loading = false;
      state.monitors = payload;
    });
    builder.addCase(fetchMonitors.rejected, (state, { error }) => {
      state.loading = false;
    });

    builder.addCase(createMonitor.pending, (state) => {
      state.submitting = true;
    });
    builder.addCase(createMonitor.fulfilled, (state, { payload }) => {
      state.submitting = false;
      state.monitors.push(payload);
    });
    builder.addCase(createMonitor.rejected, (state, { error }) => {
      state.submitting = false;
    });
  },
});
```

Приклад 2.5 – Реалізація slice’у для списку моніторів

2.3.3 Опис форми для створення монітору і нотатка про доменні імена

В даному підрозділі я хочу розглянути форму для створення нового монітору, оскільки вона має кілька важливих деталей.

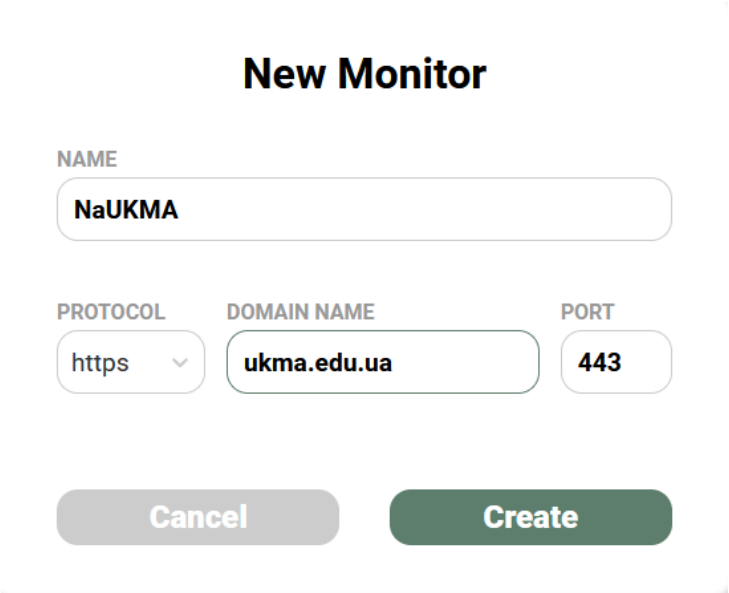

Рисунок 2.8 – Форма створення нового монітору

Як можемо бачити, форма має чотири поля:

- назва монітору
- протокол веб-серверу, SSL-сертифікат якого буде моніторитись
- доменне ім’я або IP-адреса цільового сервера
- порт цільового сервера

З цих полів варто звернути увагу на поле для доменного імені. Справа в тім, що застосунок підтримує роботу з IDN доменними іменами, тобто такими іменами, що складаються не лише з латинських літер.

Рисунок 2.9 – Приклад IDN доменного імені

Як бачимо, у відображенні таких доменних імен можуть бути будь-які символи юнікоду, включаючи емодзі. Додатково, для коректної роботи з такими іменами, у них є латинська альтернатива, яка називається punycode. Підтримка такого виду доменних імен робить розроблений застосунок більш універсальним у сучасних реаліях, коли IDN-домени стрімко набирають популярність.

РОЗДІЛ 3

3.1 Тестування застосунку на реальному прикладі

Потрапивши на головну сторінку застосунку, можемо бачити верхню панель, в якій є посилання на сторінки моніторів і налаштувань, а також дані поточного користувача, або ж лише кнопки «Log in» і «Register».

Рисунок 3.1 – Головна сторінка застосунку

На сторінці моніторів можемо бачити попередньо додані монітори і деяку інформацію про кожен з них:

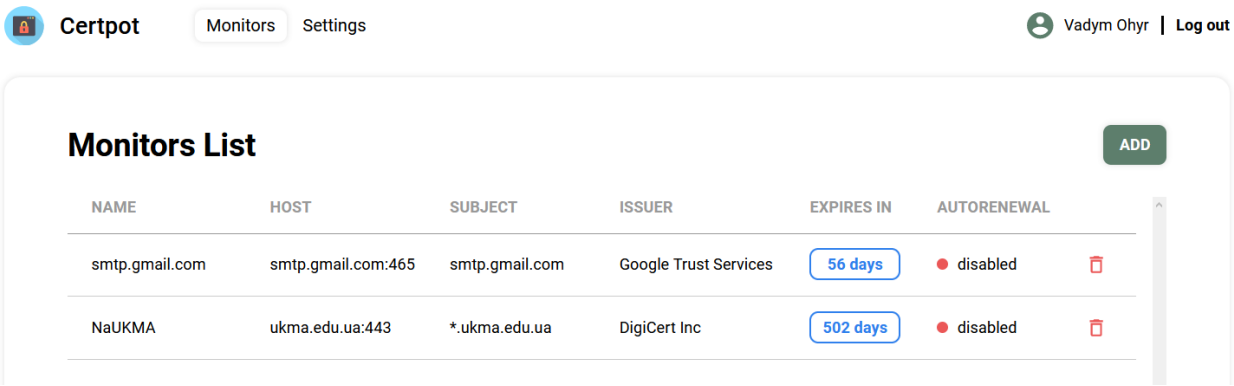

Рисунок 3.2 – Сторінка моніторів

Створимо новий монітор, який буде відповідати за SSL-сертифікат мого особистого веб-сайту – ohyr.dev:

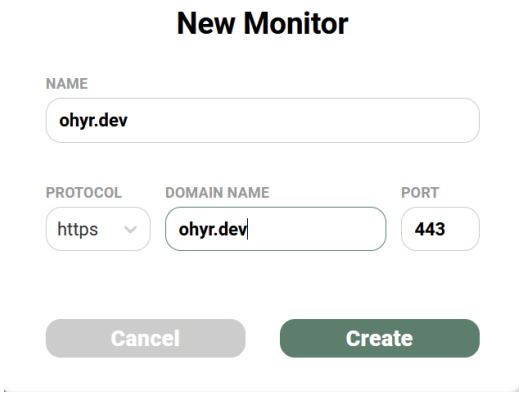

Рисунок 3.3 – Форма для створення нового монітору

Перейдемо на сторінку щойно створеного монітору, де можемо бачити повну інформацію про поточний стан монітору, а також елементи керування монітором.

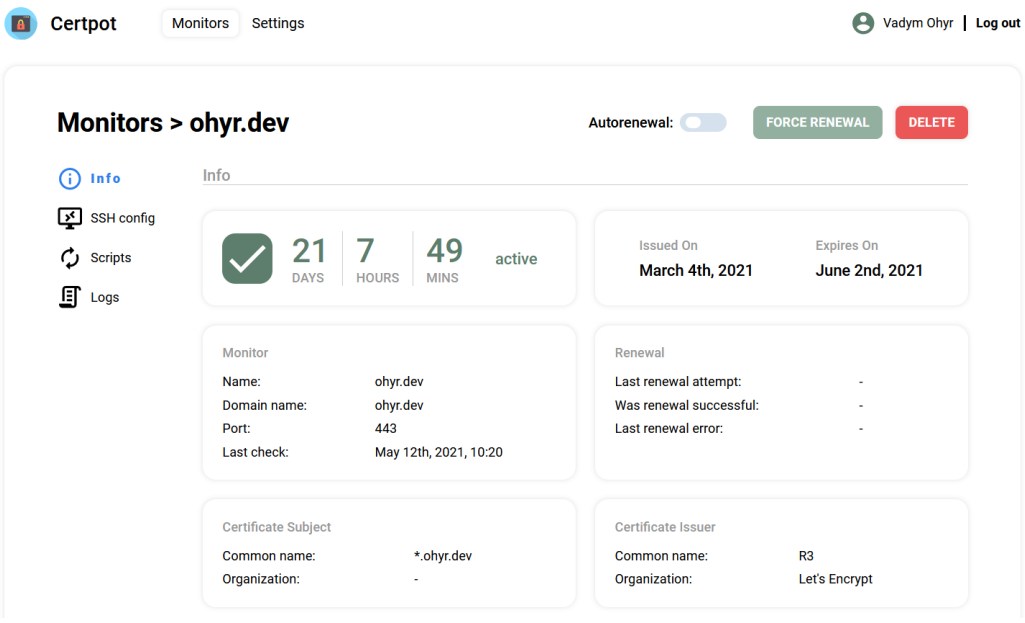

Рисунок 3.4 – Сторінка монітору

На вкладці «SSH config» можемо почати налаштування оновлення сертифікату, а саме – конфігурацію SSH для підключення до цільового серверу. Є можливість зберегти налаштування, а також протестувати з’єднання зі введеними реквізитами.

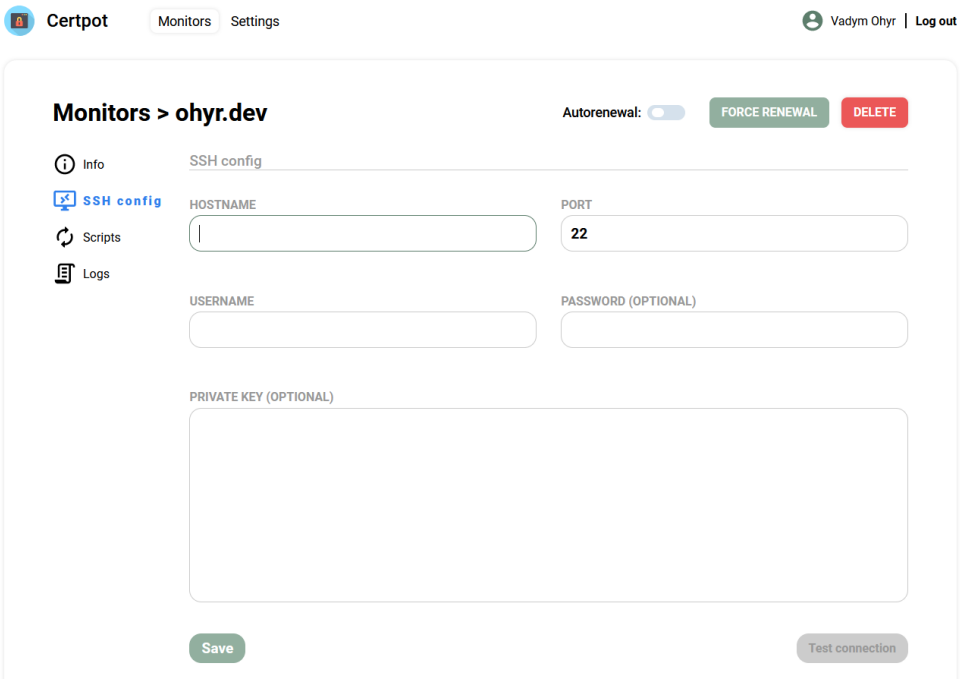

Рисунок 3.5 – Форма конфігурації SSH-доступу до цільового серверу

Для подальшого налаштування оновлення сертифікату перейдемо на вкладку «Scripts», де можемо ввести необхідні команди, що будуть виконані на цільовому сервері при оновленні сертифікату.

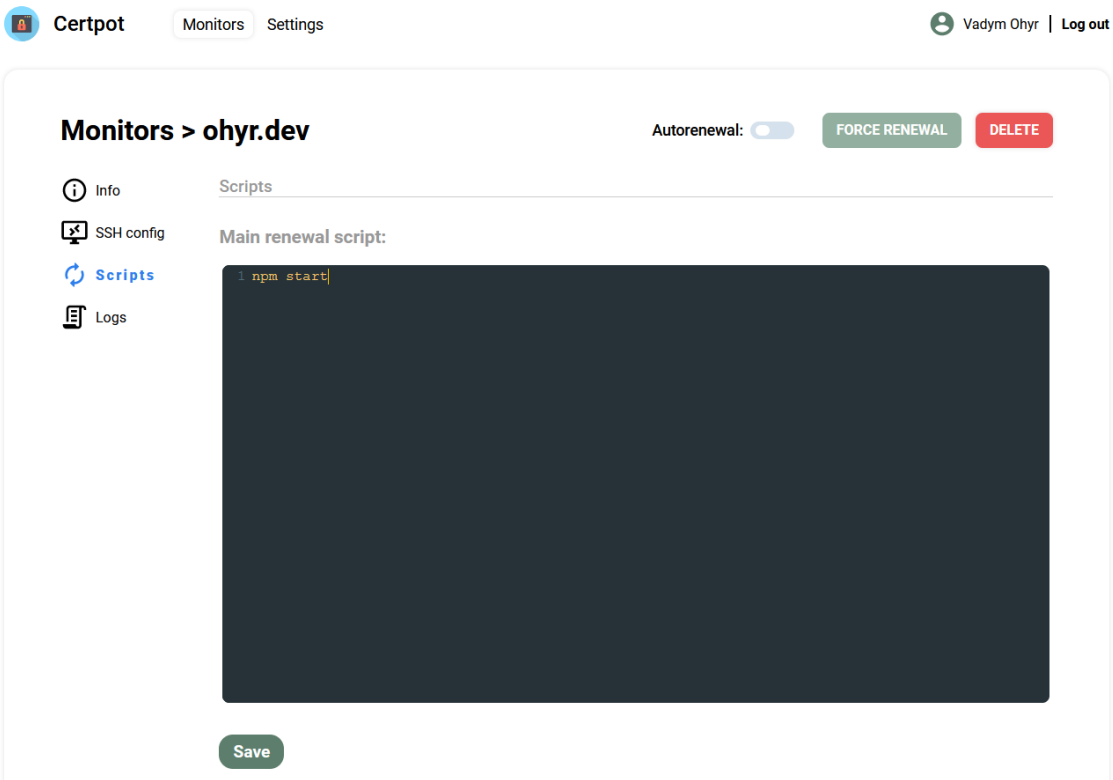

Рисунок 3.6 – Форма для команд оновлення сертифікату

Після введення і збереження всіх необхідних для оновлення даних, можемо запустити примусове оновлення, натиснувши на кнопку «Force Renewal». Оскільки провайдер мого SSL-сертифікату не надає можливості повністю автоматичного оновлення wildcard-сертифікатів, бачимо відповідну помилку.

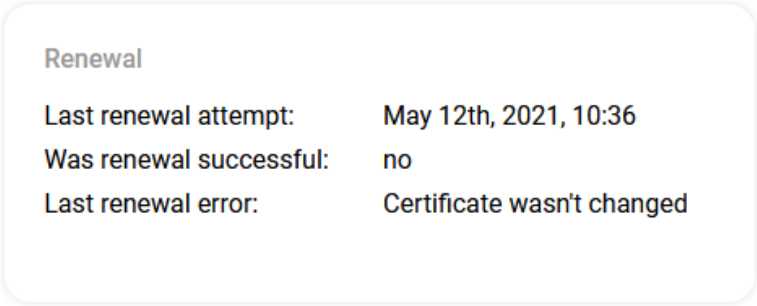

Рисунок 3.7 – Інформація про результат останньої спроби оновлення

Для детальнішої інформації про результат виконання команд оновлення можемо перейти на вкладку «Logs», де можемо бачити дату останньої спроби оновлення сертифікату, а також детальну інформацію про хід виконання команд.

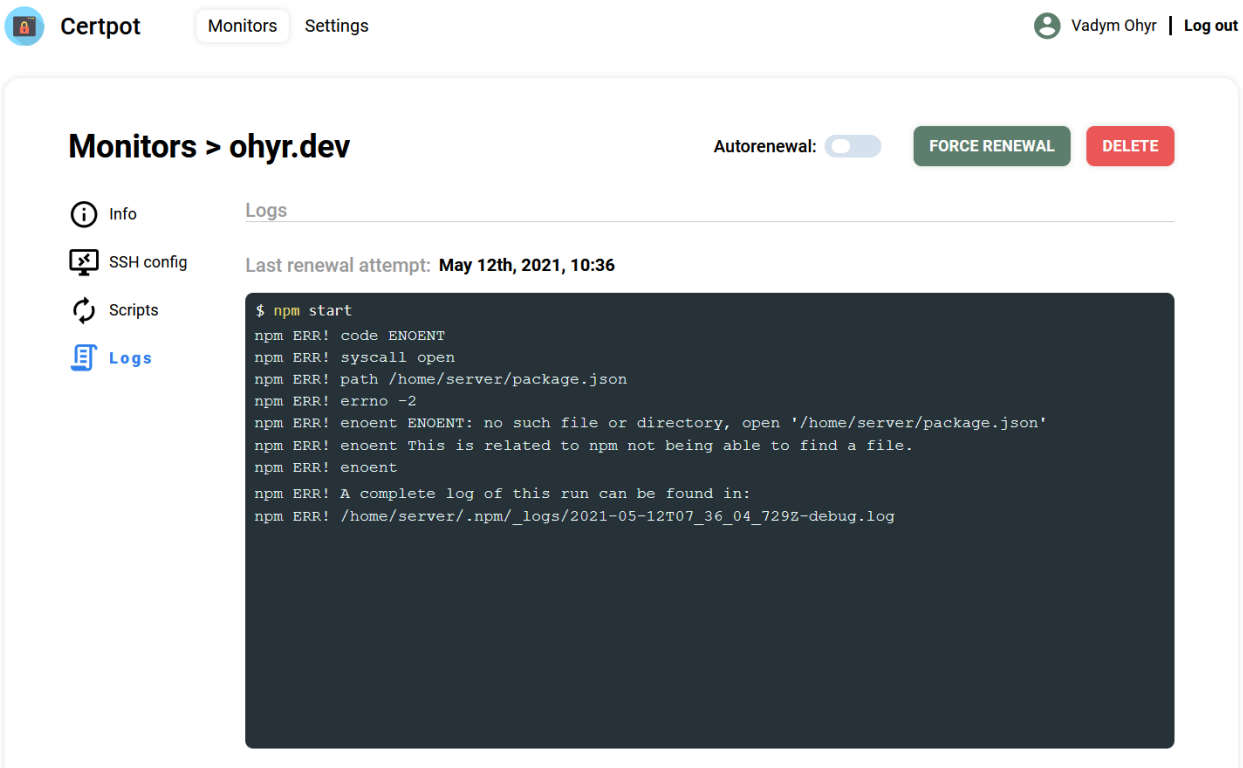

Рисунок 3.8 – Хід виконання команд останньої спроби оновлення сертифікату

Для подальшого автоматизованого оновлення сертифікату самим застосунком можемо перетягнути перемикач «Autorenewal». Таким чином, застосунок буде автоматично щоденно перевіряти сертифікат, і коли настане час, спробує його оновити, підключившись до веб-серверу користувача за SSH і виконавши вказані команди, а також сповістивши користувача про результат.

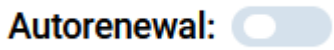

Рисунок 3.9 – Перемикач автоматичного оновлення сертифікату

ВИСНОВКИ

Під час виконання курсової роботи було оглянуто предметну область, а саме, автоматизований моніторинг і оновлення SSL-сертифікатів серверу.

Досліджено процес розробки бекенду веб-застосувань на базі фреймворку ASP.NET Core з використанням супутніх технологій, а також фронтенду на базі бібліотеки ReactJS.

Оглянуто існуючі аналоги застосування, було висвітлено переваги і недоліки кожного з них, що було враховано у формуванні завдання курсової роботи.

У результаті роботи було розроблено веб-застосування для автоматизованого моніторингу SSL-сертифікатів сервера та їх оновлення. Воно має продуману архітектуру, реалізує відомі патерни проектування програмних рішень і є зручним для використання, що забезпечується використаними бібліотеками і системою обробки помилок.

Застосування має великий потенціал до подальшого вдосконалення та розробки. До прикладу, можна додати підтримку подій в реальному часі, щоб мати можливість відслідковувати виконання скрипту оновлення SSL-сертифікату в реальному часі. Також можна додати підтримку токенів відміни (Cancellation Tokens) для навмисного переривання виконання запитів. І звісно ж, варто реалізувати шифрування даних користувачів для віддаленого підключення по SSH для виконання команд оновлення, оскільки якщо база даних буде скомпрометована, зловмисник зможе причинити ще більше шкоди серверам користувачів.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Google Transparency Report – HTTPS encryption on the web [Електронний ресурс]. Режим доступу: https://transparencyreport.google.com/https/overview?hl=en
2. Oh Dear! [Електронний ресурс]. Режим доступу: https://ohdear.app/
3. Site24x7 [Електронний ресурс]. Режим доступу: https://www.site24x7.eu/
4. Web Framework Benchmarks [Електронний ресурс]. Режим доступу: https://www.techempower.com/benchmarks/#section=data-r20&hw=ph&test=composite
5. The Clean Architecture [Електронний ресурс]. Режим доступу: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html
6. CQRS [Електронний ресурс]. Режим доступу: https://martinfowler.com/bliki/CQRS.html
7. Mediator Pattern [Електронний ресурс]. Режим доступу: https://www.oodesign.com/mediator-pattern.html
8. Redux Concepts and Data Flow [Електронний ресурс]. Режим доступу: https://redux.js.org/tutorials/fundamentals/part-2-concepts-data-flow