

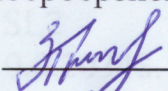
Міністерство освіти і науки України
Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки

ПОЯСНЮВАЛЬНА ЗАПИСКА

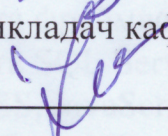
до випускної кваліфікаційної роботи бакалавра
на тему: «Проект підсистеми безпеки
багатокомпонентної вебінфраструктури, розгорнутої у
контейнеризованому середовищі Docker»

Виконав: студент 4-го курсу групи КБ-22-2

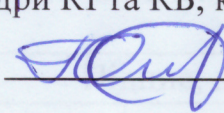
спеціальності 125 «Кібербезпека»

 Роман ЗАРУЦЬКИЙ

Керівник: старший викладач кафедри КІ та КБ

 Ігор ФАЛЬКОВСЬКИЙ

Рецензент: доцент кафедри КІ та КБ, к.пед.н., доцент

 Олена ГОЛОВНЯ

Роль	Консультант	Затвердив	Затвердження прийняв
1	Старший викладач кафедри КІ та КБ Валентин МИКОЛАЙЧУК	20.04.2026	20.04.2026

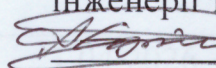
Житомир – 2026

Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки
Спеціальність 125 «Кібербезпека»
Освітня програма «Кібербезпека»

ЗАТВЕРДЖУЮ

Завідувач

кафедри комп'ютерної
інженерії та кібербезпеки

 Андрій Єфіменко

2 лютого 2026 р.

ЗАВДАННЯ

на випускн кваліфікаційну роботу бакалавра

Студент: Заруцький Роман Віталійович

Тема роботи: «Проект підсистеми безпеки багатокомпонентної вебінфраструктури, розгорнутої у контейнеризованому середовищі Docker»

затверджена Наказом університету від 2 лютого 2026 р. № 58/с

Термін здачі студентом закінченої роботи 9 червня 2026 р.

Вихідні дані роботи (зазначається предмет і об'єкт дослідження):

Об'єктом дослідження є процес побудови та розгортання багатокомпонентної вебінфраструктури з відокремленими рівнями представлення, прикладної обробки та зберігання даних.

Предметом дослідження є технології реалізації багатокомпонентної вебінфраструктури з використанням контейнеризації Docker.

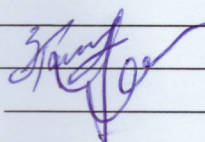
Обладнання та програмне забезпечення:

1. Гіпервізор Oracle VM VirtualBox 7.2.6
2. ОС Windows Server 2022 Standard Evaluation, Windows 10 Home
3. Docker Desktop 4.71.0 із підсистемою WSL2
4. Вебсервер Internet Information Services 10.0.20348.1
5. Середовище виконання Node.js 24.16.0 та фреймворк Express 5.2.1
6. Microsoft SQL Server 2022 Express 16.0 та SQL Server Management Studio
7. Мови та технології розробки: HTML, CSS, JavaScript

Консультанти з випускної кваліфікаційної роботи із зазначенням розділів, що їх стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Старший викладач кафедри КІ та КБ Вадим МИКОЛАЙЧУК	20.04.2026	20.04.2026

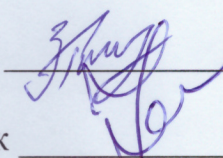
Студент
Керівник

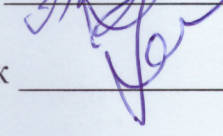


Роман ЗАРУЦЬКИЙ
Ігор ФАЛЬКОВСЬКИЙ

Календарний план

№ з/п	Назви етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Отримання та аналіз завдання на кваліфікаційну роботу	12.03.2026	Виконано
2	Опрацювання літературних і технічних джерел	25.03.2026	Виконано
3	Підготовка теоретичного розділу	1.04.2026	Виконано
4	Проектування архітектури системи	5.04.2026	Виконано
5	Вибір і налаштування програмних засобів	9.04.2026	Виконано
6	Реалізація компонентів системи	17.04.2026	Виконано
7	Перевірка процесдатності та механізмів захисту	21.04.2026	Виконано
8	Оформлення практичного розділу	6.05.2026	Виконано
9	Підготовка висновків, додатків і остаточне оформлення роботи	21.05.2026	Виконано
10	Підготовка доповіді та презентації	24.05.2026	Виконано
11	Захист кваліфікаційної роботи		Виконано

Студент  Роман ЗАРУЦЬКИЙ

Керівник  Ігор ФАЛЬКОВСЬКИЙ

ФІКТ-КР.Б-125-5-ПЗ

№ докум.	Піс.	Лист	Арх.	Арх.
Заручий Р.В.	1	1	1	1
Фальковський І.Г.	1	1	1	1
Григорук О.С.	1	1	1	1
Роздольська Л.І.	1	1	1	1
Бурдакова А.С.	1	1	1	1

ФІКТ, Гр. КБ-22-2

АНОТАЦІЯ

У межах кваліфікаційної роботи розроблено проєкт безпечної багатокомпонентної вебінфраструктури, що складається з клієнтського компонента в Docker-контейнері, вебвузла з IIS і Node.js/Express Backend API та окремого сервера Microsoft SQL Server 2022 Express.

Метою кваліфікаційної роботи є проєктування, реалізація та перевірка безпечної багатокомпонентної вебінфраструктури з відокремленими рівнями представлення, прикладної обробки та зберігання даних.

Визначено основні завдання побудови та захисту багатокомпонентної вебсистеми. Проаналізовано принципи сегментації, ізоляції, найменших привілеїв і багаторівневого захисту. Розглянуто особливості використання Docker, IIS, Node.js/Express і Microsoft SQL Server у спільній інфраструктурі. Спроєктовано склад вузлів, мережеву взаємодію, довірчі межі та правила доступу. Показано розгортання клієнтського контейнера, реверс-проксі IIS, Backend API та бази даних. Наведено результати функціонального тестування та контрольних перевірок механізмів захисту.

Пояснювальна записка містить 81 сторінку, 66 рисунків та 12 таблиць. Список використаних джерел містить 20 найменувань і розміщений на 2 сторінках.

Ключові слова: БЕЗПЕКА, ВЕБІНФРАСТРУКТУРА, КОНТЕЙНЕРИЗАЦІЯ, РЕВЕРС-ПРОКСІ, СЕГМЕНТАЦІЯ.

					ККІК<.КР.Б – 125 – 5 – ПЗ				
Змн.	Арк.	№ докум.	Підпис	Дата	Проект підсистеми безпеки багатокомпонентної вебінфраструктури, розгорнутої у контейнеризованому середовищі Docker	Літ.	Арк.	Аркушів	
Розроб.		Заруцький Р.В.		19.06.26					
Керівник		Фальковський І.Г.		19.06.26				81	
Реценз.		Головня О.С.		19.06.26					
Н. контр.		Фальковський І.Г.		19.06.26					
Затверд.		Єфіменко А.А.		19.06.26			ФІКТ, Гр. КБ-22-2		

ANNOTATION

The bachelor's qualification work presents a project of a secure multicomponent web infrastructure consisting of a client component in a Docker container, a web node with IIS and a Node.js/Express Backend API, and a separate Microsoft SQL Server 2022 Express server.

The purpose of the qualification work is to design, implement, and verify a secure multicomponent web infrastructure with separated presentation, application-processing, and data-storage layers.

The main tasks of building and protecting a multicomponent web system are defined. The principles of segmentation, isolation, least privilege, and defence in depth are analysed. The use of Docker, IIS, Node.js/Express, and Microsoft SQL Server within a single infrastructure is considered. The system nodes, network interaction, trust boundaries, and access rules are designed. The deployment of the client container, IIS reverse proxy, Backend API, and database is demonstrated. The results of functional testing and security-control verification are presented.

The explanatory note contains 81 pages, 66 figures, and 12 tables. The list of references includes 20 sources and occupies 2 pages.

Keywords: CONTAINERIZATION, WEB INFRASTRUCTURE, REVERSE PROXY, SECURITY, SEGMENTATION.

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				5
Змн.	Арк.	№ докум.	Підпис	Дата		

ЗМІСТ

ВСТУП	8
РОЗДІЛ 1 ТЕОРЕТИЧНІ ЗАСАДИ ПОБУДОВИ БЕЗПЕЧНОЇ БАГАТОКОМПОНЕНТНОЇ ВЕБІНФРАСТРУКТУРИ	9
1.1 Архітектурні принципи безпечної багатокомпонентної вебінфраструктури	9
1.2 Контейнеризація та захист вебрівня засобами Docker та IIS.....	12
1.3 Захист прикладного рівня Node.js та рівня даних SQL Server	15
1.4 Порівняльний аналіз та обґрунтування вибору програмних засобів.....	19
Висновки до розділу.....	23
РОЗДІЛ 2 ПРОЄКТУВАННЯ БЕЗПЕЧНОЇ БАГАТОКОМПОНЕНТНОЇ ВЕБІНФРАСТРУКТУРИ.....	24
2.1 Проєктування архітектури та мережевої взаємодії компонентів	24
2.2 Проєктування клієнтського та прикладного компонентів	32
2.3 Проєктування бази даних	37
2.4 Проєктування механізмів захисту	41
Висновки до розділу.....	44
РОЗДІЛ 3 РЕАЛІЗАЦІЯ ТА ПЕРЕВІРКА ПІДСИСТЕМИ БЕЗПЕКИ.....	45
3.1 Розгортання компонентів вебінфраструктури.....	45
3.2 Налаштування мережевої взаємодії та механізмів захисту	62
3.3 Перевірка стійкості до типових атак	73
Висновки до розділу.....	77
ВИСНОВКИ.....	78
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ	80
ДОДАТКИ.....	82

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface - прикладний програмний інтерфейс

ARR – Application Request Routing - модуль маршрутизації запитів IIS

CORS – Cross-Origin Resource Sharing - механізм керування міжджерельними запитами

DB – Database - база даних

HTML – HyperText Markup Language - мова розмітки гіпертексту

HTTP – HyperText Transfer Protocol - протокол передавання гіпертексту

HTTPS – HyperText Transfer Protocol Secure - захищений варіант HTTP

IIS – Internet Information Services - вебсервер Microsoft

IP – Internet Protocol - мережевий протокол та адресація вузлів

JSON – JavaScript Object Notation - текстовий формат обміну даними

SQL – Structured Query Language - мова структурованих запитів

SSMS – SQL Server Management Studio - середовище адміністрування SQL Server

TCP – Transmission Control Protocol - протокол керування передаванням

TLS – Transport Layer Security - протокол захисту транспортного рівня

URL – Uniform Resource Locator - уніфікований покажчик ресурсу

WSL 2 – Windows Subsystem for Linux 2 - підсистема Linux у Windows

XSS – Cross-Site Scripting - міжсайтове виконання сценаріїв

БД – база даних

ВМ – віртуальна машина

ОС – операційна система

ПЗ – програмне забезпечення

СУБД – система управління базами даних

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				7
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Актуальність теми. Сучасні вебзастосунки дедалі частіше реалізуються як багатокомпонентні системи, у яких окремі програмні та інфраструктурні елементи взаємодіють через мережеві інтерфейси. Безпека такої системи залежить не лише від програмного коду, а й від правильності мережевої взаємодії, розмежування повноважень і налаштування кожного компонента. Пряме відкриття внутрішніх портів, надмірні права службових облікових записів, відсутність перевірки введених даних або незахищене передавання запитів створюють ризик компрометації всієї інфраструктури.

Практичною проблемою є побудова узгодженої системи захисту, у якій клієнт не отримує прямого доступу до бази даних, зовнішні запити проходять через контрольовану точку входу, прикладний сервер виконується з обмеженими повноваженнями, а кожен мережевий зв'язок має визначене призначення. Дослідження такої архітектури є актуальними для вебсервісів, хмарних середовищ, навчальних платформ та інших програмних комплексів, у яких необхідно забезпечити контрольовану взаємодію компонентів, обмежити площу атаки та зменшити наслідки можливої компрометації.

Метою кваліфікаційної роботи є проєктування, реалізація та перевірка безпечної багатокомпонентної вебінфраструктури з відокремленими клієнтським, прикладним і інформаційним рівнями, контрольованою маршрутизацією запитів та обмеженими повноваженнями її компонентів.

Об'єктом дослідження є процес побудови та функціонування багатокомпонентної вебінфраструктури з розподілом функцій між рівнем представлення, прикладним рівнем і рівнем даних.

Предметом дослідження є методи, архітектурні рішення та програмні засоби захисту взаємодії Docker-контейнера, IIS, Node.js/Express Backend API і Microsoft SQL Server у локальному віртуалізованому середовищі.

Практичне значення результатів полягає у можливості використання стенда як навчального прикладу сегментації, найменших привілеїв і багаторівневого захисту.

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				8
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1 ТЕОРЕТИЧНІ ЗАСАДИ ПОБУДОВИ БЕЗПЕЧНОЇ БАГАТОКОМПОНЕНТНОЇ ВЕБІНФРАСТРУКТУРИ

1.1 Архітектурні принципи безпечної багатокомпонентної вебінфраструктури

Багатокомпонентна вебінфраструктура є середовищем, у якому функції системи розподілено між кількома взаємодіючими рівнями. У класичній трирівневій моделі виділяють рівень представлення, прикладний рівень і рівень даних. Рівень представлення забезпечує взаємодію з користувачем, прикладний рівень обробляє запити та реалізує правила роботи системи, а рівень даних відповідає за зберігання і зміну інформації. Вебсервер, засоби передавання запитів і прикладний програмний інтерфейс не обов’язково утворюють окремі архітектурні рівні: вони можуть бути компонентами прикладного рівня, які виконують різні функції під час обробки одного запиту.

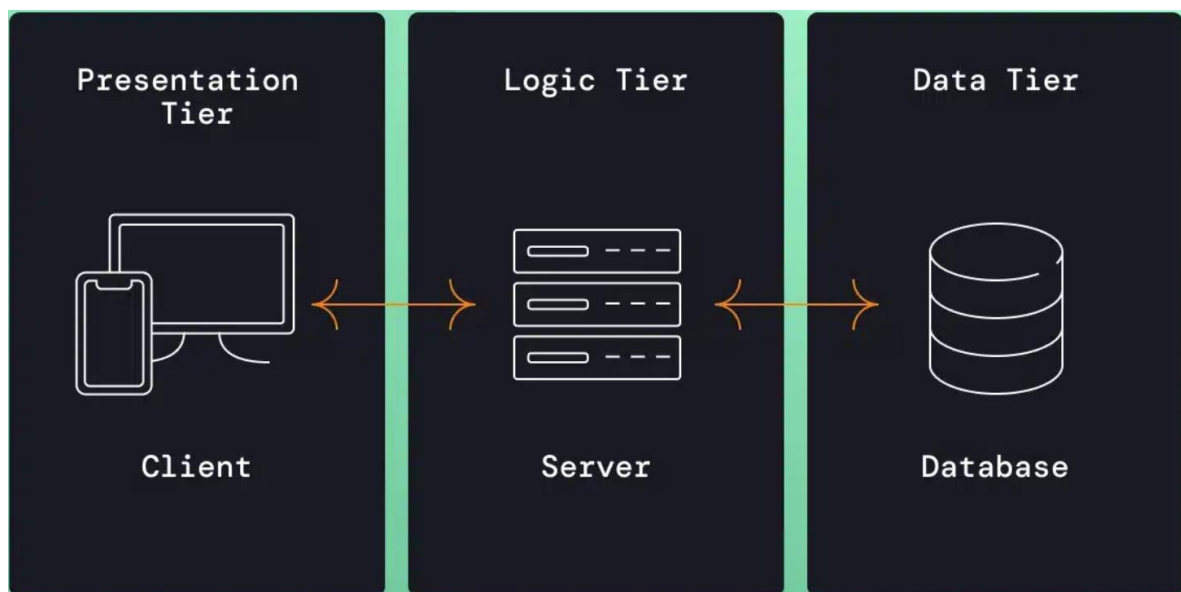


Рис. 1.1 – Трирівнева модель

На відміну від монолітного підходу, де інтерфейс, прикладна логіка та робота з даними можуть бути тісно поєднані в одному процесі або на одному вузлі, багатокомпонентна архітектура дає можливість розмежувати функції та права доступу. Таке розмежування не усуває вразливості, але зменшує ймовірність того, що порушення безпеки одного компонента автоматично призведе до повного контролю над усією системою. NIST розглядає контейнерні технології як форму

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				9
Змн.	Арк.	№ докум.	Підпис	Дата		

віртуалізації операційної системи та водночас наголошує на окремих питаннях ізоляції, конфігурації, автентифікації, аудиту й захисту комунікацій. Отже, під час проєктування необхідно враховувати не лише захист кожної програми, а й межі взаємодії між компонентами системи. Загальні функції та безпекові завдання рівнів вебінфраструктури подано в табл. 1.1. [1]

Таблиця 1.1

Функції та безпекові завдання рівнів вебінфраструктури

Рівень	Основні функції	Основні безпекові завдання
Рівень представлення	Відображення даних, формування вебзапитів	Захист з'єднання, безпечна обробка даних у браузері, недопущення прямого доступу до внутрішніх ресурсів
Прикладний рівень	Приймання запитів, перевірка даних, виконання прикладної логіки, взаємодія з базою даних	Контроль маршрутів і частоти запитів, перевірка вхідних даних, обмеження прав процесу, журналювання подій
Рівень даних	Зберігання, пошук, додавання, зміна та видалення інформації	Автентифікація підключень, розмежування дозволів, мережеве обмеження доступу, захист облікових даних

Місця, у яких дані переходять між компонентами з різними рівнями довіри, називають довірчими межами. Наприклад, дані, отримані від браузера, не можуть вважатися перевіреними лише тому, що вони надійшли через вебінтерфейс. Перед використанням на прикладному рівні їх необхідно перевірити, а передавання запиту до бази даних має відбуватися через контрольований програмний інтерфейс. Виділення довірчих меж допомагає визначити, де саме повинні застосовуватися перевірки, шифрування, журналювання та обмеження доступу.

Фундаментальними принципами захисту багатокомпонентної архітектури є сегментація, ізоляція, мінімізація площі атаки та найменші привілеї. Сегментація передбачає поділ системи на компоненти або зони відповідно до їхнього призначення. Ізоляція обмежує вплив одного компонента на інші. Площа атаки охоплює всі доступні порти, служби, програмні функції та інтерфейси, через які на систему може здійснюватися вплив; її мінімізація полягає у вимкненні або неопублікуванні непотрібних можливостей. Принцип найменших привілеїв означає,

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		10

що користувач, служба, контейнер або обліковий запис бази даних отримує лише ті дозволи, які необхідні для виконання визначених функцій. [2]

Сегментація може бути фізичною, логічною та функціональною. Фізичний поділ досягається розміщенням компонентів на різних вузлах. Логічний поділ створюється за допомогою віртуальних машин, контейнерів, мережесхем правил та окремих облікових контекстів. Функціональна сегментація полягає в тому, що кожний рівень виконує лише власні завдання: клієнтська частина не працює з базою даних безпосередньо, прикладний сервіс не отримує адміністративних повноважень, а внутрішні служби не публікуються для зовнішніх користувачів без необхідності. У результаті зменшується кількість прямих зв'язків між компонентами й спрощується локалізація наслідків інциденту.

Контейнери та віртуальні машини є різними засобами ізоляції. Віртуальна машина містить гостьову операційну систему з власним ядром, тоді як контейнери використовують ядро хостової системи й ізолюють процеси на рівні операційної системи. Завдяки цьому контейнери зазвичай швидше запускаються та потребують менше ресурсів, але їхня межа ізоляції не є тотожною межі окремої віртуальної машини. Поєднання цих підходів дає змогу використовувати віртуальні машини для відокремлення серверних середовищ, а контейнери - для стандартизованого запуску окремих застосунків усередині визначеного середовища. [3]

Важливим складником захисту є журналювання подій. Воно дає змогу фіксувати звернення до системи, помилки, невдалі спроби доступу та інші відхилення, які можуть свідчити про несправність або зловмисну активність. OWASP рекомендує враховувати вимоги до журналювання ще під час проєктування, однак конкретний обсяг залежить від призначення системи та рівня ризику. У багатокомпонентному середовищі журнали можуть формуватися на різних рівнях, але навіть журналювання на вебсервері вже надає відомості про HTTP-методи, запитувані адреси, IP-адреси клієнтів, коди відповідей і тривалість обробки запитів. [4]

Таким чином, архітектурний поділ не є самою ціллю. Його безпекове значення полягає у створенні керованих зв'язків між компонентами, обмеженні

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		11

повноважень і можливості окремо контролювати кожний етап обробки запиту. Переваги багатокомпонентної моделі реалізуються тоді, коли вебсервер, контейнерне середовище, прикладний код і база даних розглядаються як елементи єдиної системи захисту.

1.2 Контейнеризація та захист вебрівня засобами Docker та IIS

Docker використовується для пакування застосунку разом із файлами, бібліотеками та параметрами, необхідними для його запуску. Підготовлений контейнерний образ можна однаково застосовувати під час розробки, тестування та розгортання, що зменшує ймовірність помилок через відмінності у версіях залежностей або налаштуваннях середовища. Контейнер при цьому є ізольованим процесом, створеним на основі образу, а не окремою повноцінною операційною системою. [5]

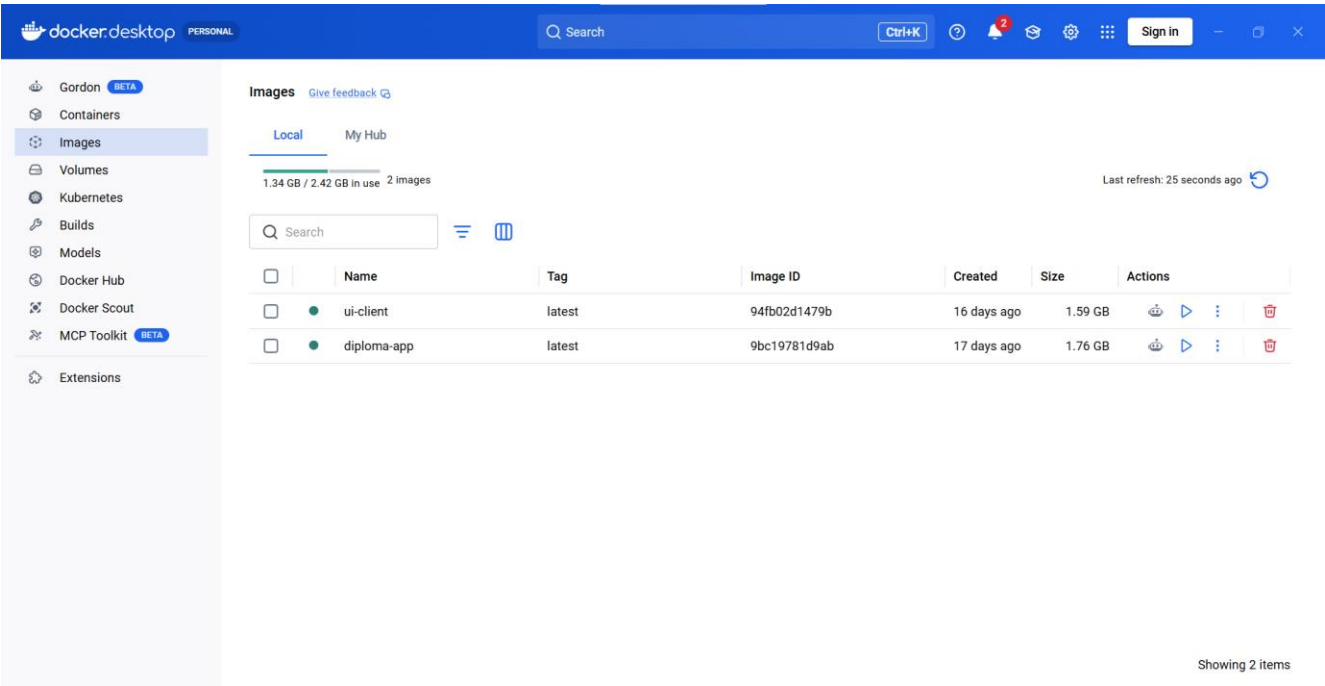


Рис. 1.2 – Docker Desktop

Контейнерний образ є незмінним шаблоном, з якого створюються контейнери. Він містить файли застосунку, системні компоненти, бібліотеки та налаштування. Зміни до вже створеного образу не вносяться безпосередньо: у разі оновлення формується нова версія. Образ складається з послідовних шарів, що відповідають етапам його створення. Незмінені шари можуть повторно

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				12
Змн.	Арк.	№ докум.	Підпис	Дата		

використовуватися під час наступного збирання, а визначений склад образу сприяє відтворюваності середовища. З погляду безпеки важливо використовувати базові образи з перевірених джерел, своєчасно оновлювати їх і не додавати компоненти, які не потрібні для роботи застосунку. [6]

Порядок створення образу задається у Dockerfile - текстовому файлі з послідовністю інструкцій. Він визначає базовий образ, робочий каталог, файли застосунку, встановлювані залежності, користувача процесу, мережевий порт і команду запуску. Таким чином, Dockerfile є не лише описом збірки, а й документованою моделлю середовища виконання. Його зміст безпосередньо впливає на площу атаки контейнера та повноваження запущеного процесу. [7]

Одним із практичних проявів принципу найменших привілеїв є запуск процесу в контейнері від непривілейованого користувача. Адміністративні права всередині контейнера не є необхідними для більшості вебзастосунків. Їх обмеження зменшує можливості процесу змінювати системні файли або встановлювати додаткові компоненти в разі компрометації. Іншим напрямом є використання мінімального базового образу, який містить лише необхідне середовище виконання та залежності. Менша кількість пакетів спрощує оновлення і скорочує кількість потенційно вразливих компонентів. [8]

Мережева взаємодія контейнера також повинна відповідати його функціям. Порт застосунку всередині контейнера не стає автоматично доступним із зовнішньої мережі: для цього виконується його публікація на хостовій системі. Такий механізм дає змогу визначити, через яку адресу й порт клієнти або інші компоненти можуть звертатися до контейнера. З безпекового погляду публікуються лише необхідні порти, а внутрішні служби залишаються недоступними для безпосередніх зовнішніх звернень. [9]

Контейнери поділяються на Linux- і Windows-контейнери залежно від ядра та системних механізмів, з якими вони працюють. Linux-контейнери використовують ядро Linux, а Windows-контейнери - сумісне ядро Windows. Вибір типу контейнера визначається залежностями застосунку й операційним середовищем. У гібридній інфраструктурі контейнери можуть поєднуватися з

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		13

віртуальними машинами та серверними службами, якщо різні компоненти потребують різних операційних систем.

У Windows для запуску Linux-контейнерів Docker Desktop може використовувати Windows Subsystem for Linux 2 (WSL 2). WSL 2 створює кероване віртуалізоване Linux-середовище з власним ядром, завдяки чому програми й контейнери для Linux можуть виконуватися на Windows-хості. Користувачеві не потрібно окремо створювати й обслуговувати повну Linux-віртуальну машину, однак безпека контейнерів у такій моделі залежить також від стану Windows, WSL 2 і Docker Desktop. Тому актуальність оновлень та обмеження доступу до файлової системи й мережі залишаються важливими складниками захисту. [10]

На вебрівні функцію приймання HTTP- і HTTPS-запитів може виконувати IIS. Його модульна архітектура означає, що окремі компоненти відповідають за обробку запитів, перенаправлення, журналювання, автентифікацію та інші функції. Під час надходження запиту IIS застосовує налаштовані правила й передає його відповідному обробнику або прикладному сервісу. Це дає змогу зосередити частину захисних функцій на одному контрольованому вузлі. [11]

У багатокомпонентній системі IIS може виконувати функцію зворотного проксі-сервера. Зворотний проксі-сервер приймає запит від клієнта, а потім передає його внутрішньому прикладному сервісу та повертає клієнтові отриману відповідь. У IIS така взаємодія може бути реалізована поєднанням URL Rewrite і Application Request Routing. URL Rewrite визначає правила зміни або перенаправлення адрес, а Application Request Routing забезпечує передавання запиту до внутрішнього сервісу. У результаті клієнт взаємодіє з IIS, тоді як внутрішня адреса прикладного процесу не обов'язково публікується назовні. [12]

Правила URL Rewrite дають змогу відокремити адреси, доступні користувачам, від внутрішніх адрес застосунку. Зовнішній маршрут може бути перетворений на адресу внутрішнього прикладного сервісу без розкриття його порту або структури розміщення. Модуль також використовується для автоматичного перенаправлення HTTP-запитів на HTTPS. Саме переписування адрес не замінює авторизацію чи мережеві обмеження, але забезпечує

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		14

централізоване керування маршрутами та зменшує необхідність дублювати такі правила у прикладному коді. [13]

HTTPS є захищеним варіантом передавання HTTP-запитів і використовує протокол TLS для шифрування з'єднання. TLS забезпечує конфіденційність переданих даних, захист від їх непомітної зміни та підтвердження справжності сервера за допомогою цифрового сертифіката. OWASP рекомендує застосовувати HTTPS до всього вебзастосунку, оскільки незахищені сторінки також можуть стати джерелом перехоплення або підміни інформації. У загальнодоступних системах використовуються сертифікати, видані довіреними центрами сертифікації. Самопідписаний сертифікат забезпечує шифрування, але не має автоматичної довіри браузерів, тому є придатним переважно для навчальних, тестових і внутрішніх середовищ. [14]

У ІІС відповідність сайту протоколу, IP-адресі, порту та сертифікату визначається прив'язками сайту. Завдяки цьому саме вебсервер може приймати захищене HTTPS-з'єднання, а потім передавати запит внутрішньому прикладному процесу. Централізоване налаштування HTTPS спрощує керування сертифікатом і перенаправлення з HTTP на HTTPS, оскільки внутрішньому сервісу не потрібно окремо публікувати власну зовнішню HTTPS-адресу. [15]

Журналювання ІІС доповнює мережевий захист можливістю аналізувати фактичні звернення. У журналах можуть фіксуватися дата й час, адреса клієнта, HTTP-метод, запитуваний ресурс, код відповіді та тривалість обробки. Ці відомості дають змогу виявляти повторювані помилки, звернення до неіснуючих маршрутів, незвичну частоту запитів і спроби автоматизованого сканування. Журнал вебсервера не відображає всіх внутрішніх подій застосунку, але є важливим джерелом інформації про зовнішню взаємодію із системою. [16]

1.3 Захист прикладного рівня Node.js та рівня даних SQL Server

Середовище виконання є сукупністю програмних засобів, які забезпечують запуск і роботу програмного коду. JavaScript спочатку виконувався переважно у веббраузерах, однак Node.js надає можливість використовувати його поза

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		15

браузером, зокрема для серверних застосунків. Node.js містить засоби для роботи з мережею, файлами, операційною системою та іншими ресурсами, необхідними для приймання запитів і формування відповідей.



Рис. 1.3 – Node.js та фреймворк Express

Важливою особливістю Node.js є цикл подій. Під час очікування завершення мережевого звернення, читання файлу або відповіді бази даних середовище може продовжувати приймати й обробляти інші події. Така модель не означає автоматичної паралельності всіх операцій, але дає змогу ефективно працювати з великою кількістю завдань, значна частина яких пов'язана з очікуванням введення та виведення. Тому Node.js добре підходить для серверних програмних інтерфейсів, які приймають HTTP-запити, виконують прикладну логіку та звертаються до зовнішніх сервісів або баз даних. [17]

Екосистема Node.js використовує npm для керування пакетами й залежностями. Файл package.json описує бібліотеки, необхідні застосунку, а файл фіксації залежностей дає змогу відтворювати узгоджені версії пакетів. Кожна зовнішня бібліотека збільшує обсяг коду, якому довіряє застосунок, тому безпека залежить від обґрунтованого вибору пакетів, контролю їх версій, своєчасного оновлення та видалення невикористовуваних залежностей. Це зменшує кількість компонентів, у яких можуть міститися відомі вразливості. [18]

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				16
Змн.	Арк.	№ докум.	Підпис	Дата		

Express є фреймворком для Node.js, який надає засоби визначення маршрутів, приймання HTTP-запитів і формування відповідей. Спільні операції в Express можуть реалізовуватися проміжними функціями. Така функція отримує запит до того, як буде виконаний кінцевий маршрут, і може перевірити дані, змінити відповідь, зафіксувати подію або відхилити звернення. Послідовне виконання проміжних функцій дає змогу централізувати спільні перевірки, а не повторювати їх у кожному маршруті.

Прикладний програмний інтерфейс (API) визначає правила взаємодії між клієнтською частиною та сервером. У багатокомпонентній архітектурі API є контрольованою точкою доступу до прикладної логіки й даних. Клієнт надсилає запит до визначеного маршруту, а сервер перевіряє його, виконує дозволену операцію та повертає структуровану відповідь. Прямий доступ клієнтської частини до бази даних є небезпечним, оскільки ускладнює розмежування прав і розкриває внутрішню структуру даних. Тому саме API виконує роль посередника, який відокремлює зовнішні запити від операцій рівня даних. [19]

Для підвищення безпеки Express-застосунку застосовуються спеціалізовані проміжні функції. Пакет Helmet установлює захисні HTTP-заголовки, які визначають окремі правила обробки вмісту браузером. Вони можуть обмежувати вбудовування сторінки в сторонні ресурси, забороняти браузеру самостійно змінювати визначений тип вмісту та посилювати використання захищеного з'єднання. Такі заголовки зменшують ризик окремих атак на клієнтську частину, але не замінюють перевірку вхідних даних, автентифікацію або захист каналу зв'язку.

Обмеження частоти запитів контролює кількість звернень, які один клієнт може виконати протягом визначеного проміжку часу. У середовищі Express цей механізм може реалізовуватися бібліотекою express-rate-limit. Після перевищення ліміту наступні запити тимчасово відхиляються. Це ускладнює масове сканування адрес, автоматизований підбір облікових даних і створення надмірного навантаження великою кількістю однотипних звернень. Обмеження частоти не

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		17

визначає наміри користувача, тому найбільшу цінність має разом із журналюванням і аналізом повторюваних подій.

CORS керує браузерними запитами між ресурсами, що мають різні протоколи, адреси або порти. За замовчуванням браузер обмежує таку взаємодію, а налаштування CORS визначає, з яких адрес сервер дозволяє приймати запити. Цей механізм має значення, коли клієнтська частина й API доступні з різних джерел. Дозвіл запитів із будь-якої адреси є виправданим лише для ресурсів, які справді мають бути загальнодоступними. Водночас CORS не встановлює особу користувача і не визначає його повноваження, тому не є заміною автентифікації та авторизації.

Для взаємодії Node.js із SQL Server використовується бібліотека mssql. Параметри підключення охоплюють адресу сервера, назву бази даних, облікові дані та налаштування з'єднання. Бібліотека підтримує пул з'єднань - набір уже встановлених підключень, які повторно використовуються для різних операцій. Це зменшує витрати часу й ресурсів порівняно зі створенням нового мережевого з'єднання для кожного HTTP-запиту.

Критичним засобом захисту під час формування SQL-запитів є параметризація. У параметризованому запиті текст команди та значення, отримані від користувача, передаються окремо. Завдяки цьому введені дані обробляються як значення, а не як частина SQL-команди. Такий підхід протидіє SQL-ін'єкції, за якої спеціально сформоване введення змінює логіку запиту до бази даних. Параметризація повинна доповнюватися перевіркою типу, довжини та допустимого змісту даних.

Microsoft SQL Server 2022 є системою керування базами даних, у якій інформація організована у вигляді пов'язаних таблиць. SQL Server забезпечує зберігання, пошук, додавання, зміну та видалення даних, а також містить механізми автентифікації, авторизації, мережевого доступу та журналювання. Автентифікація визначає, який користувач або сервіс установлює з'єднання, а авторизація - які дії він має право виконувати після підключення. [20]

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		18

База даних не повинна бути безпосередньо доступною клієнтській частині вебзастосунку. Звернення до неї проходять через прикладний сервіс, який контролює маршрути й допустимі операції. На мережевому рівні SQL Server приймає підключення через визначений TCP-порт, однак наявність відкритого порту не означає необхідності робити його доступним для всіх вузлів. Правила брандмауера можуть обмежувати підключення адресами або мережевими сегментами, з яких працює прикладний сервер. Таке обмеження зменшує кількість джерел, здатних безпосередньо звертатися до СУБД.

Для підключення прикладного сервісу використовується окремий службовий обліковий запис. Його права обмежуються конкретною базою даних та операціями, необхідними застосунку. Адміністративні облікові записи на кшталт системного адміністратора не призначені для повсякденного доступу вебзастосунку. Розмежування адміністративного й прикладного доступу спрощує відкликання дозволів, заміну облікових даних, аналіз журналів і локалізацію наслідків компрометації. Ролі читання та запису можуть використовуватися як базовий спосіб надання доступу, однак точніші дозволи на окремі таблиці або операції краще відповідають принципу найменших привілеїв.

Паролі, рядки підключення та інші конфіденційні параметри не повинні зберігатися безпосередньо у вихідному коді. Їх винесення до змінних середовища відокремлює конфігурацію від програмної логіки й зменшує ризик випадкового потрапляння секретів до репозиторію. Водночас змінні середовища не є самостійним сховищем секретів: доступ до файлів конфігурації та середовища процесу також повинен бути обмежений. Під час журналювання необхідно уникати запису паролів, токенів і повних рядків підключення. У загальнодоступному середовищі також перевіряється сертифікат SQL Server, тоді як спрощені налаштування довіри допустимі лише в контрольованих тестових умовах.

1.4 Порівняльний аналіз та обґрунтування вибору програмних засобів

Вибір програмних засобів для побудови багатокомпонентної вебінфраструктури повинен визначатися не лише їхньою поширеністю, а й

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		19

відповідністю архітектури системи, операційному середовищу, вимогам безпеки та характеру прикладного навантаження. Одна технологія не може вважатися універсально кращою за іншу, оскільки кожне рішення має власні переваги, обмеження та сферу доцільного застосування. Тому вибір програмних засобів потребує порівняння з функціонально близькими альтернативами.

Функції приймання HTTP- та HTTPS-запитів, керування сертифікатами, журналювання й передавання запитів прикладному сервісу можуть виконувати IIS, Nginx або Apache HTTP Server. Усі три рішення здатні працювати як вебсервери та передавати запити внутрішнім застосункам, але відрізняються способом адміністрування та інтеграцією з операційною системою.

IIS є компонентом Windows Server і підтримує керування сайтами, журналами, сертифікатами, правами доступу та мережевими прив'язками через графічні й командні засоби Windows. За допомогою URL Rewrite та Application Request Routing він може передавати запити до внутрішнього прикладного сервісу. Nginx широко використовується як вебсервер, сервер статичних файлів, засіб передавання запитів і балансування навантаження. Apache HTTP Server також має модульну будову й підтримує передавання запитів через модуль mod_proxy. Порівняння цих рішень подано в табл. 1.2.

Таблиця 1.2

Порівняння вебсерверів

Критерій	IIS	Nginx	Apache HTTP Server
Основне середовище використання	Windows, Windows Server	Linux, Unix, Windows	Linux, Unix, Windows
Спосіб налаштування	IIS Manager, PowerShell, конфігураційні файли	Конфігураційні файли	Конфігураційні файли та модулі
Передавання запитів внутрішньому сервісу	URL Rewrite, Application Request Routing	Модуль HTTP Proxy	Модуль mod_proxy
Робота із сертифікатами у Windows	Інтеграція зі сховищем сертифікатів Windows	Сертифікати задаються у конфігурації	Сертифікати задаються у конфігурації
Журналювання	Вбудоване журналювання IIS із налаштуванням полів	Журнали доступу та помилок	Журнали доступу та помилок

Продовження таблиці 1.2

Перевага для розробленої системи	Відповідність Windows Server та використаним засобам адміністрування	Доцільніший для повністю Linux-орієнтованої інфраструктури	Гнучка модульність, але без переваг інтеграції з Windows Server
----------------------------------	--	--	---

Для реалізації прикладного програмного інтерфейсу доцільно порівняти Node.js із ASP.NET Core. У цьому контексті коректніше використовувати назву ASP.NET Core, оскільки саме ця сучасна платформа застосовується для створення серверних вебзастосунків і API. Класичний ASP.NET Framework значно тісніше пов'язаний із Windows та IIS, тоді як ASP.NET Core є багатоплатформним і використовує власний вебсервер Kestrel. Основні відмінності між Node.js та ASP.NET Core наведено в табл. 1.3.

Таблиця 1.3

Порівняння Node.js та ASP.NET

Критерій	Node.js	ASP.NET
Інтеграція з IIS	Через URL Rewrite та Application Request Routing	Через ASP.NET Core Module
Модель обробки операцій	Цикл подій і неблокуючі операції введення-виведення	Асинхронні операції та керування потоками середовищем .NET
Структура фреймворку	Мінімальне ядро, функції додаються проміжними компонентами й пакетами	Значна кількість засобів входить до платформи
Вбудовані засоби	Базові HTTP-функції та маршрутизація Express	Конфігурація, журналювання, впровадження залежностей, автентифікація та авторизація
Керування залежностями	npm і package.json	NuGet і файли проєкту .NET
Типізація	Динамічна в JavaScript	Статична в C#
Відповідність невеликому API	Дає змогу створити компактний прикладний сервіс	Також підтримує компактні API, але має ширшу платформну структуру
Відповідність Windows-інфраструктурі	Працює незалежно від операційної системи, але потребує окремої інтеграції з IIS	Має безпосередню підтримку IIS та засобів Microsoft

Для розробленої системи Node.js було обрано з кількох причин. По-перше, клієнтська та серверна частини можуть використовувати одну мову програмування - JavaScript. Це зменшує кількість різних мов і програмних середовищ, які необхідно підтримувати в межах невеликого проєкту.

По-друге, окреме виконання Node.js і IIS підкреслює багатокомпонентний характер архітектури. IIS залишається зовнішнім вебсервером, а Node.js - незалежним прикладним сервісом. Це дає змогу окремо дослідити захист HTTPS-з'єднання, правила передавання запитів, журнали IIS, мережеву доступність API та права прикладного процесу. У разі використання ASP.NET Core з виконанням усередині процесу IIS межа між вебсервером і застосунком була б менш вираженою.

По-третє, Node.js є багатоплатформним середовищем. Прикладний сервіс може бути перенесений із Windows Server до Linux або контейнерного середовища без зміни основної мови та архітектури API. Це знижує залежність прикладного рівня від конкретної серверної операційної системи.

Для зберігання структурованих даних могли використовуватися Microsoft SQL Server, PostgreSQL або MySQL. Усі три системи підтримують таблиці, транзакції, ролі, дозволи та мережеве підключення прикладних сервісів. Різниця полягає передусім в інтеграції з операційною системою, засобах адміністрування, особливостях керування доступом та сумісності з обраною інфраструктурою.

Таблиця 1.4

Порівняння систем керування базами даних

Критерій	Microsoft SQL Server	PostgreSQL	MySQL
Тип системи	Реляційна СУБД Microsoft	Відкрита об'єктно-реляційна СУБД	Відкрита реляційна СУБД
Підтримка ролей і дозволів	Серверні ролі, ролі баз даних та об'єктні дозволи	Єдина система ролей і дозволів на об'єкти	Облікові записи, ролі та привілеї
Інтеграція з Windows	Висока, включно з Windows Authentication і засобами Microsoft SQL Server	Підтримується, але без настільки тісної інтеграції	Підтримується, але без настільки тісної інтеграції
Основні засоби адміністрування	SQL Server Management Studio	pgAdmin, psql та інші засоби	MySQL Workbench і консольні засоби
Відповідність розроблених інфраструктурі	Висока завдяки використанню Windows Server і засобів Microsoft	Доцільна для багатоплатформного відкритого стеку	Доцільна для традиційних вебсистем із простішими вимогами

Вибір SQL Server дає змогу дослідити мережеве відокремлення рівня даних, налаштування TCP/IP, обмеження доступу до порту бази даних і використання окремих адміністративних та прикладних облікових записів. PostgreSQL або MySQL могли б виконувати основні функції зберігання даних не менш коректно, однак їх застосування не мало б суттєвої переваги в інфраструктурі, побудованій на Windows Server та IIS.

Висновки до розділу

У першому розділі було розглянуто теоретичні засади побудови безпечної багатокомпонентної вебінфраструктури. Проведений аналіз показав, що використання багаторівневої архітектури дає змогу розмежувати функції представлення, прикладної логіки та зберігання даних, що сприяє підвищенню рівня захисту системи та спрощує контроль взаємодії між її компонентами.

Досліджено особливості контейнеризації із застосуванням Docker, використання вебсервера IIS як точки входу до системи та засобу реалізації HTTPS і reverse проху, а також принципи побудови прикладного рівня на основі Node.js і рівня даних на базі Microsoft SQL Server. Окрему увагу приділено таким механізмам захисту, як сегментація сервісів, ізоляція компонентів, принцип найменших привілеїв, використання захищених каналів зв'язку, журналювання подій та захист від типових загроз.

Також було виконано порівняльний аналіз альтернативних технологій та обґрунтовано вибір IIS, Node.js і Microsoft SQL Server для реалізації проєкту. Проведене дослідження підтвердило доцільність використання обраного технологічного стеку для побудови захищеної багатокомпонентної вебінфраструктури, що забезпечує належний рівень функціональності, керованості та безпеки.

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		23

РОЗДІЛ 2 ПРОЄКТУВАННЯ БЕЗПЕЧНОЇ БАГАТОКОМПОНЕНТНОЇ ВЕБІНФРАСТРУКТУРИ

2.1 Проєктування архітектури та мережевої взаємодії компонентів

Проєктована система будується як багатокомпонентна вебінфраструктура, у якій клієнтська частина, прикладна обробка та зберігання даних розміщені окремо. Такий поділ запроваджено не для формального збільшення кількості компонентів, а для чіткого розмежування їхніх завдань. Клієнтська частина відповідає за взаємодію з користувачем, вузол Serv-WEB-IIS приймає зовнішні запити та виконує прикладну обробку, а Serv-DB-SQL зберігає дані. Завдяки цьому зміни в одному компоненті не потребують повного перебудування всієї системи.

Фізично стенд складається з основного комп'ютера та двох віртуальних машин. На основному комп'ютері з Windows 10 Home працюють VirtualBox, Docker Desktop, WSL 2 і браузер. У Docker-контейнері розміщується клієнтська частина вебзастосунку. Віртуальна машина Serv-WEB-IIS працює під керуванням Windows Server 2022 і містить IIS, модулі URL Rewrite та Application Request Routing, а також окремий процес Node.js/Express із прикладним API. Друга віртуальна машина Serv-DB-SQL також використовує Windows Server 2022 і містить SQL Server 2022 Express та засоби адміністрування бази даних.

Логічно система відповідає трирівневій моделі. Рівень представлення утворює клієнтський вебінтерфейс, який завантажується в браузер. Прикладний рівень складається з IIS і Node.js/Express API. IIS є вебсервером за призначенням платформи, але в цій конфігурації його головна роль полягає у прийманні зовнішніх запитів і передаванні звернень до внутрішнього API. Node.js/Express запускає власний HTTP-сервер на локальній адресі та виконує маршрути, перевірки й роботу з базою даних. Рівень даних представлений SQL Server.

Розміщення IIS і Node.js на одному серверному вузлі не скасовує поділу їхніх функцій. IIS працює як зовнішня точка входу, тоді як Node.js доступний лише локально на Serv-WEB-IIS. Це означає, що клієнт не повинен звертатися до порту Node.js безпосередньо.

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		24

Контейнеризовано лише клієнтську частину. Усередині контейнера знаходяться файли HTML, CSS і JavaScript, а невеликий процес Node.js/Express передає їх браузеру через HTTP. Після завантаження сторінки клієнтський JavaScript виконується вже у браузері основного комп'ютера. Саме браузер надсилає запити до IIS. Тому контейнер не виконує прикладної логіки та не підключається до SQL Server, але забезпечує однаковий спосіб запуску вебінтерфейсу на основному вузлі.

Обрана структура дає змогу розглядати систему як послідовність контрольованих переходів: від користувача до клієнтської сторінки, від браузера до IIS, від IIS до локального Node.js API та від API до SQL Server. Кожен перехід має визначену адресу, порт і призначення. Це спрощує подальше налаштування, оскільки для кожного з'єднання можна окремо перевірити доступність і встановити очікуваний результат. Опис вузлів подано в табл. 2.1.

Таблиця 2.1

Склад вузлів проєктованої інфраструктури

Вузол	Адреса і середовище	Програмні компоненти	Призначення
Основний вузол	Windows 10 Home; 192.168.56.1	VirtualBox, Docker Desktop, WSL 2, браузер	Запуск клієнтського контейнера, відкриття вебінтерфейсу та надсилання запитів
Serv-WEB-IIS	Windows Server 2022; 192.168.56.102	IIS, URL Rewrite, ARR, Node.js, Express	Приймання зовнішніх запитів, передавання їх до API та виконання прикладної логіки.
Serv-DB-SQL	Windows Server 2022; 192.168.56.101	SQL Server 2022 Express, SSMS	Зберігання QualificationDB і перевірка прав облікового запису застосунку.

Схему розміщення компонентів наведено на рис. 2.2. Вона показує не лише місце встановлення програм, а й напрям штатної взаємодії. Користувач відкриває клієнтську частину з контейнера, після чого браузер звертається до IIS. IIS передає запит локальному API, а Node.js встановлює з'єднання з SQL Server. Червоними

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		25

пунктирними лініями на схемі позначено небажані прямі звернення, які не використовуються у нормальній роботі системи.

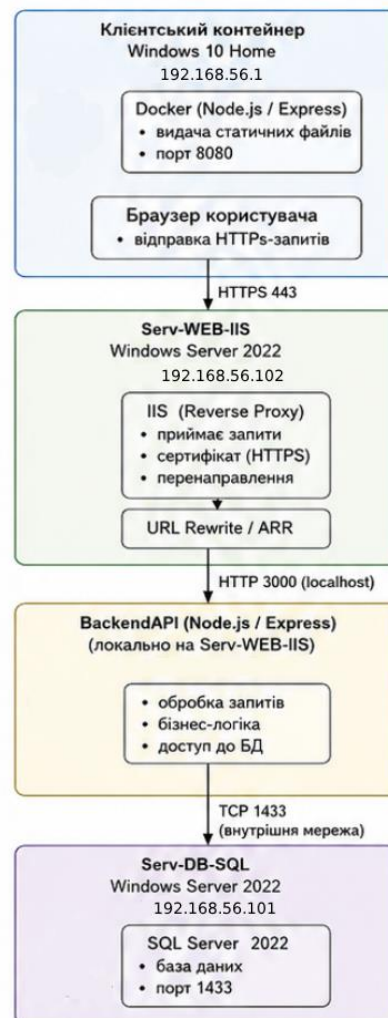


Рис. 2.1 – Схема розгортання та мережевої взаємодії компонентів

Для взаємодії основного комп'ютера і двох віртуальних машин використовується мережа VirtualBox Host-Only 192.168.56.0/24. Основний вузол має адресу 192.168.56.1, Serv-DB-SQL – 192.168.56.101, а Serv-WEB-IIS – 192.168.56.102.

Режим Host-Only створює окремий сегмент між хостом і віртуальними машинами. Сервери доступні з основного комп'ютера, але не підключаються до фізичної локальної мережі як звичайні пристрої.

Використання Bridged Adapter для цього стенда було б менш зручним. У такому режимі віртуальні машини отримували б адреси з фізичної мережі та могли б бути видимими для інших пристроїв. Для виробничого сервера це може бути нормальним рішенням, але для локального прототипу така доступність не потрібна.

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		26

NAT, навпаки, добре підходить для вихідного доступу, проте вимагає додаткового перенаправлення портів для звернень з основного вузла. Host-Only забезпечує прямий і зрозумілий зв'язок без цих додаткових налаштувань.

Статичні адреси потрібні через те, що вони використовуються в декількох місцях. Адреса SQL Server зазначається в параметрах Backend API, адреса IIS використовується клієнтським кодом, а правила брандмауера можуть враховувати джерело з'єднання. Якщо адреса віртуальної машини зміниться, частина запитів перестане проходити.

Мережева схема не передбачає маршрутизації безпосередньо від клієнтської частини до SQL Server. Навіть якщо основний комп'ютер технічно знаходиться в тому самому Host-Only сегменті, клієнтський JavaScript не містить параметрів підключення до SQL Server і не використовує протокол бази даних. Усі операції з даними виконуються через Backend API. Це зберігає єдиний порядок обробки запитів і не переносить службові облікові дані до браузера.

Для кожного компонента визначено окремий мережевий інтерфейс або порт. Клієнтська частина публікує порт вебінтерфейсу на основному комп'ютері. IIS приймає HTTP-запити на порту 80 та HTTPS-запити на порту 443. Внутрішній Node.js API слухає порт 3000 лише на адресі 127.0.0.1. SQL Server приймає TCP-з'єднання на порту 1433. Такий розподіл дозволяє одразу визначити, яка програма повинна відповідати на конкретне звернення.

Порт 3000 має особливе значення. У Backend API використано прив'язку до 127.0.0.1, тому процес Node.js приймає з'єднання лише з того самого сервера. Відкриття порту 3000 у брандмауері не зробить API доступним з основного комп'ютера, доки застосунок не почне слухати адресу 192.168.56.102 або всі мережеві інтерфейси. Отже, локальна прив'язка є основним обмеженням, а IIS є єдиним передбаченим шляхом до API.

Матриця в табл. 2.2 описує саме очікуваний робочий потік. Її призначення полягає в тому, щоб показати, які звернення використовуються системою і які з них потрібно перевіряти під час тестування. Наприклад, основний вузол повинен мати доступ до IIS, але не повинен отримувати відповідь від Node.js на порту 3000.

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		27

Матриця мережевої взаємодії компонентів

Джерело	Призначення	Протокол і порт	Очікуваний стан	Призначення з'єднання
Браузер основного вузла	Контейнеризована клієнтська частина	HTTP; опублікований порт контейнера	Дозволено	Завантаження HTML, CSS і JavaScript.
Браузер основного вузла	IIS на 192.168.56.102	HTTP 80 / HTTPS 443	Дозволено	Звернення до API через зовнішню адресу.
Основний вузол	Node.js API на 192.168.56.102:3000	TCP 3000	Недоступно зовні	Недопущення обходу IIS.
IIS	Node.js API	HTTP 127.0.0.1:3000	Дозволено локально	Передавання запитів до внутрішнього API.
Node.js API	SQL Server 192.168.56.101	TCP 1433	Дозволено	Читання та додавання даних через webapp_user.
Основний вузол	SQL Server 192.168.56.101	TCP 1433	Залежить від області Local Subnet	Адміністративна перевірка; не використовується клієнтським застосунком.

Для перегляду повідомлень клієнтський код формує GET-запит до шляху /api/messages за адресою IIS. IIS приймає зовнішнє з'єднання, перевіряє правило маршрутизації та передає запит на локальну адресу Node.js. Метод GET і шлях запиту зберігаються, тому Backend API отримує звернення до власного маршруту /api/messages. Після обробки Node.js формує запит SELECT до SQL Server і отримує набір записів.

SQL Server повертає результат прикладному процесу. Backend API перетворює отримані записи на JSON-відповідь, після чого відповідь повертається до IIS і далі до браузера. IIS не виконує логіку вибірки повідомлень і не підключається до таблиці безпосередньо. Його завдання обмежується прийманням зовнішнього HTTP-запиту та передаванням його внутрішньому серверу.

Додавання повідомлення відбувається за тією самою загальною схемою, але використовується метод POST. Браузер формує JSON з полями Username та MessageText. IIS передає метод, шлях і вміст запиту до Node.js. Backend API перевіряє наявність і допустиму довжину значень, після чого виконує

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		28

параметризований INSERT. SQL Server додає запис, а клієнт отримує підтвердження або повідомлення про помилку.

Послідовність обробки наведено на рис. 2.2, де окремо показано дії браузера, IIS, Node.js API та SQL Server.

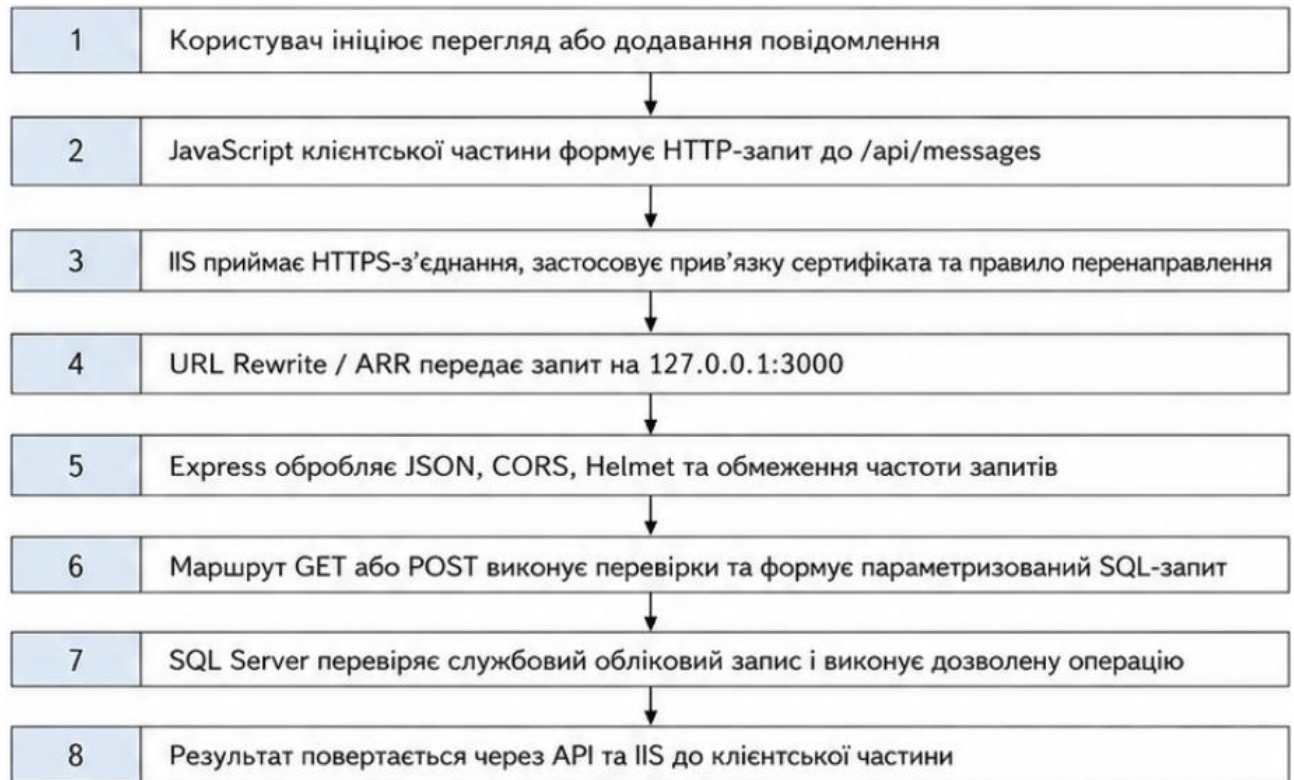


Рис. 2.2 – Послідовність обробки клієнтського запиту

Зовнішня та внутрішня адреси використовуються для різних завдань. Клієнтському коду потрібна лише адреса IIS, наприклад шлях /api/messages на вузлі 192.168.56.102. Адреса 127.0.0.1:3000 використовується тільки правилом IIS на Serv-WEB-IIS. Це не дві альтернативні адреси одного рівня для користувача, а дві частини послідовного маршруту. Зовнішня адреса веде до IIS, а внутрішня – від IIS до Node.js.

HTTP і HTTPS також не слід описувати як два незалежні способи доступу до різних компонентів. Обидва протоколи приймає IIS. На IIS використовується самопідписаний сертифікат, тому браузер може показувати попередження про довіру. Порт 80 може застосовуватися для початкового звернення з подальшим переходом на HTTPS. Точна адреса в app.js повинна бути узгоджена з налаштованим правилом, щоб клієнтський запит не обходив передбачений маршрут.

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		29

Зворотний трафік проходить у межах уже встановлених сеансів, тому для повернення відповіді не потрібно відкривати додаткові вхідні порти на основному комп'ютері.

Поточна схема допускає подальше розширення без зміни клієнтського маршруту. Якщо Node.js API буде перенесено на окремий сервер, у IIS зміниться лише адреса призначення для внутрішнього передавання. Клієнт і надалі звертатиметься до того самого шляху `/api/messages`. У такому випадку між IIS і новим вузлом API потрібно буде додатково налаштувати мережевий доступ, але логіка клієнтської частини залишиться незмінною.

Окрім адрес і портів, під час проєктування визначено межі(див. табл. 2.3), на яких один компонент передає дані іншому. Такі межі не означають, що сусідній компонент автоматично вважається безпечним. Їхнє призначення полягає в тому, щоб зафіксувати, які саме дані можуть переходити далі, хто ініціює з'єднання та яка перевірка виконується після переходу. Для цієї системи суттєвими є межі між браузером та IIS, IIS та Node.js API, а також Backend API та SQL Server.

Таблиця 2.3

Межі взаємодії та умови переходу між компонентами

Межа взаємодії	Дані, що передаються	Умова подальшої обробки	Наслідок порушення
Браузер - IIS	Метод, шлях, заголовки та JSON для POST	Запит надходить на дозволenu адресу IIS і відповідає формату HTTP	Запит відхиляється або не доходить до прикладного маршруту
IIS - Node.js API	Маршрут <code>/api/messages</code> , метод і тіло запиту	Правило URL Rewrite/ARR спрямовує звернення на 127.0.0.1:3000	Клієнт отримує помилку маршрутизації, але локальний порт не публікується
Node.js API - SQL Server	Параметризований SELECT або INSERT	Доступний TCP 1433, правильний <code>webapp_user</code> і необхідний дозвіл	SQL Server відмовляє в операції; клієнту повертається загальна помилка
SQL Server - Node.js API	Набір записів або результат виконання команди	Відповідь повертається в межах установленого сеансу	API не підтверджує операцію, доки база не завершила команду

На першій межі дані надходять із середовища, яке не контролюється сервером. Навіть якщо запит сформовано сторінкою з контейнера, браузер дозволяє

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		30

користувачу відкрити інструменти розробника, змінити JavaScript або надіслати звернення іншою програмою. Тому IIS приймає лише мережеве з'єднання, але не підтверджує коректність прикладних полів. Остаточне рішення про допустимість Username і MessageText залишається за Backend API.

Друга межа розміщена всередині Serv-WEB-IIS. IIS передає запит локальному HTTP-серверу, але не виконує його бізнес-логіку. Внутрішня адреса 127.0.0.1 обмежує коло можливих ініціаторів з'єднання самим серверним вузлом. Завдяки цьому зміна правила IIS впливає на доступність маршруту, але не на права SQL Server і не на перевірку даних у Node.js.

Третя межа відокремлює прикладний процес від рівня даних. SQL Server не аналізує, чи надійшов запит від браузера, IIS або іншого вебкомпонента. Для нього значення мають мережеве підключення, службовий логін, контекст бази та видані дозволи. Саме тому обмеження webapp_user є окремим проєктним рішенням: навіть коректний маршрут API не повинен надавати прикладному процесу більше можливостей, ніж потрібно для реалізованих GET і POST.

Архітектура також визначає очікувану поведінку за недоступності окремого компонента. Якщо не запущено клієнтський контейнер, браузер не отримає файли сторінки, але сервери IIS і SQL Server залишаються працездатними. Якщо зупинено IIS, зовнішній маршрут API стає недоступним, хоча локальний процес Node.js може продовжувати працювати. Зупинка Backend API призведе до помилки передавання запиту з IIS, а недоступність SQL Server дозволить Node.js прийняти HTTP-звернення, але не завершити операцію з даними.

Таке розділення станів потрібне не лише для пошуку несправностей. Воно запобігає хибному висновку, що успішне відкриття вебсторінки підтверджує роботу всієї системи. Завантаження HTML перевіряє клієнтський компонент, відповідь IIS перевіряє зовнішній маршрут, результат /api/messages підтверджує роботу Backend API, а фактична поява нового запису в MessagesTable підтверджує завершення операції на рівні даних. Отже, працездатність оцінюється послідовно, відповідно до меж взаємодії.

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		31

2.2 Проєктування клієнтського та прикладного компонентів

Клієнтський компонент є вебінтерфейсом для перегляду й додавання повідомлень. Він розміщується у Docker-контейнері, щоб його файли та середовище запуску формували окремий відтворюваний компонент. Контейнер можна зібрати з Dockerfile і запустити на іншому комп'ютері з Docker без ручного встановлення Express та копіювання файлів у системні каталоги вебсервера.

У середині контейнера використовується Node.js і невеликий застосунок Express. У цьому компоненті Node.js не виконує роль Backend API. Він запускає HTTP-процес, який передає браузеру вміст каталогу public. Такий процес потрібний тому, що сам Docker-контейнер не є вебсервером і без запущеної програми не зможе відповідати на запити. Замість Express міг бути використаний інший сервер статичних файлів, але для прототипу обрано знайоме JavaScript-середовище.

Основною частиною клієнтського компонента є каталог public. Файл index.html визначає структуру сторінки: область відображення повідомлень, поле імені, поле тексту та кнопку надсилання. Файл styles.css задає оформлення. Файл app.js містить функції, які надсилають GET- і POST-запити до API, обробляють JSON-відповідь та оновлюють сторінку. Цей код виконується у браузері, а не всередині контейнера після передавання файлів.

Адреса API в app.js є звичайною HTTP- або HTTPS-адресою сервісу IIS. Вона не є рядком підключення до SQL Server і не містить пароля бази даних. Клієнтський код повинен знати, куди надсилати вебзапити, але не повинен мати параметрів серверного підключення. Завдяки цьому зміна пароля webapp_user або конфігурації SQL Server не потребує зміни клієнтських файлів.

Dockerfile описує послідовність створення контейнера: вибір базового образу Node.js, копіювання package.json, встановлення залежностей, додавання server.js і каталогу public, а також команду запуску. На рівні проєктування достатньо зафіксувати, що результатом збірки є автономний компонент вебінтерфейсу з одним опублікованим портом.

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		32

Контейнеризація клієнтської частини не використовується як заміна всіх серверних засобів. IIS і SQL Server залишаються у Windows Server, оскільки саме там досліджуються їхні власні механізми конфігурації. Такий підхід дозволяє поєднати два способи розгортання в одному стенді: контейнер для вебінтерфейсу та віртуальні машини для серверних компонентів. Склад і призначення файлів клієнтського компонента узагальнено в табл. 2.4.

Таблиця 2.4

Структура клієнтського компонента

Елемент	Функція	Використання у системі
Dockerfile	Опис збирання образу та запуску контейнера	Створює однакове середовище для клієнтської частини.
package.json / package-lock.json	Опис залежності Express і зафіксованих версій пакетів	Дозволяє відтворити встановлення залежностей.
server.js	Запуск HTTP-процесу та передавання каталогу public	Робить клієнтські файли доступними браузеру.
public/index.html	Структура сторінки	Містить форму та область відображення повідомлень.
public/app.js	Взаємодія з API	Формує GET і POST до адреси IIS.
public/styles.css	Оформлення вебінтерфейсу	Визначає зовнішній вигляд елементів сторінки.

IIS розміщується на Serv-WEB-IIS і приймає зовнішні звернення до API. У цій роботі IIS не виконує бізнес-логіку роботи з повідомленнями та не формує SQL-запити. Його основна функція полягає в тому, щоб надати клієнту стабільну адресу, прийняти HTTP- або HTTPS-з'єднання та передати дозволений маршрут внутрішньому Node.js API.

Таке використання IIS відповідає ролі реверс-проксі. Клієнт надсилає запит IIS і не бачить локальної адреси Node.js. IIS створює новий внутрішній запит до 127.0.0.1:3000, отримує відповідь і повертає її клієнту. При цьому IIS залишається вебсервером, оскільки саме він приймає зовнішній HTTP-трафік. Проте прикладним вебсервером, який виконує маршрути API, є Node.js/Express.

Для передавання запитів використовуються URL Rewrite та Application Request Routing. URL Rewrite визначає, які зовнішні шляхи потрібно обробити, а ARR забезпечує передавання на інший HTTP-процес. У розділі проєктування не потрібно наводити точний синтаксис регулярного виразу або повний вміст

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		33

web.config. Важливим є правило: запити до API передаються локальному Node.js, а інші шляхи не повинні автоматично спрямовуватися до внутрішнього процесу.

На IIS налаштовується прив'язка сайту до портів 80 і 443. Порт 80 використовується для HTTP, а порт 443 – для HTTPS із сертифікатом. Правило переходу з HTTP на HTTPS повинно застосовуватися до зовнішнього звернення клієнта і не заважати локальному передаванню до Node.js. У практичній частині це перевіряється окремо: запит до HTTP-адреси має перейти на HTTPS або бути оброблений відповідно до налаштованого правила, а запит до /api/messages після передавання повинен зберігати метод GET чи POST.

Після налаштування зовнішнього маршруту окремо проєктується внутрішній процес Backend API.

Backend API запускається окремим процесом Node.js на Serv-WEB-IIS. Файл server.js створює застосунок Express, підключає необхідні модулі, визначає маршрути та запускає HTTP-сервер на 127.0.0.1:3000. Назва server.js не є формальною ознакою ролі компонента, вирішальним є те, що застосунок викликає listen і самостійно приймає HTTP-запити. Тому цей компонент правильно називати внутрішнім прикладним HTTP-сервером або API-сервером. Опис модулів бекенду описано в табл. 2.5.

Express організовує обробку запиту в послідовному порядку. Спочатку застосунок приймає HTTP-звернення і, для POST, перетворює JSON у доступний для програми об'єкт. Далі застосовуються загальні проміжні функції, після чого виконується маршрут GET або POST. Такий порядок дозволяє не повторювати однакові налаштування в кожному маршруті.

Модуль Helmet додає до відповіді набір HTTP-заголовків, які встановлюють безпечніші правила для браузера. У цьому проєкті Helmet не замінює перевірку введених даних і не впливає на права в SQL Server. Його призначення полягає в тому, щоб посилити стандартну HTTP-відповідь прикладного сервера без необхідності задавати кожний заголовок окремо.

Модуль express-rate-limit обмежує кількість звернень від одного клієнта за визначений проміжок часу. Для невеликого навчального API це захищає від дуже

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		34

частого повторення запитів і дозволяє перевірити реакцію сервісу на надмірну активність.

CORS потрібний через те, що вебінтерфейс і API завантажуються з різних адрес. Браузер порівнює джерело сторінки з адресою запиту й без відповідного дозволу може заблокувати доступ клієнтського JavaScript до відповіді. Модуль cors формує необхідні заголовки.

Бібліотека mssql забезпечує встановлення з'єднання з SQL Server, виконання команд і отримання результатів. Backend API не формує мережевий протокол SQL самостійно. Він передає бібліотеці конфігурацію та параметри запиту. Для користувача це приховано: браузер отримує JSON і не взаємодіє з SQL Server безпосередньо.

Таблиця 2.5

Використані модулі Backend API

Модуль	Призначення в проєкті	Результат використання
express	Створення HTTP-сервера і маршрутів API	Обробка GET і POST до /api/messages.
mssql	Підключення до SQL Server та виконання запитів	Отримання і додавання записів у QualificationDB.
dotenv	Завантаження параметрів із .env	Відокремлення адрес і облікових даних від коду.
helmet	Формування захисних HTTP-заголовків	Посилення стандартної відповіді API.
express-rate-limit	Обмеження частоти запитів	Повернення 429 за надмірної кількості звернень.
cors	Дозвіл браузерної взаємодії між різними адресами	Доступ клієнтського JavaScript до відповіді API.

Програмний інтерфейс має один основний шлях /api/messages і підтримує дві операції. Метод GET використовується для отримання наявних повідомлень, а POST – для додавання нового запису. Така структура відповідає невеликому обсягу функцій прототипу й дозволяє окремо перевірити читання та запис у базі даних.

GET-запит не потребує тіла. Після його отримання Backend API виконує вибірку з MessagesTable і формує масив об'єктів JSON. Кожний об'єкт містить ім'я користувача та текст повідомлення. Поле Id може використовуватися для внутрішнього впорядкування, але клієнтському інтерфейсу достатньо даних, які потрібно показати на сторінці.

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		35

POST-запит містить JSON з полями Username і MessageText. Перед зверненням до SQL Server API перевіряє наявність значень та їхню довжину. Обмеження до 50 символів для імені й до 100 символів для повідомлення узгоджені зі структурою таблиці. Така перевірка дає змогу відхилити некоректні дані до виконання SQL-команди та повернути користувачеві зрозумілу помилку.

Опис операцій наведено в табл. 2.6. У цьому контексті доцільніше говорити про операції програмного інтерфейсу, а не використовувати абстрактний термін «контракт». Таблиця показує, які дані надсилає клієнт і який результат очікується від кожного методу.

Таблиця 2.6

Операції програмного інтерфейсу

Метод і шлях	Вхідні дані	Обробка	Результат
GET /api/messages	Тіло запиту відсутнє	Вибірка записів із MessagesTable	JSON-масив повідомлень або повідомлення про помилку.
POST /api/messages	JSON: Username до 50 символів; MessageText до 100 символів	Перевірка значень і параметризований INSERT	Підтвердження додавання; 400 для некоректних даних; 429 при перевищенні ліміту.

Конфігурація Backend API поділяється на параметри запуску та параметри доступу до даних. Порт локального HTTP-сервера визначає, де IIS повинен знайти API. Параметри DB_SERVER і DB_DATABASE визначають вузол і базу, а DB_USER та DB_PASSWORD – обліковий запис застосунку. Таке розділення робить server.js зрозумілішим, оскільки в ньому залишається логіка, а не конкретні паролі й адреси.

Файл .env не є частиною клієнтського контейнера. Він розміщується на Serv-WEB-IIS поряд із Backend API або в іншому каталозі, з якого процес може отримати змінні.

Обробка помилок має зберігати зрозумілий результат для клієнта. Якщо введені дані неправильні, користувачеві потрібно повідомити про необхідність виправлення форми. Якщо SQL Server недоступний, достатньо повідомити, що операцію не виконано. Детальний текст системної помилки не повинен

відображатися на вебсторінці, оскільки він може містити назви файлів, адреси або внутрішні параметри.

2.3 Проектування бази даних

Рівень даних реалізується в SQL Server 2022 Express на окремій віртуальній машині Serv-DB-SQL. База QualificationDB призначена для зберігання повідомлень, які відображаються у клієнтській частині. Винесення SQL Server на окремий вузол дозволяє відокремити зберігання даних від приймання HTTP-запитів і не встановлювати СУБД на той самий сервер, де працюють IIS та Node.js.

У межах прототипу структура даних навмисно залишається простою. Основною таблицею є MessagesTable(див. табл. 2.7). Вона містить ідентифікатор запису, ім'я користувача та текст повідомлення. Такої моделі достатньо для перевірки повного циклу: введення даних у браузері, передавання POST-запиту, виконання INSERT, зберігання запису й подальше отримання через GET.

Поле Id має тип INT, автоматично збільшується та є первинним ключем. Воно забезпечує унікальність записів і дозволяє зберігати порядок їх створення. Username має тип NVARCHAR(50), а MessageText – NVARCHAR(100). Використання NVARCHAR дає змогу зберігати українські символи. Обидва текстові поля не повинні бути порожніми на рівні таблиці та перевіряються також у Backend API.

Обмеження довжини на рівні бази й API повинні бути узгоджені. Якщо API дозволить довший текст, SQL Server відхилить запит або виникне помилка збереження. Якщо API встановить меншу межу без пояснення, користувач не зможе використати всю передбачену довжину поля. Тому для проекту прийнято однакові значення: 50 символів для Username і 100 символів для MessageText.

Таблиця 2.7

Структура таблиці MessageTable

Поле	Тип даних	Обмеження	Призначення
Id	INT	IDENTITY(1,1), PRIMARY KEY	Унікальний ідентифікатор і порядок записів.

Username	NVARCHAR(50)	NOT NULL	Ім'я, яке відображається поруч із повідомленням.
MessageText	NVARCHAR(100)	NOT NULL	Текст повідомлення.

Backend API не використовує адміністративний обліковий запис SQL Server. Для нього створено окремий логін webapp_user, який зіставлено з користувачем у базі QualificationDB. Такий поділ потрібний для того, щоб пароль прикладного сервісу не надавав можливості керувати всім екземпляром SQL Server або змінювати його серверні налаштування.

Обліковий запис webapp_user використовується тільки Backend API. Клієнтський код не знає його імені та пароля, а IIS не підключається до бази даних самостійно. Якщо пароль потрібно змінити, оновлюються логін у SQL Server і серверна конфігурація Backend API; клієнтська частина при цьому не змінюється.

Backend API встановлює TCP-з'єднання з адресою 192.168.56.101 і портом 1433 за допомогою пакета mssql. Серверні параметри підключення передаються бібліотеці під час запуску, після чого вона створює сеанс у межах QualificationDB. IIS у цьому етапі не бере участі, оскільки його завдання завершується передаванням HTTP-запиту Node.js.

Права службового облікового запису вже налаштовано відповідно до фактичних операцій API. webapp_user має точкові дозволи SELECT та INSERT лише на об'єкт dbo.MessagesTable. Він не входить до широких ролей читання або запису всієї бази, не має дозволів UPDATE і DELETE та не може створювати, змінювати або видаляти таблиці. Отже, право SQL Server прямо відповідає маршруту, який реалізований у Backend API.

Таке налаштування є важливим незалежно від перевірок у Node.js. Маршрут API визначає, які операції повинні бути доступні клієнту, але саме SQL Server встановлює остаточну межу можливостей службового логіна. Якщо в коді випадково з'явиться команда UPDATE або DELETE, база даних відхилить її через відсутність дозволу. Це зменшує наслідки помилки в застосунку або компрометації його процесу.

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		38

Кінцевий користувач при цьому не отримує SQL-дозволів. Він може лише ініціювати GET або POST через HTTP, а Backend API виконує наперед визначену команду від імені webapp_user. Тому поняття прав клієнта й прав службового облікового запису не є тотожними. Навіть за анонімної моделі відвідувач не може передати власну SQL-команду, обрати іншу таблицю або змінити тип операції, якщо відповідного маршруту немає в серверному коді.

Відсутність UPDATE і DELETE також узгоджується з функціональною моделлю відкритої дошки повідомлень. Система не автентифікує кінцевих користувачів і не може надійно встановити власника запису. Надання редагування або видалення без перевірки авторства створило б можливість змінювати чужі повідомлення. Тому прототип підтримує тільки читання загального списку та додавання нового запису, а адміністративне очищення даних виконується окремо під контролем обліковим записом SQL Server.

У моделі допускається повторення однакових імен і текстів, оскільки Username є відображуваним підписом, а не унікальним обліковим записом. Введення окремого обмеження UNIQUE для цього поля суперечило б анонімній моделі й не дозволяло б різним відвідувачам використовувати однакове ім'я. Натомість унікальність забезпечується технічним полем Id, яке не залежить від введених користувачем даних.

Для ефективної роботи бібліотека повторно використовує з'єднання через пул. Практичний зміст цього механізму простий: Backend API не відкриває повністю нове підключення до SQL Server для кожного окремого повідомлення, якщо вже існує придатне з'єднання. Це зменшує затримки й кількість одночасних операцій встановлення сеансу.

Параметри підключення не виводяться у вебінтерфейс і не повертаються в JSON-відповіді. У разі помилки клієнт отримує загальне повідомлення, тоді як причина перевіряється на сервері. Це не вимагає складної системи моніторингу, але запобігає випадковому розкриттю пароля або повного рядка підключення користувачу.

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		39

Операція читання реалізується запитом SELECT до MessagesTable. Вона не містить значень, введених користувачем, і повертає перелік повідомлень. Для додавання використовується INSERT із двома параметрами: Username та MessageText. Значення передаються бібліотеці mssql окремо від тексту SQL-команди.

Параметризована передача означає, що введений користувачем рядок розглядається як значення поля, а не як частина SQL-команди. Наприклад, лапки або послідовність, схожа на SQL-оператор, повинні зберігатися як текст повідомлення. Це є основним захистом від SQL-ін'єкції в реалізованому API й доповнює перевірку довжини вхідних даних.

Перевірка на рівні Node.js і параметризація виконують різні завдання. Перевірка визначає, чи відповідає значення правилам застосунку: чи воно не порожнє та чи не перевищує довжину. Параметризація визначає безпечний спосіб передавання цього значення до SQL Server. Тому обидва механізми потрібні одночасно й не дублюють один одного.

Після виконання INSERT SQL Server повертає результат Backend API. API не повинен вважати операцію успішною до завершення команди в базі. Якщо SQL Server відхилив запит, клієнт отримує повідомлення про невиконану операцію, а сторінка не повинна показувати повідомлення як збережене. Такий порядок зберігає узгодженість між даними, що відображаються у браузері, та фактичним станом таблиці MessagesTable.

Serv-DB-SQL не виконує функцій IIS або Node.js. На цьому вузлі зосереджено SQL Server і засоби його налаштування. Такий розподіл спрощує адміністрування: параметри мережевого протоколу, порт 1433, база QualificationDB, логіни та ролі перевіряються в одному середовищі. Вебінтерфейс і прикладні файли не зберігаються на сервері бази даних.

Визначена структура бази даних і спосіб доступу до неї є основою для подальшого розподілу механізмів захисту між компонентами.

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				40
Змн.	Арк.	№ докум.	Підпис	Дата		

2.4 Проектування механізмів захисту

Механізми захисту в проєкті розподіляються між кількома компонентами. Жоден із них не розглядається як єдиний засіб безпеки. Мережеве розміщення обмежує доступність служб, IIS контролює зовнішній маршрут до API, Node.js перевіряє HTTP-запити, а SQL Server перевіряє обліковий запис і права в базі.

Контейнер клієнтської частини відокремлює її залежності й файли від інших компонентів системи. Це не слід подавати як абсолютну ізоляцію від основної операційної системи. Практичне значення контейнера полягає в тому, що вебінтерфейс збирається і запускається як окремий компонент з визначеним набором файлів та одним HTTP-процесом.

Облікові дані SQL Server зберігаються лише в серверній конфігурації Backend API на Serv-WEB-IIS. Клієнтському коду відома тільки адреса API, тому перегляд HTML і JavaScript не розкриває службовий логін, пароль або параметри прямого підключення до бази даних.

Незалежно від способу формування запиту Backend API перевіряє структуру JSON, наявність полів Username і MessageText та їхню допустиму довжину. Клієнтські обмеження й правильне проходження запиту через IIS не замінюють цього серверного контролю, оскільки звернення може бути сформоване вручну або змінене до надсилання.

Контейнер публікує лише порт клієнтського вебінтерфейсу. Зовнішній доступ до API здійснюється через IIS, тоді як Node.js прив'язаний до 127.0.0.1:3000. URL Rewrite та ARR передають локальному процесу тільки запити до передбаченого шляху API, тому контейнер не створює додаткових маршрутів до прикладного сервера або SQL Server.

IIS і Node.js розміщені на одному вузлі, тому внутрішнє HTTP-з'єднання через loopback не виходить у Host-Only мережу. Це зменшує кількість мережових ділянок для одного запиту. Водночас Node.js залишається окремим процесом, а отже, його запуск, зупинка та помилки можна перевіряти незалежно від стану IIS.

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		41

Доступ до SQL Server контролюється двома незалежними умовами: мережевою доступністю порту 1433 і дозволами облікового запису в QualificationDB. Виконання лише однієї з них не надає доступу до таблиць.

Використання окремого webapp_user обмежує повноваження Backend API порівняно з адміністративним обліковим записом sa та зменшує можливі наслідки витоку прикладного пароля.

Для перевірки повноти захисту кожний реалізований механізм зіставлено з конкретною загрозою. Це дозволяє не оцінювати безпеку лише за кількістю використаних бібліотек. Механізм вважається обґрунтованим тоді, коли зрозуміло, який небажаний шлях або наслідок він обмежує. Відповідність основних загроз і засобів контролю наведено в табл. 2.8.

Таблиця 2.8

Відповідність загроз механізмам захисту

Загроза або небажаний стан	Компонент контролю	Реалізований механізм	Очікуваний результат
Пряме звернення клієнта до Node.js	Serv-WEB-IIS	Прив'язка API до 127.0.0.1:3000 та доступ через IIS	Порт 3000 не приймає зовнішнє з'єднання
Прямий доступ браузера до SQL Server	Serv-DB-SQL / Backend API	Відсутність SQL-параметрів у клієнті та робота лише через API	Клієнт не виконує SQL-команди
SQL-ін'єкція через поля повідомлення	Node.js / mssql	Параметризований INSERT	Введення обробляється як значення, а не частина команди
Некоректні або завеликі дані	Backend API	Перевірка структури JSON і довжини полів	Запит відхиляється до звернення до бази
Надмірна кількість звернень	Backend API	express-rate-limit	Після перевищення ліміту повертається 429
Запит із непередбаченого браузерного джерела	Backend API / браузер	CORS-заголовки	Браузер не надає JavaScript доступу до відповіді
Надлишкові повноваження прикладного логіна	SQL Server	Лише SELECT та INSERT на dbo.MessagesTable	UPDATE, DELETE і зміна структури відхиляються
Розкриття внутрішніх параметрів через помилку	Backend API	Загальна відповідь клієнту та серверна обробка помилки	Клієнт не отримує пароль або повний рядок підключення

Наведені механізми утворюють послідовний захист, але не дублюють один одного. Локальна прив'язка Node.js визначає, звідки можливе мережеве з'єднання; IIS визначає зовнішній маршрут; Backend API перевіряє форму прикладних даних; параметризація визначає спосіб передавання значень у SQL-команду; дозволи webapp_user обмежують операції, які SQL Server узагалі погодиться виконати. Відмова одного контролю не повинна автоматично надавати повний доступ до наступного рівня.

Самопідписаний сертифікат використовується для перевірки HTTPS і маршруту через IIS. Він забезпечує шифрування з'єднання після прийняття сертифіката, але не має автоматично підтвердженої довіри, як сертифікат від загальновизнаного центру сертифікації.

Host-Only мережа також є властивістю стенда, а не моделлю доступу для публічного вебсервісу. Вона обмежує видимість віртуальних машин локальним хостом і полегшує контроль адрес, проте не демонструє захист від повного набору зовнішніх мережевих загроз. Практичний результат цього рішення полягає в тому, що під час роботи над прототипом сервери не публікуються у фізичній мережі, а всі необхідні переходи можна перевірити в межах контрольованого сегмента.

Залишковий ризик оцінюється відповідно до призначення прототипу. Анонімний користувач може вказати чуже відображуване ім'я або надіслати небажаний текст, тому система не повинна подавати Username як підтверджене авторство. Обмеження довжини та частоти зменшують зловживання, але не виконують змістовну модерацію. Це не суперечить архітектурі стенда, якщо в роботі чітко зазначено, що повідомлення є відкритими, а прикладна функція використовується для демонстрації захищеної взаємодії компонентів.

Таким чином, реалізована конфігурація не намагається вирішити всі можливі завдання безпеки однією технологією. Її мета полягає в тому, щоб для кожного переходу застосувати відповідний контроль: локалізувати прикладний порт, спрямувати зовнішній трафік через IIS, перевірити запит у Node.js, безпечно передати значення до SQL Server і обмежити службовий логін двома необхідними

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		43

операціями. Такий підхід робить захисні рішення перевірюваними й пов'язаними з реальною функціональністю системи.

Висновки до розділу

У другому розділі спроектовано багатокомпонентну вебінфраструктуру, яка складається з контейнеризованої клієнтської частини, зовнішнього рівня IIS, внутрішнього Node.js/Express API та SQL Server на окремій віртуальній машині. Для взаємодії вузлів використовується Host-Only мережа 192.168.56.0/24 зі статичними адресами 192.168.56.1, 192.168.56.102 та 192.168.56.101.

Визначено послідовність проходження запиту. Браузер завантажує вебінтерфейс із Docker-контейнера та звертається до IIS. IIS приймає зовнішній HTTP- або HTTPS-запит і передає маршрут /api/messages локальному Node.js API на 127.0.0.1:3000. Backend API виконує перевірку даних, формує параметризовану SQL-команду та підключається до SQL Server через TCP 1433.

Окремо спроектовано програмні компоненти. Клієнтська частина містить HTML, CSS і JavaScript, а Node.js/Express у контейнері використовується тільки для її передавання браузеру. IIS виконує роль зовнішньої точки входу та реверс-проксі. Node.js/Express на Serv-WEB-IIS є внутрішнім прикладним HTTP-сервером, який реалізує GET і POST для роботи з повідомленнями.

Рівень даних представлений базою QualificationDB і таблицею MessagesTable. Для прикладного доступу використовується webapp_user, якому вже надано лише дозволи SELECT та INSERT.

Підготовлена архітектура є достатньою для практичного розгортання й поетапної перевірки. Кожний компонент має визначене призначення, а несправність або помилкове налаштування можна локалізувати на конкретній ділянці: клієнтському контейнері, IIS, Node.js API або SQL Server.

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		44

РОЗДІЛ 3 РЕАЛІЗАЦІЯ ТА ПЕРЕВІРКА ПІДСИСТЕМИ БЕЗПЕКИ

3.1 Розгортання компонентів вебінфраструктури

Механізми захисту в проєкті розподіляються між кількома компонентами.

Практична реалізація розпочинається з підготовки ізолюваного мережевого середовища та встановлення програмних компонентів на відповідних вузлах. Компоненти розміщено окремо відповідно до їхнього призначення: клієнтська частина працює в Docker-контейнері, IIS і Node.js API розгорнуто на вебвузлі, а SQL Server - на окремій віртуальній машині.

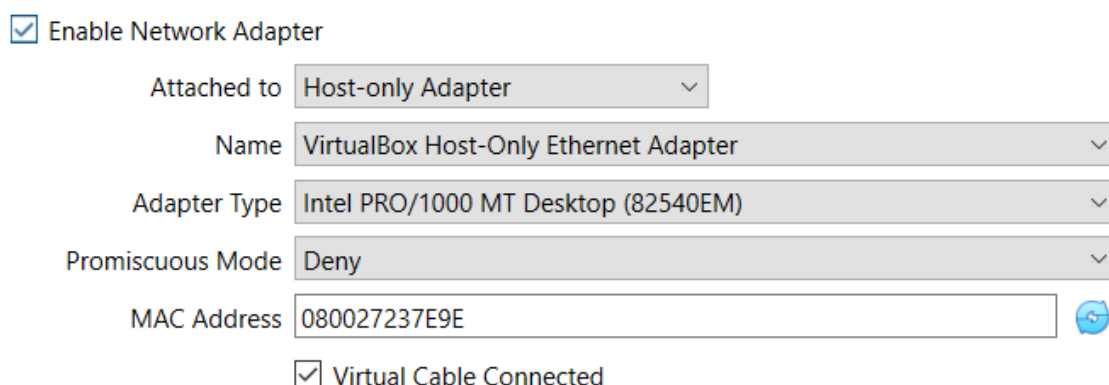
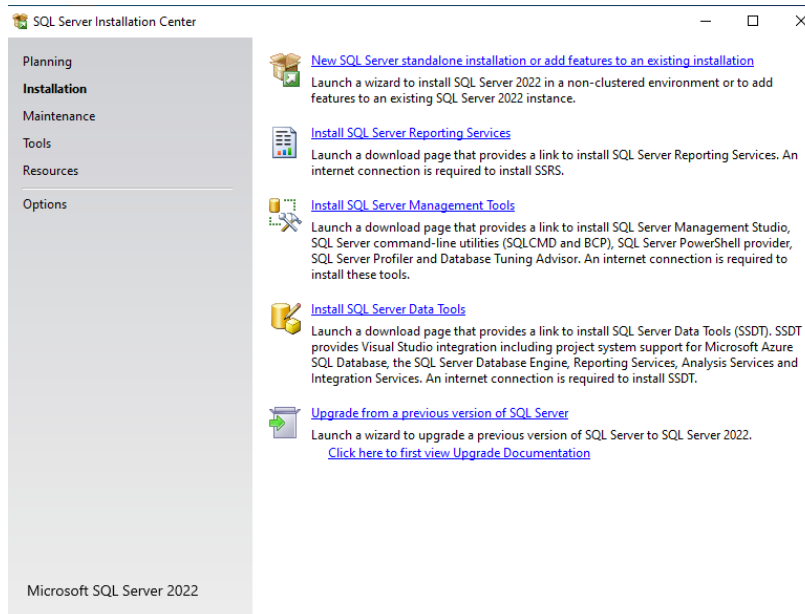


Рис. 3.1 – Налаштування мережевого адаптера Host-Only у VirtualBox

Для обох серверних віртуальних машин використано мережевий режим Host-Only Adapter. Він забезпечує обмін даними між основним комп'ютером і віртуальними машинами в окремій мережі 192.168.56.0/24, але не публікує сервери безпосередньо у фізичній локальній мережі.



		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				45
Змн.	Арк.	№ докум.	Підпис	Дата		

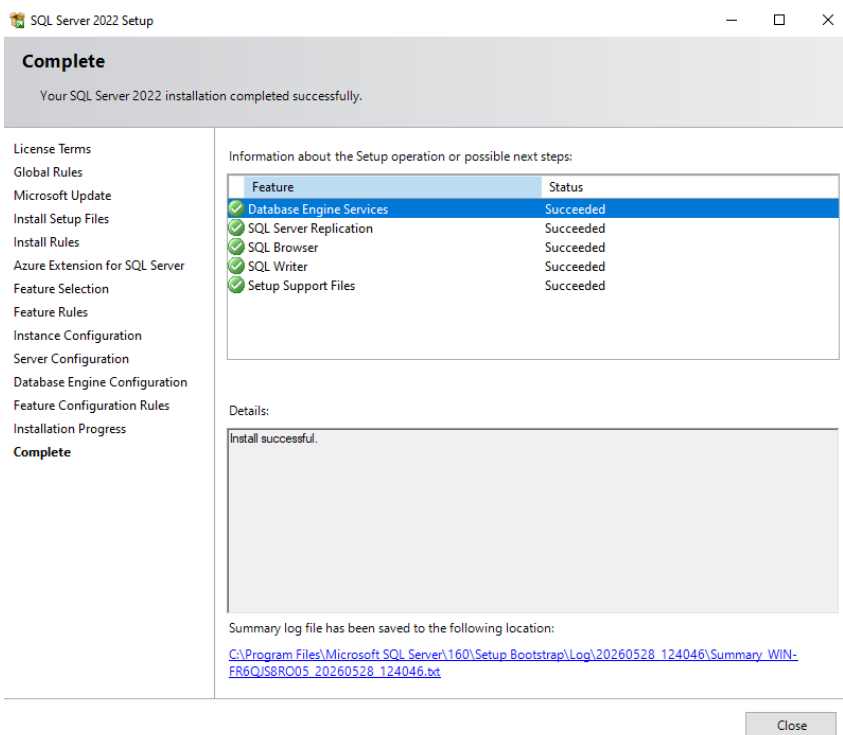
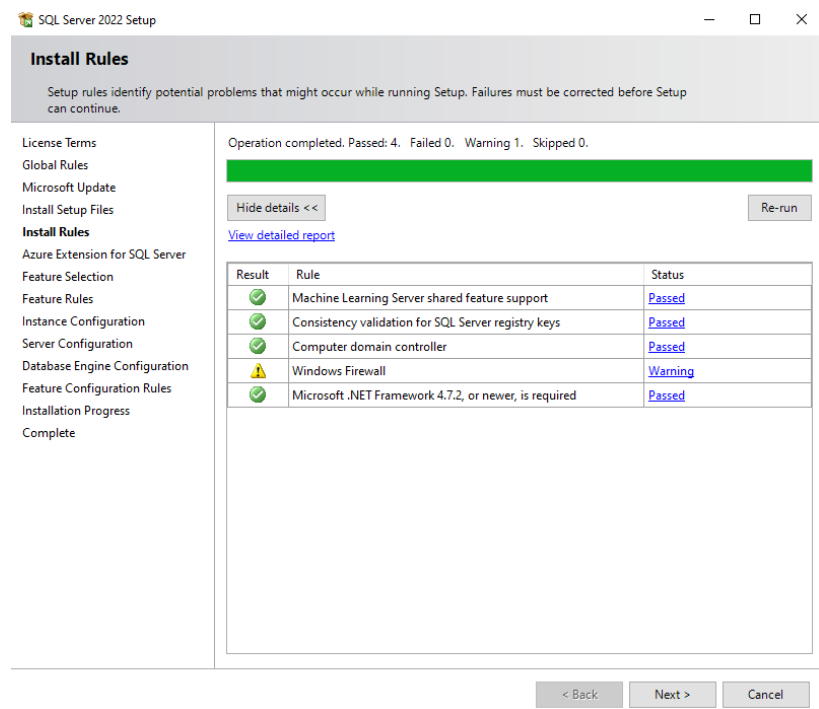


Рис. 3.2 – 3.4 – Основні етапи встановлення Microsoft SQL Server 2022 Express

На вузлі Serv-DB-SQL встановлено SQL Server 2022 Express із компонентом Database Engine Services. Для екземпляра задано ім'я SQLEXPRESS, а під час налаштування автентифікації обрано змішаний режим, який дозволяє використовувати як облікові записи Windows, так і окремі логіни SQL Server. Завершальний етап підтверджує успішне встановлення служби бази даних. Повне встановлення Microsoft SQL Server 2022 подано в додатку А.

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				46
Змн.	Арк.	№ докум.	Підпис	Дата		

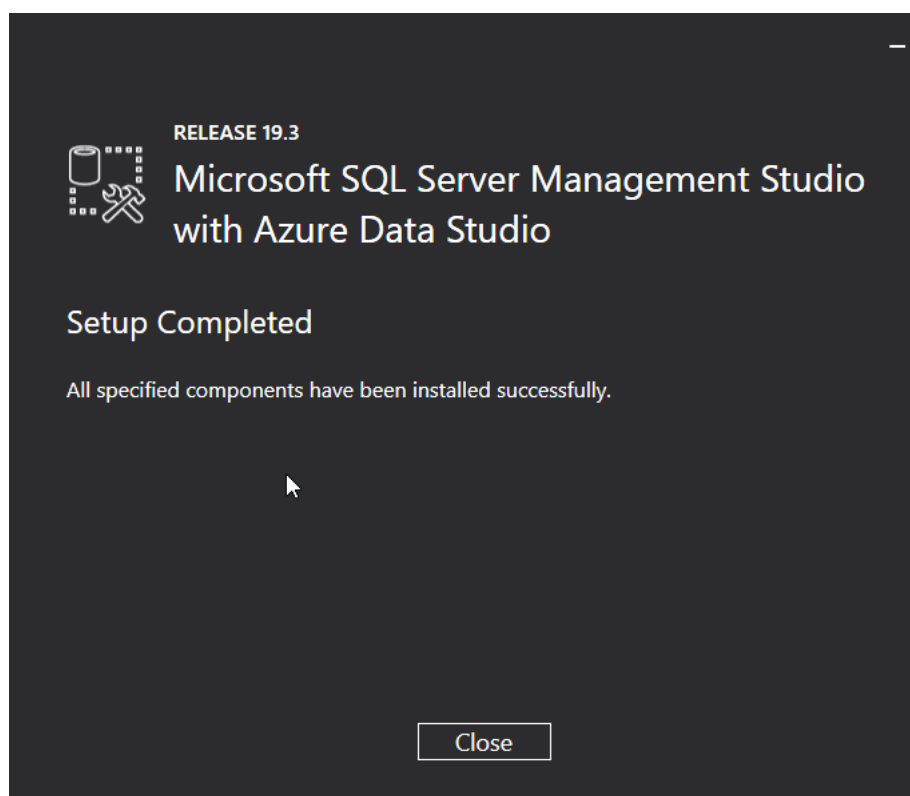
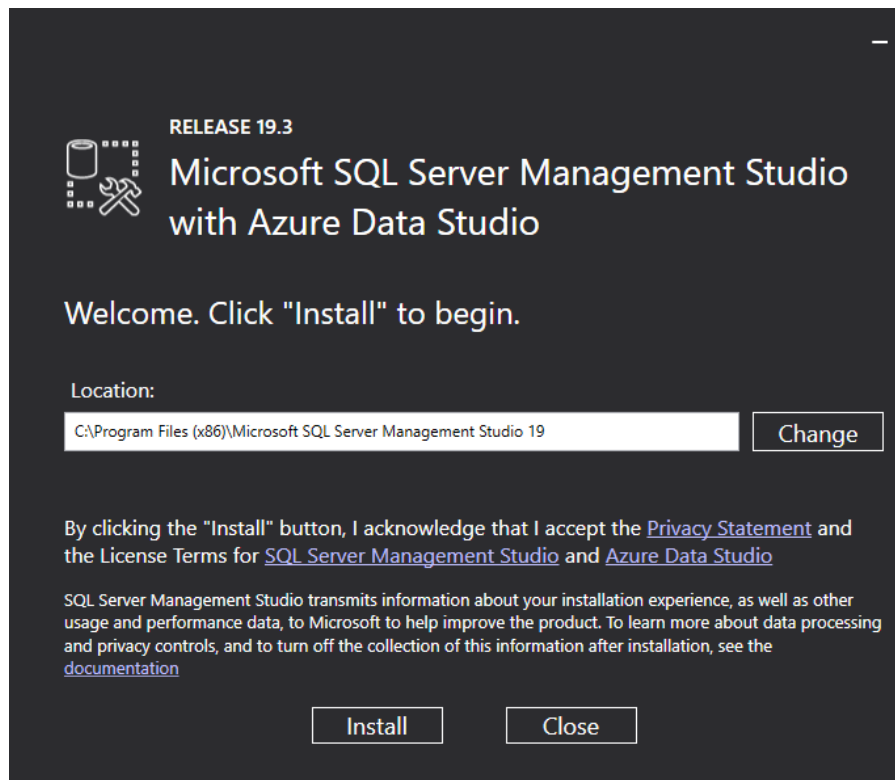


Рис. 3.5 - 3.6 – Встановлення Microsoft SQL Server Management Studio

Для адміністрування SQL Server додатково встановлено SQL Server Management Studio. Це середовище використовується для створення бази даних, таблиць, службового облікового запису застосунку та перевірки дозволів. SSMS не

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		47

бере участі у штатному потоці запитів вебзастосунку й використовується лише адміністратором під час налаштування та контролю.

Add Roles and Features Wizard

Before you begin

Before You Begin

Installation Type

Server Selection

Server Roles

Features

Confirmation

Results

DESTINATION SERVER
WIN-FR6QJS8RO05

This wizard helps you install roles, role services, or features. You determine which roles, role services, or features to install based on the computing needs of your organization, such as sharing documents, or hosting a website.

To remove roles, role services, or features:
[Start the Remove Roles and Features Wizard](#)

Before you continue, verify that the following tasks have been completed:

- The Administrator account has a strong password
- Network settings, such as static IP addresses, are configured
- The most current security updates from Windows Update are installed

If you must verify that any of the preceding prerequisites have been completed, close the wizard, complete the steps, and then run the wizard again.

To continue, click Next.

☐ Skip this page by default

< Previous

Next >

Install

Cancel

Add Roles and Features Wizard

Select destination server

Before You Begin

Installation Type

Server Selection

Server Roles

Features

Confirmation

Results

DESTINATION SERVER
WIN-FR6QJS8RO05

Select a server or a virtual hard disk on which to install roles and features.

☒ Select a server from the server pool

☐ Select a virtual hard disk

Server Pool

Filter:

Name	IP Address	Operating System
WIN-FR6QJS8RO05	192.168.56.102	Microsoft Windows Server 2022 Standard Evaluation

1 Computer(s) found

This page shows servers that are running Windows Server 2012 or a newer release of Windows Server, and that have been added by using the Add Servers command in Server Manager. Offline servers and newly-added servers from which data collection is still incomplete are not shown.

< Previous

Next >

Install

Cancel

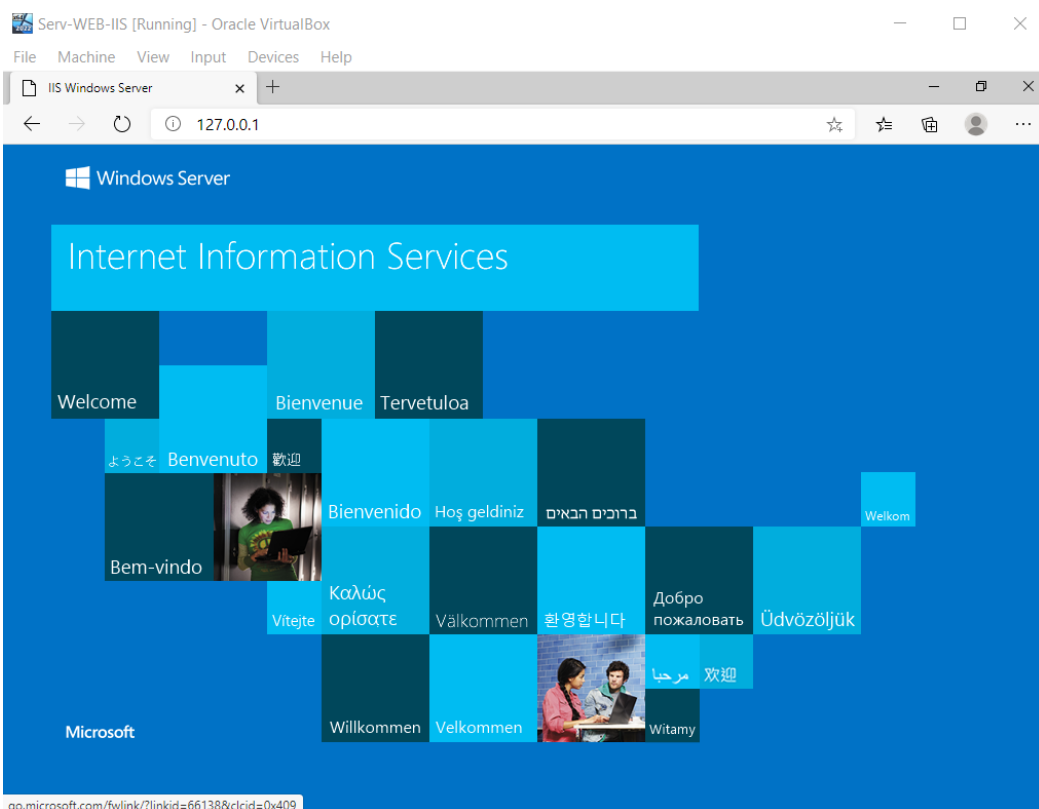
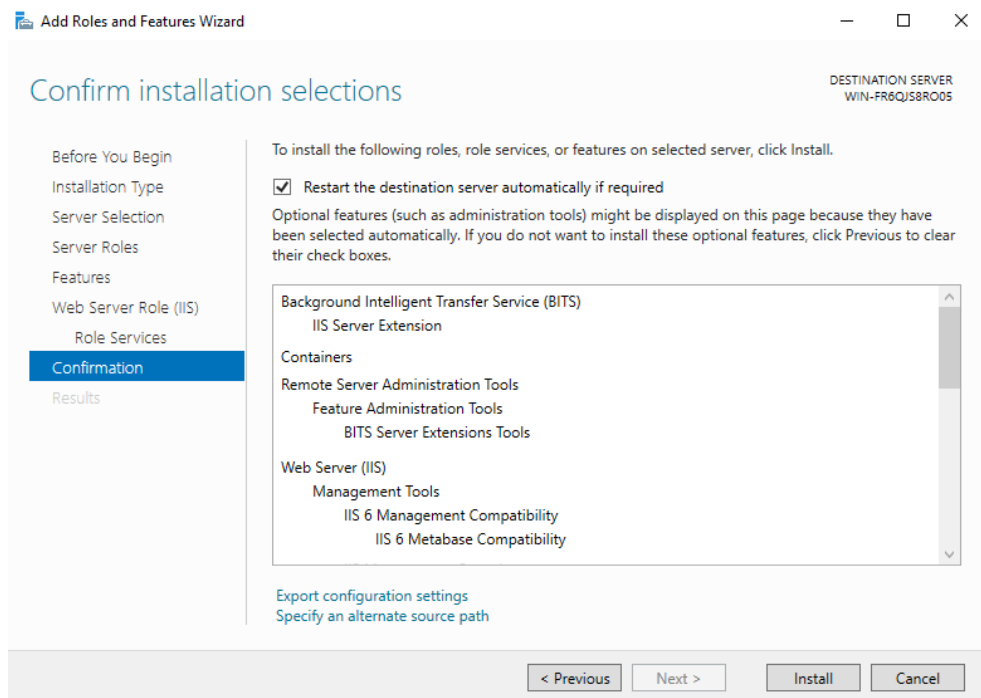
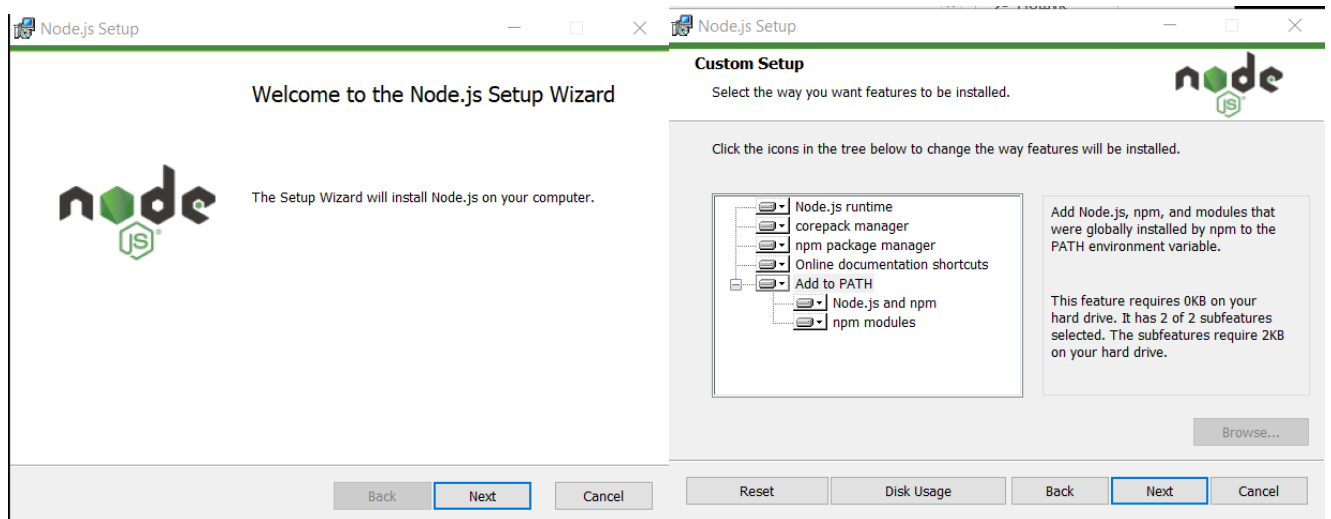


Рис. 3.7 -3.10 – Встановлення ролі Web Server (IIS) на вузлі Serv-WEB-IIS

На вебвузлі через Server Manager встановлено роль Web Server (IIS) із засобами керування. Після завершення інсталяції працездатність IIS перевірено відкриттям стандартної сторінки на локальній адресі. У подальшій конфігурації IIS використовується як зовнішня точка приймання запитів і реверс-проксі для внутрішнього Node.js API. Повний процес встановлення наведений в додатку Б.

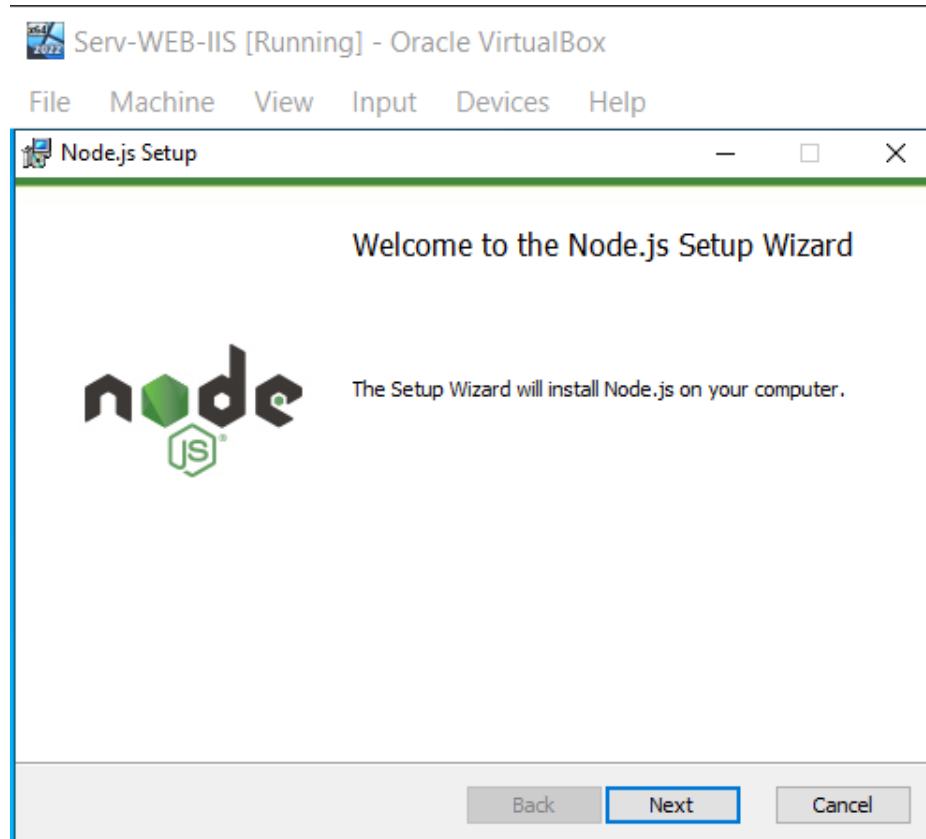
		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		49



```
PS C:\Users\zarut> npm -v
11.13.0
PS C:\Users\zarut> node -v
v24.16.0
```

Рис. 3.11-3.13 – Встановлення Node.js та перевірка версій Node.js і npm

На вузлі Serv-WEB-IIS встановлено середовище Node.js разом із менеджером пакетів npm. Після встановлення перевірено доступність команд node і npm та їхні версії. Це середовище використовується для запуску внутрішнього прикладного HTTP-сервера на основі Express.



		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				50
Змн.	Арк.	№ докум.	Підпис	Дата		

```

Serv-WEB-IIS [Running] - Oracle VirtualBox
File Machine View Input Devices Help

Administrator: Windows PowerShell

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Administrator> cd C:\BackendAPI
PS C:\BackendAPI> node -v
v24.16.0
PS C:\BackendAPI> npm init -y
Wrote to C:\BackendAPI\package.json:

{
  "name": "backendapi",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "type": "commonjs"
}

PS C:\BackendAPI>

PS C:\BackendAPI> npm install express mssql dotenv helmet express-rate-limit cors
added 140 packages, and audited 141 packages in 49s

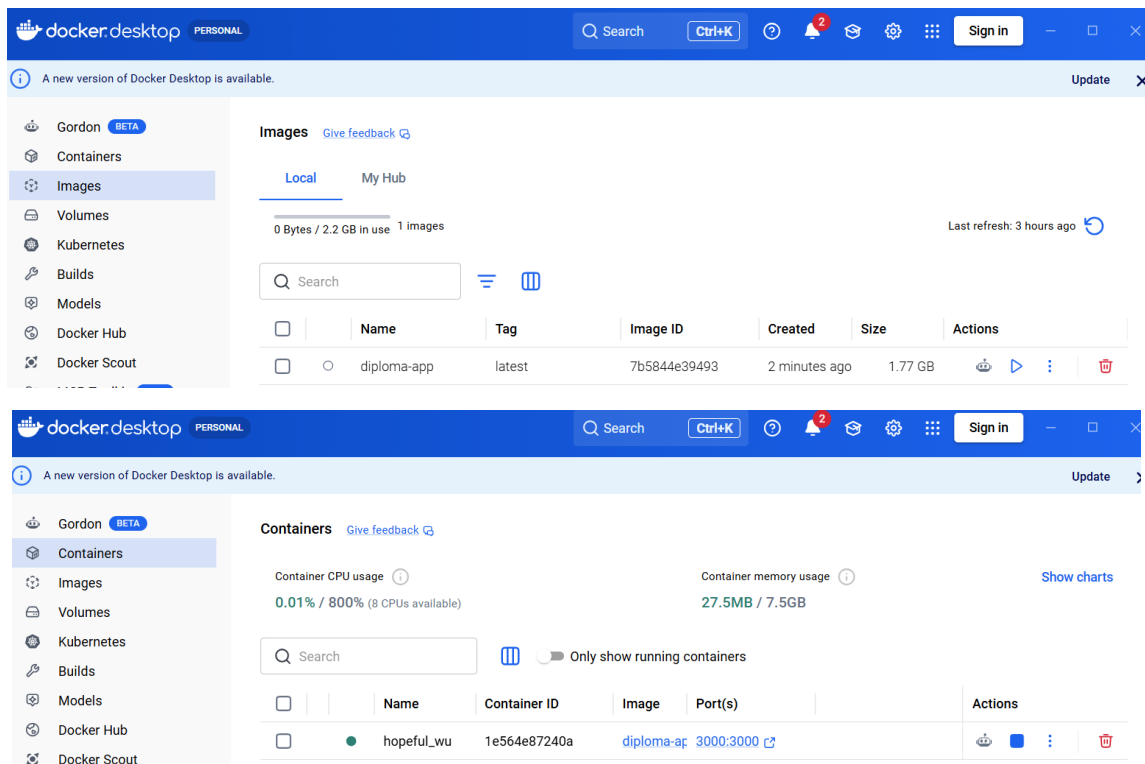
42 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS C:\BackendAPI>

```

Рис. 3.14-3.16 – Створення Backend API та встановлення серверних залежностей

Для серверного компонента створено окремий каталог Backend API. У ньому ініціалізовано Node.js-проект і встановлено бібліотеки Express, mssql, dotenv, Helmet, express-rate-limit та cors. Такий склад залежностей відповідає фактичним завданням компонента: обробці HTTP-запитів, підключенню до SQL Server, завантаженню параметрів середовища та застосуванню прикладних механізмів захисту.



```
PS D:\dockerdocker\diploma-app> docker build -t diploma-app .
[+] Building 61.3s (10/10) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 162B
=> [internal] load metadata for docker.io/library/node:20
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/5] FROM docker.io/library/node:20@sha256:8f693eaa7e0a8e71560c9a82b55fd54c2ae920a2ba5d2cde28bac7d1c01c9ba5
=> => resolve docker.io/library/node:20@sha256:8f693eaa7e0a8e71560c9a82b55fd54c2ae920a2ba5d2cde28bac7d1c01c9ba5
=> [internal] load build context
=> => transferring context: 48.60MB
=> CACHED [2/5] WORKDIR /app
=> [3/5] COPY package.json ./
=> [4/5] RUN npm install
=> [5/5] COPY . .
=> exporting to image
=> => exporting layers
=> => exporting manifest sha256:2709b81e9036ba93a82ce2ba260a4d2296d36b6e53f974781ff75278bcf8fc8b
=> => exporting config sha256:0cd1d074b88ae0f886a24f65b43c795934da60412cc21fc14be7c5f08b6564ae
=> => exporting attestation manifest sha256:37a8a691d95450ffc86549b7aa6ecb5068b2f888d4ca48a16f40f3fd3a6c9c9
=> => exporting manifest list sha256:118c3e6385f66ffdb67eef5256db26606009ebd1cc978d7c866192383f7eb
=> => naming to docker.io/library/diploma-app:latest
=> => unpacking to docker.io/library/diploma-app:latest
PS D:\dockerdocker\diploma-app> docker run -d -p 3000:3000 diploma-app
73c6a1b6d8c67523fe0382d2b4dcd042823420c094c78e32c626b3b8d3853596
PS D:\dockerdocker\diploma-app>
```

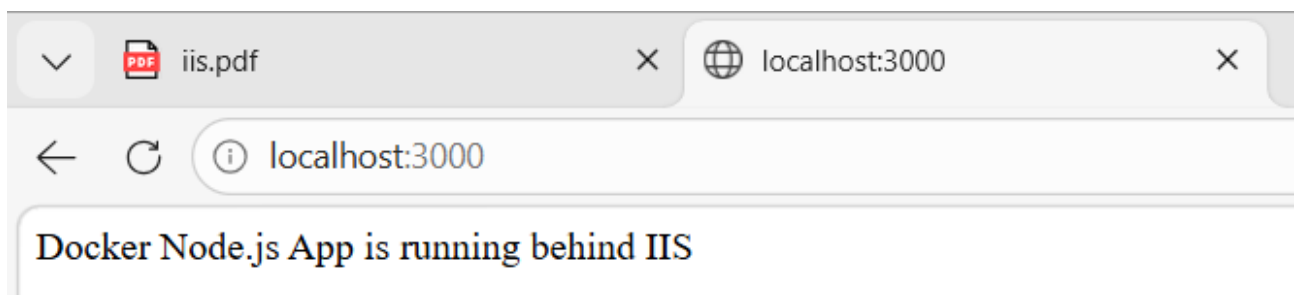


Рис. 3.17-3.20 – Створення Docker-образу та запуск контейнера клієнтської частини

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				52
Змн.	Арк.	№ докум.	Підпис	Дата		

Клієнтську частину з файлами HTML, CSS і JavaScript зібрано в окремий Docker-образ. Після запуску контейнера його порт опубліковано на основному вузлі, завдяки чому вебінтерфейс відкривається в браузері за локальною адресою. Node.js усередині цього контейнера виконує лише допоміжну функцію передавання файлів клієнтської частини браузеру.

Нижче наведено основні лістинги, які використовуються під час розгортання клієнтського та серверного компонентів.

Файли index.html та styles.css визначають структуру й зовнішнє оформлення вебсторінки: область відображення повідомлень, поля введення та кнопку надсилання. Оскільки вони не містять основної прикладної логіки та не реалізують ключових механізмів захисту системи, їх детальний буде наведено в додатку В.

Лістинг коду server.js(основний вузол):

```
const express = require('express');
const app = express();
app.use(express.static('public'));
app.listen(80, () => {
  console.log('UI running');
});
```

У контейнерному файлі server.js команда require('express') підключає бібліотеку Express, а виклик express() створює об'єкт вебзастосунку. Команда app.use(express.static('public')) реєструє проміжну функцію, яка шукає запитовані файли в каталозі public. Виклик app.listen(80) відкриває HTTP-порт усередині контейнера та переводить процес у режим очікування запитів.

Лістинг коду app.js(основний вузол):

```
const API = "http://192.168.56.102/api/messages";
async function load() {
  const res = await fetch(API);
  const data = await res.json();
  list.innerHTML = data.map(x =>
    `<p><b>${x.Username}</b>: ${x.MessageText}</p>`
  ).join('');
}
async function send() {
  await fetch(API, {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({
      Username: user.value,
      MessageText: msg.value
    })
  });
}
```

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		53

```

    })
  });
  load();
}
load();

```

Конструкція `const API = "http://192.168.56.102/api/messages";` оголошує константу `API`, у якій зберігається адреса серверного маршруту для отримання та додавання повідомлень. Ключове слово `const` забороняє повторне присвоєння іншого значення цій змінній. Адреса `192.168.56.102` відповідає вебвузлу з `IIS`, а шлях `/api/messages` визначає ресурс `Backend API`, з яким взаємодіє клієнтська частина.

Ключове слово `async` перед оголошенням функції `load()` указує, що всередині неї виконуються асинхронні операції. Команда `await fetch(API)` надсилає `HTTP`-запит методом `GET` за адресою, збереженою в константі `API`, і призупиняє подальше виконання функції до отримання відповіді. Отриманий об'єкт відповіді записується в константу `res`. Вираз `await res.json()` зчитує тіло `HTTP`-відповіді та перетворює `JSON`-масив у `JavaScript`-об'єкт, який зберігається у змінній `data`.

Властивість `list.innerHTML` задає `HTML`-вміст елемента сторінки, на який посилається змінна `list`. Метод `data.map()` послідовно обробляє кожний об'єкт масиву повідомлень. Параметр `x` представляє поточний запис. Шаблонний рядок, обмежений зворотними лапками, створює для кожного запису елемент `<p>`. Вираз `${x.Username}` підставляє значення поля `Username`, а `${x.MessageText}` - значення поля `MessageText`. Тег `<b>` виділяє ім'я автора напівжирним шрифтом. Метод `map()` повертає масив сформованих рядків, а `join("")` об'єднує їх в один рядок без додаткових розділювачів. Отриманий `HTML`-код вставляється в область відображення повідомлень.

Функція `send()` також оголошена з ключовим словом `async`, оскільки надсилання даних виконується асинхронно. Виклик `fetch(API, { ... })` надсилає запит до того самого маршруту `API`, але другий аргумент визначає його параметри. Властивість `method: "POST"` задає метод `POST`, призначений для передавання нового повідомлення серверу. Об'єкт `headers` містить заголовок `Content-Type`:

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		54

application/json, який повідомляє Backend API, що тіло запиту передано у форматі JSON.

Властивість `body` визначає тіло POST-запиту. Метод `JSON.stringify()` перетворює JavaScript-об'єкт у JSON-рядок. Поле `Username` отримує значення з властивості `user.value`, тобто з поля введення імені, а поле `MessageText` - значення `msg.value` з поля введення повідомлення. Оператор `await` очікує завершення надсилання запиту. Після цього викликається функція `load()`, яка повторно отримує дані з сервера та оновлює список повідомлень на сторінці.

Остання команда `load()`; викликає функцію завантаження одразу після виконання клієнтського сценарію. Завдяки цьому наявні повідомлення автоматично відображаються під час початкового відкриття вебсторінки, а не лише після натискання кнопки надсилання.

Лістинг коду `Dockerfile`(основний вузол):

```
FROM node:20
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN chown -R node:node /app
USER node
EXPOSE 80
CMD ["node", "server.js"]
```

У `Dockerfile` команда `FROM` вибирає базовий образ `Node.js`. Команда `WORKDIR` задає робочий каталог усередині образу. Інструкція `COPY` переносить `package.json`, `server.js` і каталог `public` до файлової системи образу. Команда `RUN npm install` виконується під час складання образу та встановлює залежності. Інструкція `EXPOSE 80` документує внутрішній порт застосунку, але сама по собі не відкриває його на хості. Команда `CMD` визначає процес, який запускається під час старту контейнера.

Команда `docker build -t ui-client .` читає `Dockerfile` з поточного каталогу, послідовно виконує його інструкції та створює образ із заданим тегом. Команда `docker run -p 8080:80 ui-client` створює контейнер із цього образу. Параметр `-p` зв'язує порт основного комп'ютера з портом 80 усередині контейнера, тому браузер може відкрити клієнтську сторінку.

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		55

Лістинг коду server.js(Serv-WEB-IIS):

```
require('dotenv').config();
const express = require('express');
const sql = require('mssql');
const helmet = require('helmet');
const { rateLimit, ipKeyGenerator } = require('express-rate-limit');
const cors = require('cors');
const app = express();
app.set('trust proxy', 'loopback');
app.disable('x-powered-by');
const allowedOrigins = [
  'http://localhost:8080'
];
app.use(cors({
  origin(origin, callback) {
    if (!origin || allowedOrigins.includes(origin)) {
      return callback(null, true);
    }
    return callback(new Error('CORS_ORIGIN_NOT_ALLOWED'));
  },
  methods: ['GET', 'POST', 'OPTIONS'],
  allowedHeaders: ['Content-Type']
}));
app.use(express.json());
app.use(helmet());
app.use(rateLimit({
  windowMs: 15 * 60 * 1000,
  max: 100,
  standardHeaders: true,
  legacyHeaders: false,
  keyGenerator: req => {
    let ip = req.ip || req.socket.remoteAddress || 'unknown';
    const ipv4WithPort = ip.match(
      /^(\\d{1,3}(?:\\.\\d{1,3}){3}):\\d+$/
    );
    if (ipv4WithPort) {
      ip = ipv4WithPort[1];
    }
    const ipv6WithPort = ip.match(
      /^\\[([0-9a-fA-F:]+)\\]:\\d+$/
    );
    if (ipv6WithPort) {
      ip = ipv6WithPort[1];
    }
    return ipKeyGenerator(ip);
  },
  message: 'Too many requests'
}));
const dbConfig = {
  user: process.env.DB_USER,
  password: process.env.DB_PASSWORD,
```

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		56

```

server: process.env.DB_SERVER,
database: process.env.DB_DATABASE,
options: {
  trustServerCertificate: true
}
};
app.get('/api/messages', async (req, res) => {
  try {
    const pool = await sql.connect(dbConfig);
    const result = await pool.request().query(`
      SELECT Username, MessageText FROM MessagesTable
    `);
    return res.json(result.recordset);
  } catch (err) {
    console.error(err);
    return res.status(500).send('Server error');
  }
});
app.post('/api/messages', async (req, res) => {
  console.log('POST WORKS:', req.body);
  const username =
    typeof req.body?.Username === 'string'
      ? req.body.Username.trim()
      : '';
  const messageText =
    typeof req.body?.MessageText === 'string'
      ? req.body.MessageText.trim()
      : '';
  if (username.length < 1 || username.length > 50) {
    return res.status(400).send(
      'Username must contain from 1 to 50 characters'
    );
  }
  if (messageText.length < 1 || messageText.length > 100) {
    return res.status(400).send(
      'MessageText must contain from 1 to 100 characters'
    );
  }
  try {
    const pool = await sql.connect(dbConfig);
    await pool.request()
      .input('Username', sql.NVarChar(50), username)
      .input('MessageText', sql.NVarChar(100), messageText)
      .query(`
        INSERT INTO MessagesTable (Username, MessageText)
        VALUES (@Username, @MessageText)
      `);
    return res.status(200).send('Inserted');
  } catch (err) {
    console.error(err);
    return res.status(500).send('DB error');
  }
});

```

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		57

```

    }
  });
  app.use((err, req, res, next) => {
    if (err?.message === 'CORS_ORIGIN_NOT_ALLOWED') {
      return res.status(403).send('CORS origin is not allowed');
    }
    if (err instanceof SyntaxError && err.status === 400 && 'body' in
err) {
      return res.status(400).send('Invalid JSON');
    }
    console.error(err);
    return res.status(500).send('Server error');
  });
  console.log('PORT from env:', process.env.PORT);
  app.listen(process.env.PORT, '127.0.0.1', () => {
    console.log('API running on port', process.env.PORT);
  });

```

Команда `require('dotenv').config()` підключає пакет `dotenv` і викликає його метод `config()`, який зчитує змінні з файла `.env` та додає їх до об'єкта `process.env`. Завдяки цьому конфіденційні параметри підключення до бази даних і номер порту не записуються безпосередньо в програмний код.

Вирази `const express = require('express')`, `const sql = require('mssql')`, `const helmet = require('helmet')` і `const cors = require('cors')` завантажують відповідно фреймворк Express, бібліотеку взаємодії з Microsoft SQL Server, засіб формування захисних HTTP-заголовків і механізм налаштування міжджерельних запитів. Конструкція `const { rateLimit, ipKeyGenerator } = require('express-rate-limit')` за допомогою деструктуризації отримує з пакета `express-rate-limit` функцію створення обмежувача частоти запитів і допоміжну функцію нормалізації IP-адрес.

Команда `const app = express()` створює екземпляр Express-застосунку, через який надалі реєструються проміжні функції, маршрути та обробники помилок. Вираз `app.set('trust proxy', 'loopback')` указує Express довіряти проксі-заголовкам лише тоді, коли запит надходить через локальний `loopback`-інтерфейс. Це необхідно, оскільки IIS передає запити до Node.js через адресу `127.0.0.1`. Команда `app.disable('x-powered-by')` вимикає автоматичне додавання заголовка `X-Powered-By`: Express, щоб не розкривати клієнту зайву інформацію про використану серверну технологію.

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		58

Масив `allowedOrigins` містить дозволені джерела браузерних запитів. У поточній конфігурації дозволено адресу `http://localhost:8080`, за якою відкривається клієнтська частина з Docker-контейнера. Ключове слово `const` забороняє повторне присвоєння іншого масиву цій змінній.

Команда `app.use(cors({ ... }))` підключає CORS як проміжну функцію до всіх запитів. Властивість `origin` містить функцію з параметрами `origin` і `callback`. Параметр `origin` зберігає адресу джерела запиту, а `callback` використовується для передавання результату перевірки модулю CORS. Умова `!origin || allowedOrigins.includes(origin)` дозволяє запити без заголовка `Origin`, наприклад від `curl`, а також запити, адреса яких входить до масиву `allowedOrigins`. Метод `includes()` перевіряє наявність значення в масиві. Команда `callback(null, true)` повідомляє, що помилки немає і джерело дозволене. Якщо адреса не дозволена, конструкція `new Error('CORS_ORIGIN_NOT_ALLOWED')` створює об'єкт помилки, який передається через `callback`. Властивість `methods` дозволяє методи `GET`, `POST` і `OPTIONS`, а `allowedHeaders` дозволяє передавання заголовка `Content-Type`.

Команда `app.use(express.json())` підключає вбудований парсер JSON. Він зчитує тіло запиту з типом `application/json`, перетворює його на JavaScript-об'єкт і записує результат у `req.body`. Команда `app.use(helmet())` підключає `Helmet`, який додає до HTTP-відповідей набір захисних заголовків, зокрема для зменшення ризику `MIME-sniffing`, вбудовування сторінки у фрейм та інших небезпечних дій браузера.

Команда `app.use(rateLimit({ ... }))` підключає обмежувач частоти до всіх маршрутів, оголошених після неї. Властивість `windowMs: 15 * 60 * 1000` визначає проміжок тривалістю 15 хвилин у мілісекундах. Властивість `max: 100` встановлює граничну кількість у 100 запитів від одного клієнта протягом цього проміжку. Параметр `standardHeaders: true` додає стандартні заголовки з інформацією про ліміт, а `legacyHeaders: false` вимикає старі заголовки виду `X-RateLimit-*`.

Властивість `keyGenerator` задає власну функцію формування ключа, за яким рахуються запити. Вираз `req.ip || req.socket.remoteAddress || 'unknown'` спочатку намагається використати IP-адресу, визначену `Express`, потім адресу TCP-

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		59

з'єднання, а за відсутності обох значень повертає рядок unknown. Оператор || вибирає перше істинне значення. Метод `ip.match()` перевіряє адресу за регулярним виразом. Вираз для IPv4 визначає формат на зразок 192.168.56.1:59138, де після адреси присутній змінний порт. Якщо збіг знайдено, конструкція `ipv4WithPort[1]` отримує тільки IP-частину без порту. Другий регулярний вираз аналогічно обробляє IPv6-адресу, записану у квадратних дужках із портом. Команда `ipKeyGenerator(ip)` формує нормалізований ключ для IPv4 або IPv6, щоб усі запити одного клієнта враховувалися разом. Властивість `message` задає текст відповіді, яка повертається після перевищення ліміту.

Об'єкт `dbConfig` містить параметри підключення до SQL Server. Властивості `user`, `password`, `server` і `database` отримують значення зі змінних середовища `process.env.DB_USER`, `process.env.DB_PASSWORD`, `process.env.DB_SERVER` і `process.env.DB_DATABASE`. Вкладений об'єкт `options` містить параметр `trustServerCertificate: true`, який дозволяє використовувати в лабораторному середовищі сертифікат SQL Server без перевірки його довіри центром сертифікації.

Команда `app.get('/api/messages', async (req, res) => { ... })` реєструє маршрут для обробки GET-запитів. Параметр `req` представляє вхідний запит, а `res` - майбутню відповідь. Ключове слово `async` дозволяє застосовувати `await`. У блоці `try` команда `await sql.connect(dbConfig)` установлює з'єднання з SQL Server за параметрами `dbConfig`, очікуючи завершення операції. Отримане з'єднання зберігається в константі `pool`.

Вираз `pool.request()` створює новий об'єкт SQL-запиту, а метод `query()` виконує команду `SELECT Username, MessageText FROM MessagesTable`. Команда `SELECT` вибирає зі всіх рядків таблиці `MessagesTable` значення стовпців `Username` і `MessageText`. Результат зберігається в константі `result`. Команда `res.json(result.recordset)` перетворює отриманий набір записів на JSON і надсилає його клієнту. Ключове слово `return` завершує виконання обробника після формування відповіді. Якщо під час підключення або вибірки виникає помилка, блок `catch` отримує її у змінній `err`, команда `console.error(err)` виводить технічні

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				60
Змн.	Арк.	№ докум.	Підпис	Дата		

відомості в консоль, а `res.status(500).send('Server error')` повертає клієнту код 500 і текстове повідомлення.

Команда `app.post('/api/messages', async (req, res) => { ... })` реєструє обробник POST-запитів для додавання повідомлень. Вираз `console.log('POST WORKS:', req.body)` виводить у консоль тіло отриманого запиту. Для формування змінної `username` перевіряється, чи є `req.body?.Username` рядком. Оператор `?.` безпечно звертається до властивості навіть тоді, коли `body` не визначено. Оператор `typeof` повертає тип значення, а тернарна конструкція умова `? значення1 : значення2` вибирає результат залежно від істинності умови. Якщо поле є рядком, метод `trim()` видаляє пробіли на початку та в кінці; інакше використовується порожній рядок. Змінна `messageText` формується аналогічно з поля `MessageText`.

Умова `username.length < 1 || username.length > 50` перевіряє, чи не є ім'я порожнім і чи не перевищує воно 50 символів. Властивість `length` повертає кількість символів, а оператор `||` означає логічне «або». Якщо умова виконується, команда `res.status(400).send(...)` повертає код 400 Bad Request і припиняє виконання маршруту. Наступна умова аналогічно перевіряє, що текст повідомлення містить від 1 до 100 символів.

Після успішної валідації команда `await sql.connect(dbConfig)` отримує підключення до SQL Server. Метод `pool.request()` створює SQL-запит. Виклик `.input('Username', sql.NVarChar(50), username)` створює параметр `Username`, задає для нього тип `NVARCHAR(50)` і передає перевірене значення змінної `username`. Виклик `.input('MessageText', sql.NVarChar(100), messageText)` аналогічно створює параметр для тексту повідомлення. Метод `.query()` виконує параметризовану команду `INSERT INTO MessagesTable`, а значення `@Username` і `@MessageText` передаються окремо від тексту SQL-команди. Такий спосіб запобігає інтерпретації введених даних як частини SQL-запиту. Після успішного додавання `res.status(200).send('Inserted')` повертає код 200 і повідомлення `Inserted`. У разі помилки блок `catch` виводить її в консоль і повертає код 500 із текстом `DB error`.

Команда `app.use((err, req, res, next) => { ... })` реєструє централізований обробник помилок Express. Наявність чотирьох параметрів дає Express змогу

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		61

розпізнати цю функцію саме як обробник помилок. Умова `err?.message === 'CORS_ORIGIN_NOT_ALLOWED'` безпечно перевіряє текст помилки. Якщо вона спричинена недозволенним джерелом, сервер повертає код 403 Forbidden. Оператор `instanceof SyntaxError` перевіряє, чи належить помилка до типу `SyntaxError`, властивість `err.status === 400` підтверджує помилку запиту, а вираз `'body' in err` перевіряє наявність властивості `body`. У такому разі сервер повертає код 400 із повідомленням `Invalid JSON`. Інші помилки виводяться через `console.error(err)` і завершуються відповіддю 500 Server error. Параметр `next` у цій реалізації не викликається, оскільки кожна помилка обробляється безпосередньо цією функцією.

Команда `console.log('PORT from env:', process.env.PORT)` виводить у консоль номер порту, отриманий із файлу `.env`. Виклик `app.listen(process.env.PORT, '127.0.0.1', callback)` запускає HTTP-сервер на заданому порту та прив'язує його лише до локальної адреси 127.0.0.1. Через це Node.js API доступний локальному IIS, але не приймає прямі мережеві з'єднання на адресі вебвузла. Після успішного запуску виконується функція зворотного виклику, яка командою `console.log()` повідомляє про початок роботи API.

3.2 Налаштування мережевої взаємодії та механізмів захисту

Після встановлення компонентів виконано налаштування бази даних, службового доступу, мережевих портів і маршрутизації запитів. Метою цього етапу є побудова контрольованого ланцюга: браузер звертається до IIS, IIS передає дозволений маршрут локальному Node.js API, а API встановлює з'єднання з SQL Server від імені окремого службового облікового запису.

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		62

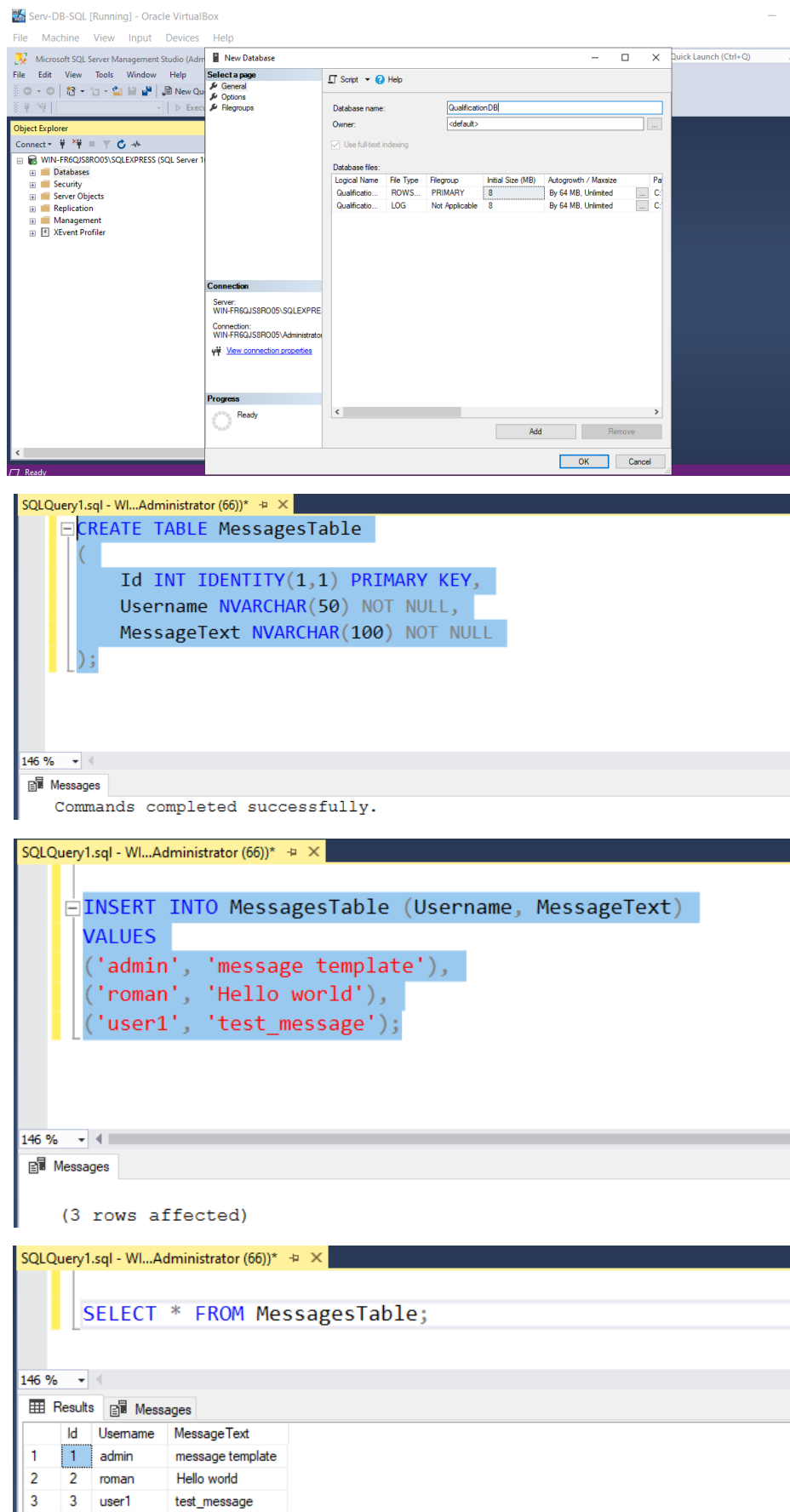


Рис. 3.21-3.24 – Створення бази QualificationDB і таблиці MessagesTable

У SQL Server створено базу даних QualificationDB та таблицю MessagesTable. Таблиця містить автоматично сформований ідентифікатор, відображуване ім'я автора й текст повідомлення. Після створення до таблиці додано тестові записи та виконано вибірку, що підтвердило правильність її структури.

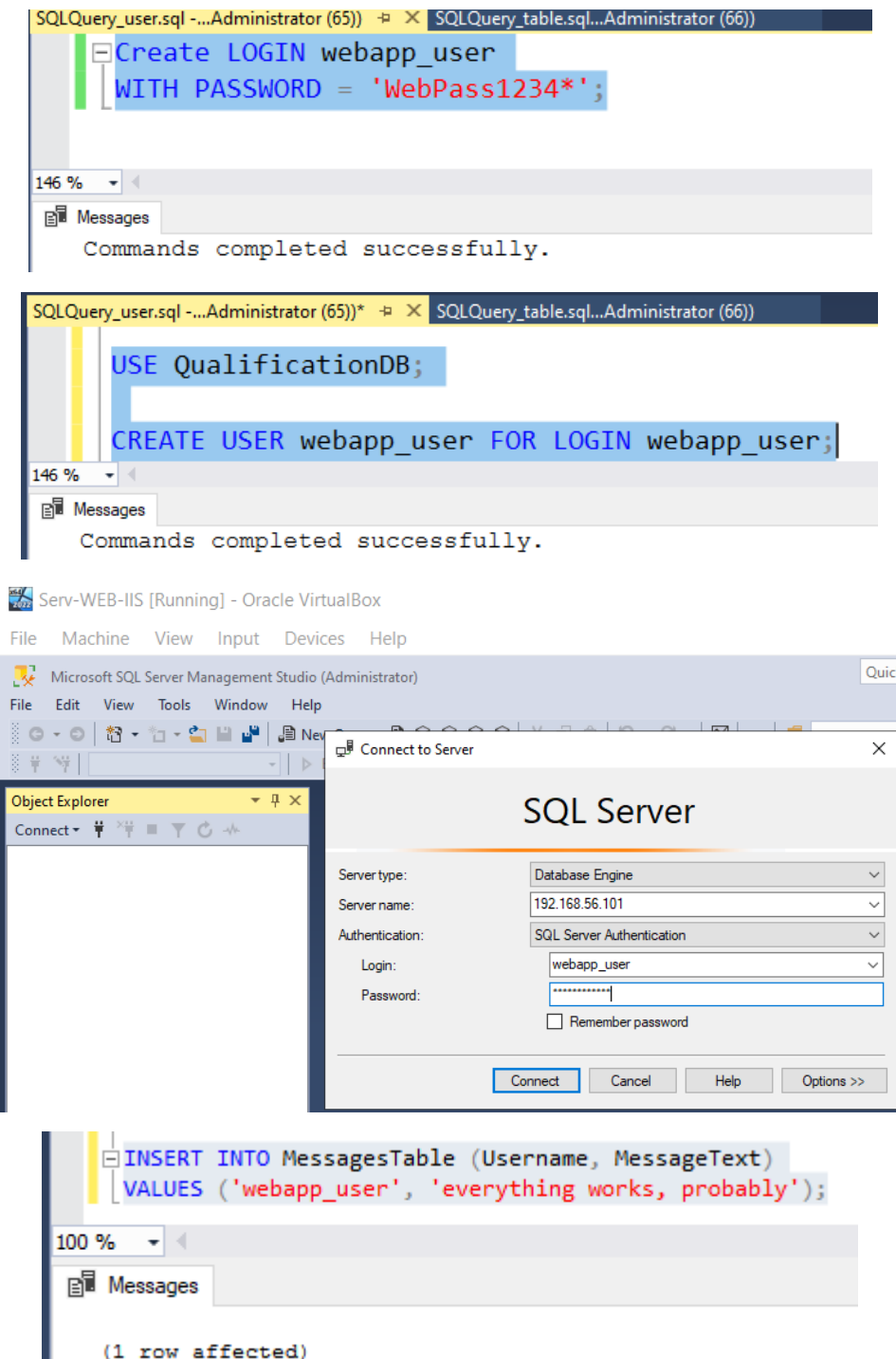


Рис. 3.25-3.28 – Створення службового логіна webapp_user та перевірка його підключення

Для Backend API створено окремий логін webapp_user і відповідного користувача в базі QualificationDB. Серверний застосунок підключається до SQL

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				64
Змн.	Арк.	№ докум.	Підпис	Дата		

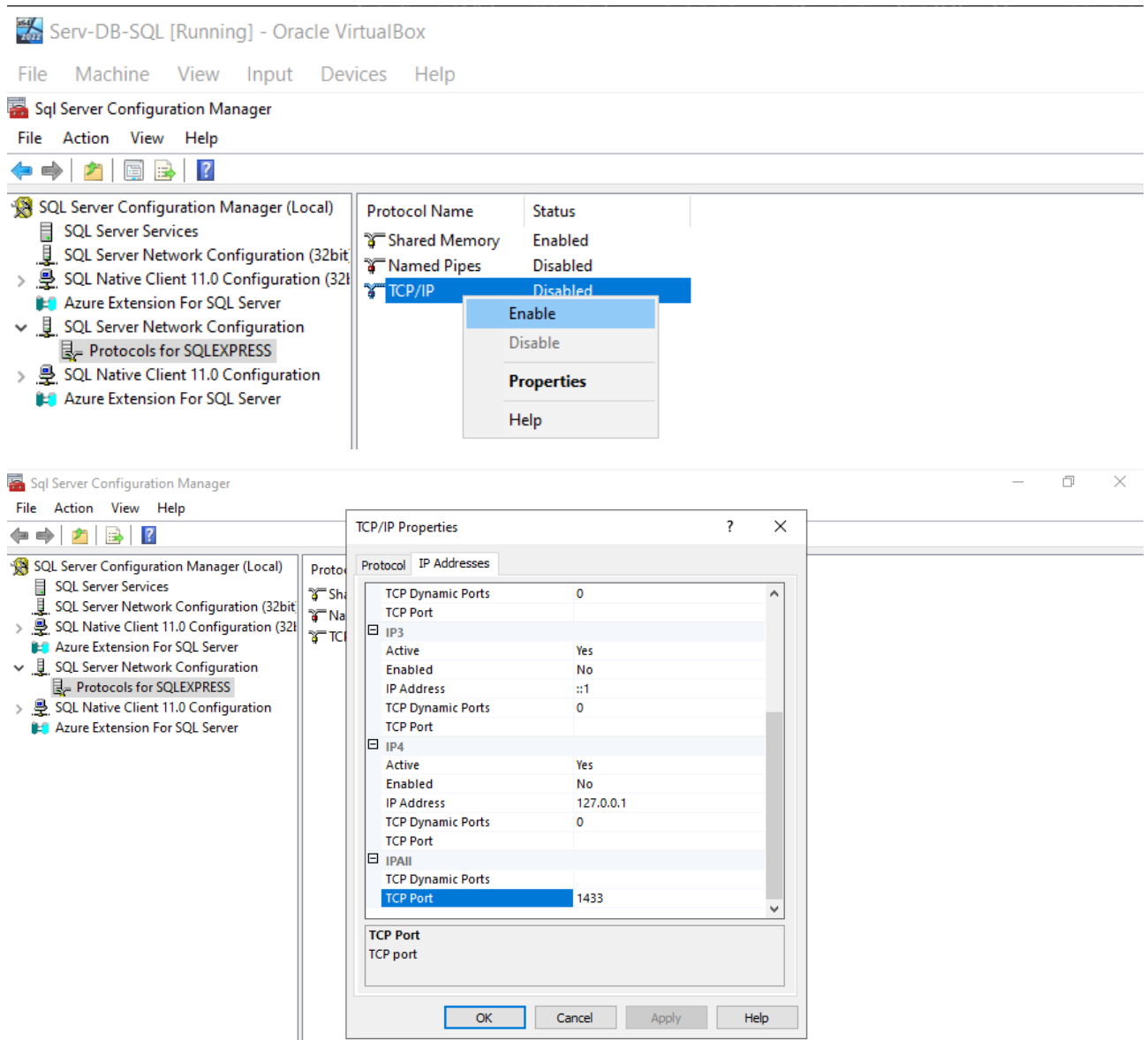
Server саме від цього імені, а не під адміністративним логіном sa. Також показано перевірку підключення та виконання тестового додавання запису від імені службового користувача.

```
SQLQuery_user.sql -...Administrator (69))* X SQLQuery_table.sql...Administrator (60))*

GRANT SELECT ON OBJECT::dbo.MessagesTable TO webapp_user;
GRANT INSERT ON OBJECT::dbo.MessagesTable TO webapp_user;
```

Рис. 3.29 – Налаштування точкових дозволів webapp_user

У фінальній конфігурації службовому користувачу потрібно залишити лише ті дозволи, які використовує API: SELECT і INSERT для таблиці dbo.MessagesTable.



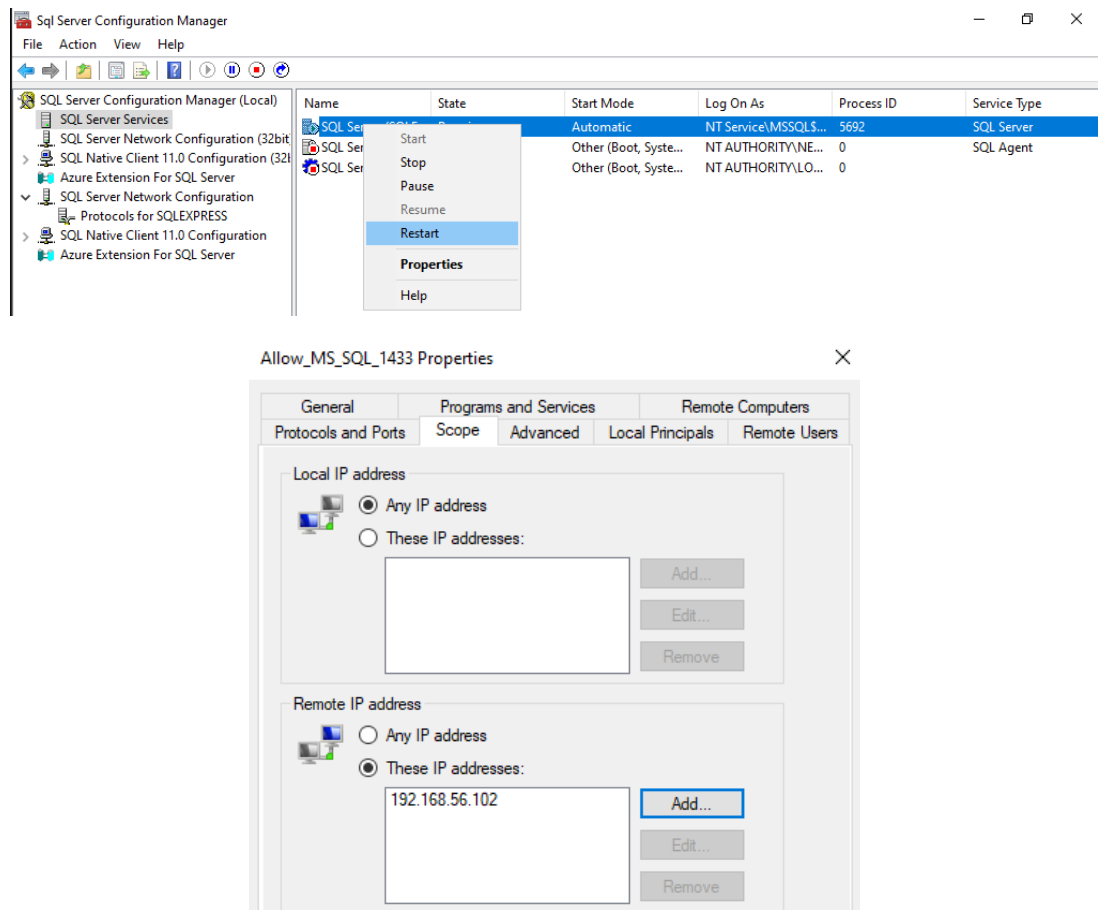


Рис. 3.30-3.33 – Налаштування TCP/IP, порту 1433 і правила брандмауера SQL Server

У SQL Server Configuration Manager увімкнено протокол TCP/IP, для екземпляра задано статичний порт 1433 і перезапущено службу SQL Server. У Windows Defender Firewall створено вхідне правило для цього порту. Область правила обмежено адресою вебвузла 192.168.56.102, тому підключення до бази має надходити лише від Backend API, розміщеного на Serv-WEB-IIS.

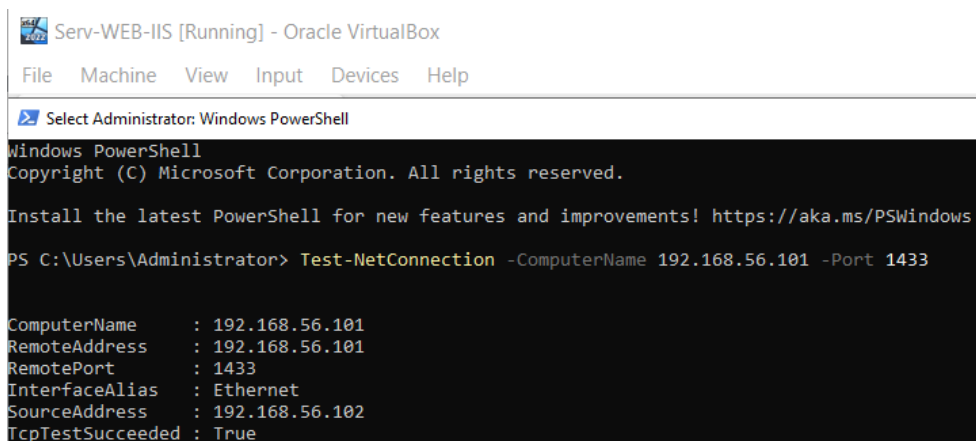


Рис. 3.34 – Перевірка доступності порту 1433 з вузла Serv-WEB-IIS

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				66
Змн.	Арк.	№ докум.	Підпис	Дата		

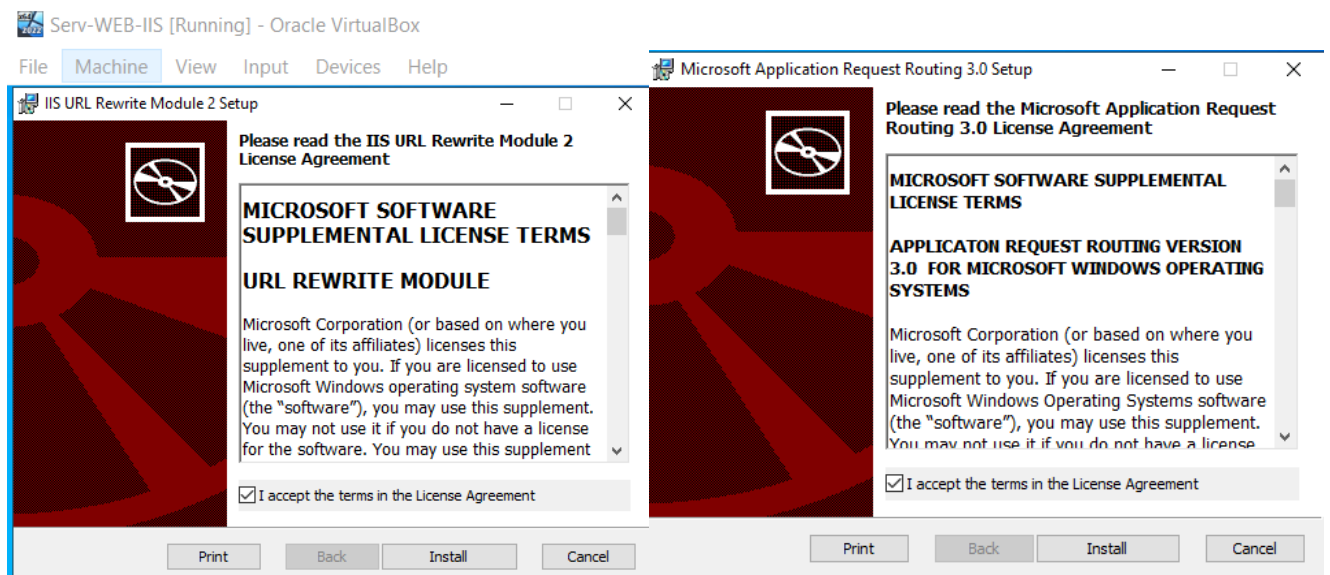
Командою Test-NetConnection на вузлі Serv-WEB-IIS перевірено доступність SQL Server за адресою 192.168.56.101 і портом 1433. Значення TcpTestSucceeded: True підтверджує, що дозволений зв'язок між прикладним і інформаційним рівнями працює. Саме цей напрямок потрібний для виконання SELECT та INSERT із Node.js API.

```
PS C:\Users\zarut> Test-NetConnection -ComputerName 192.168.56.101 -Port 1433
WARNING: TCP connect to (192.168.56.101 : 1433) failed
WARNING: Ping to 192.168.56.101 failed with status: TimedOut

ComputerName           : 192.168.56.101
RemoteAddress          : 192.168.56.101
RemotePort             : 1433
InterfaceAlias         : Ethernet 2
SourceAddress          : 192.168.56.1
PingSucceeded          : False
PingReplyDetails (RTT) : 0 ms
TcpTestSucceeded       : False
```

Рис. 3.35 – Перевірка заборони прямого доступу клієнтського вузла до SQL Server

Після звуження області правила брандмауера аналогічна перевірка з основного вузла повинна завершитися невдало. Такий результат підтверджує, що браузер і Docker-контейнер не можуть підключатися до SQL Server напряму, а взаємодія з даними відбувається лише через прикладний сервер.



```
PS C:\Users\Administrator> iisreset

Attempting stop...
Internet services successfully stopped
Attempting start...
Internet services successfully restarted
PS C:\Users\Administrator> _
```

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				67
Змн.	Арк.	№ докум.	Підпис	Дата		

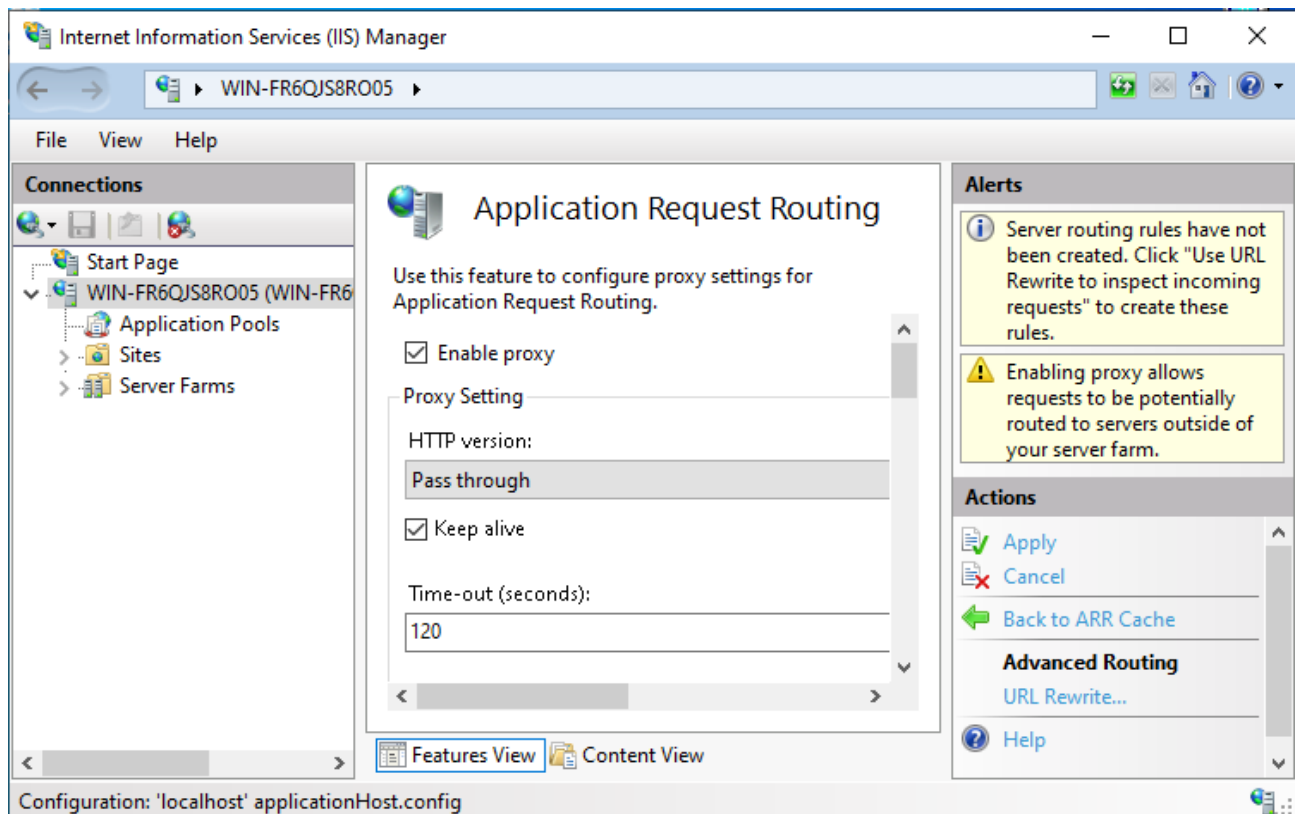
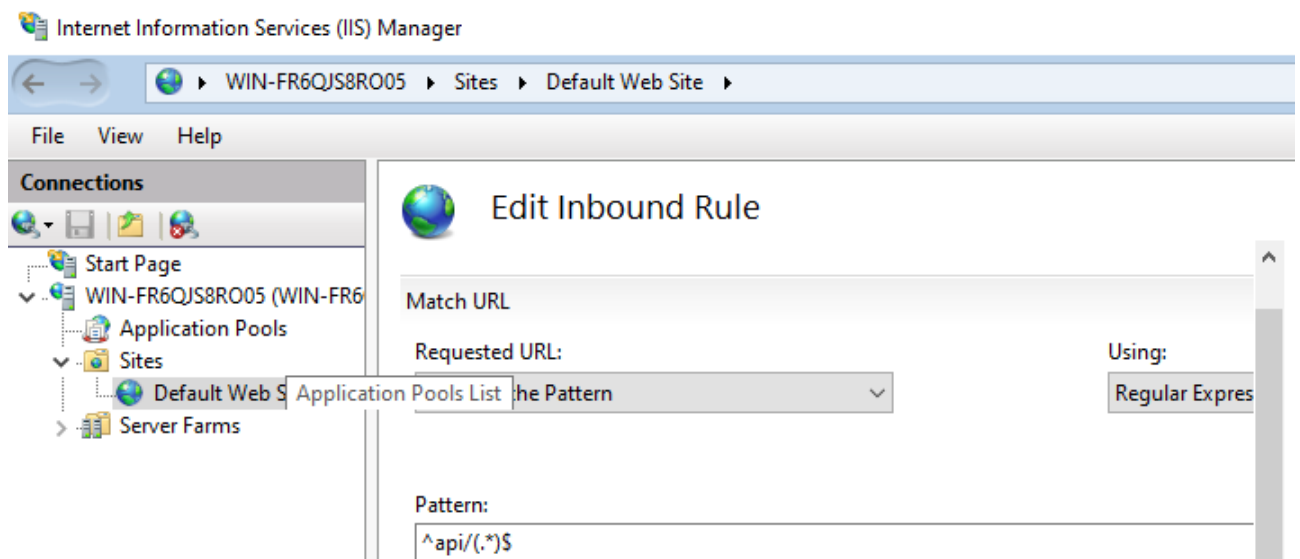


Рис. 3.36-3.39 – Встановлення URL Rewrite та Application Request Routing

Для реалізації реверс-проксі на IIS встановлено модулі URL Rewrite і Application Request Routing. Після встановлення служб IIS виконано перезапуск, а в налаштуваннях ARR увімкнено режим Proxy. Це дозволило IIS приймати зовнішні запити та передавати їх внутрішньому Node.js-процесу.



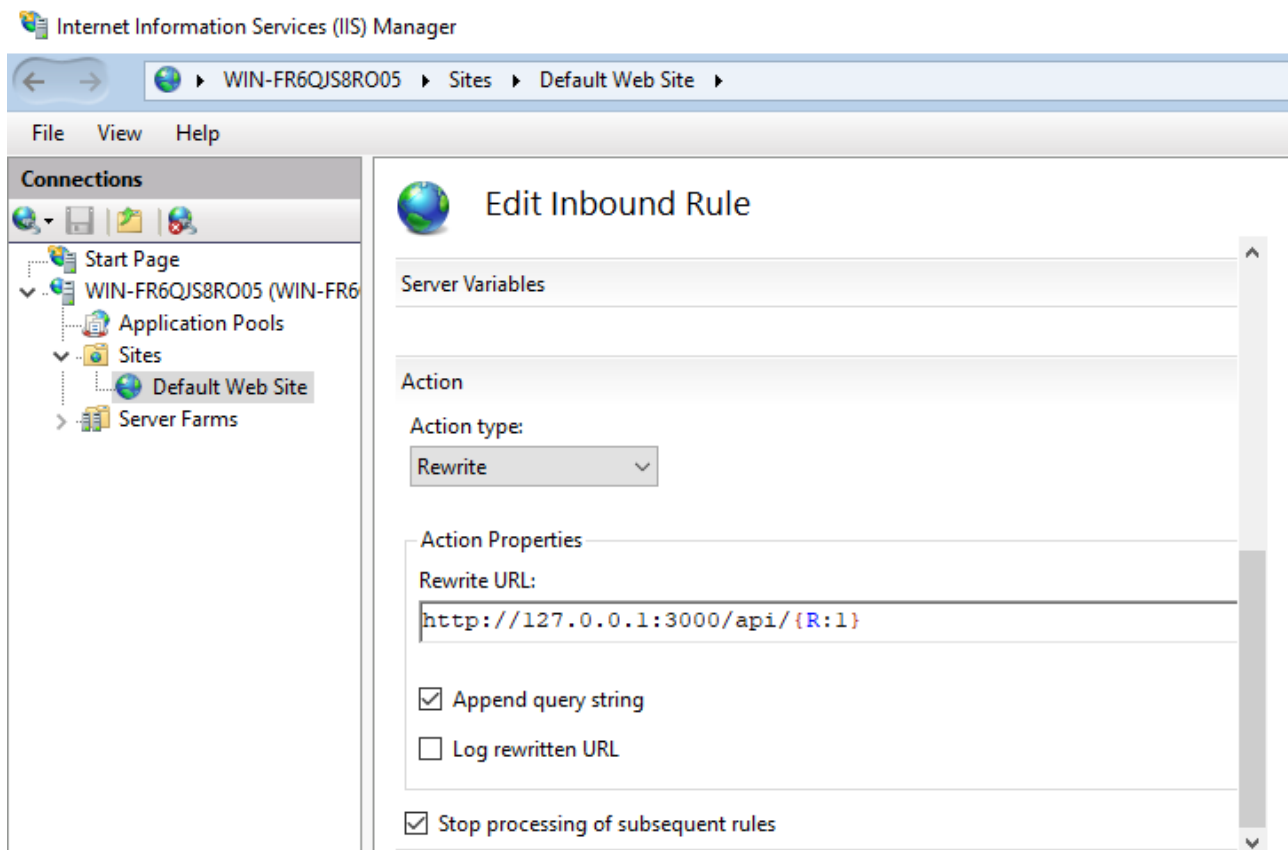
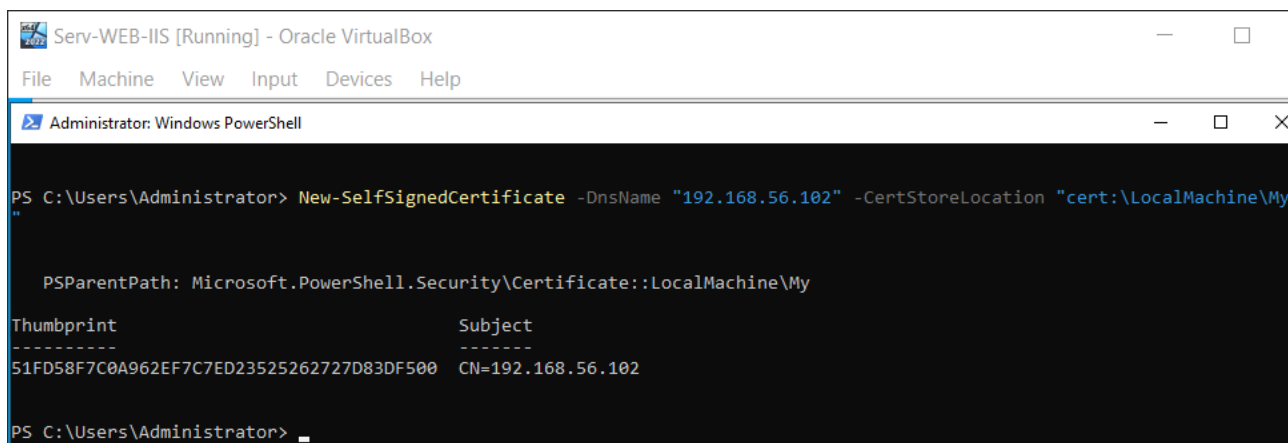


Рис. 3.40-3.41 – Налаштування правила передавання API-запитів до Node.js

На сайті IIS створено правило, яке обробляє зовнішній шлях /api/ і передає відповідні запити на локальну адресу Node.js.

У правилі URL Rewrite шаблон ^api/(.*)\$ означає: початок адреси має містити api/, а вираз (.*) зберігає решту шляху в групі підстановки. Адреса `http://127.0.0.1:3000/api/{R:1}` передає збережену частину маршруту локальному Node.js. Позначення {R:1} підставляє значення першої групи регулярного виразу, тому /api/messages зберігає кінцевий сегмент messages.



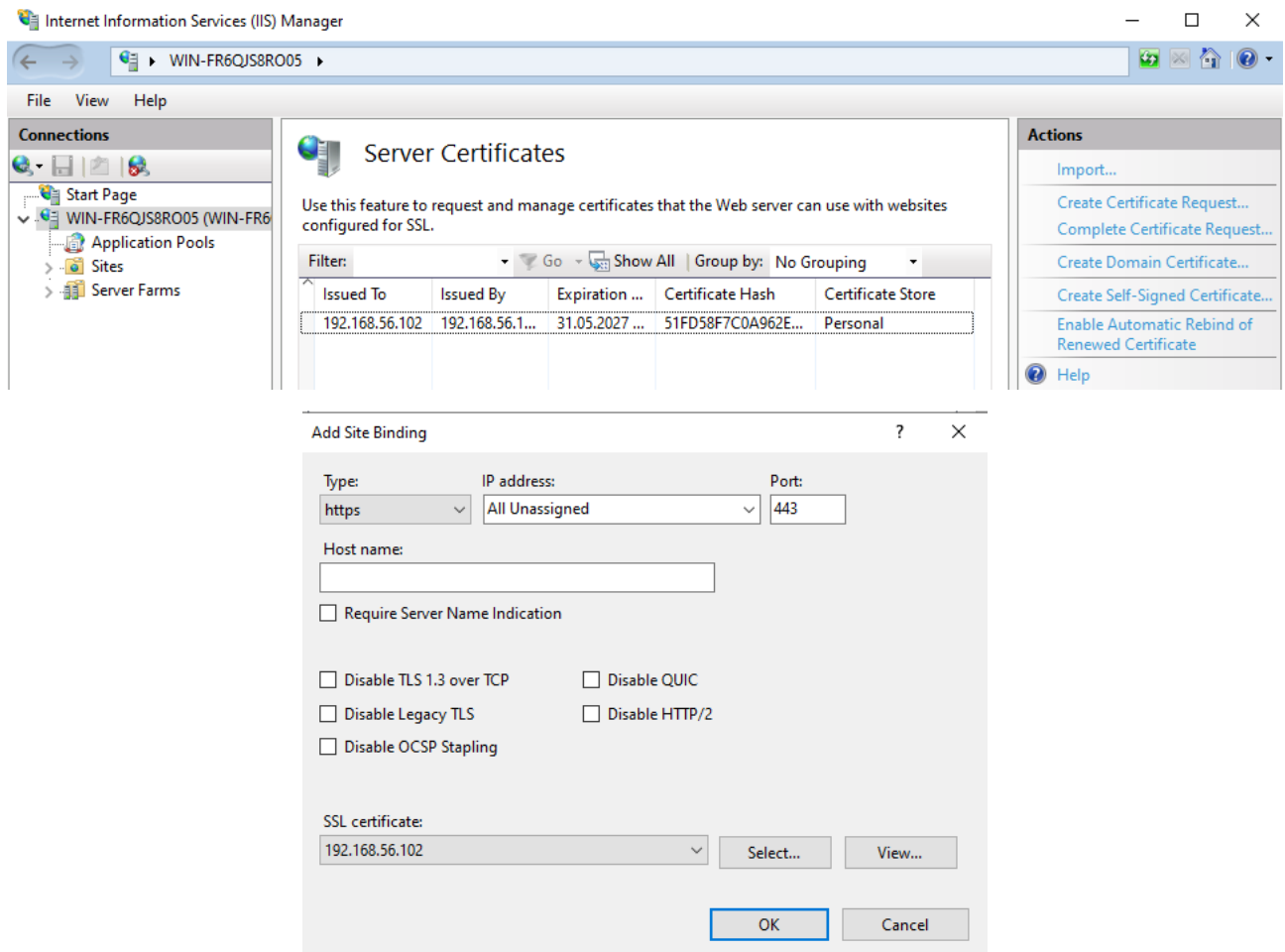
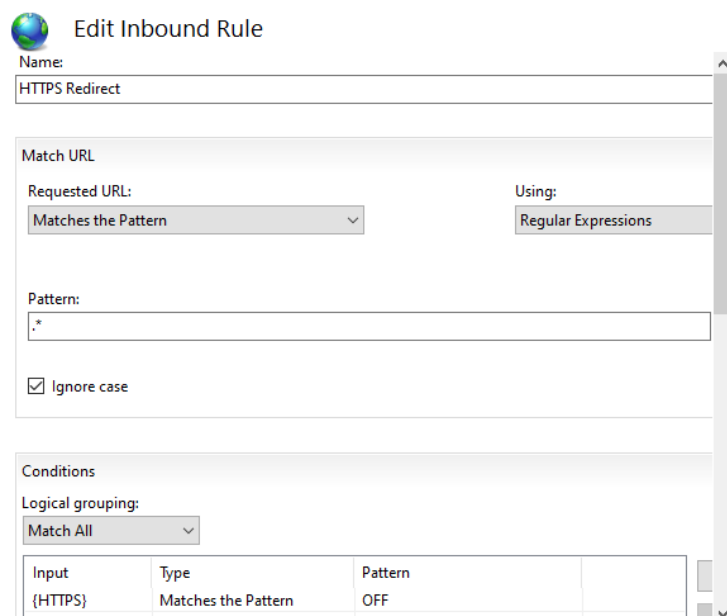


Рис. 3.42-3.44 – Створення самопідписаного сертифіката та прив'язки HTTPS

На вебвузлі створено самопідписаний сертифікат для адреси 192.168.56.102. Сертифікат прив'язано до сайту IIS на порту 443. Браузер може попереджати про недовірений центр сертифікації, однак після прийняття сертифіката передавання даних між браузером та IIS відбувається через HTTPS.



		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				70
Змн.	Арк.	№ докум.	Підпис	Дата		

Add Condition ? X

Condition input:

Check if input string:

Pattern:

☒ Ignore case

Edit Inbound Rule

Conditions

Server Variables

Action
 Action type:

Action Properties
 Redirect URL:

☒ Append query string

Redirect type:

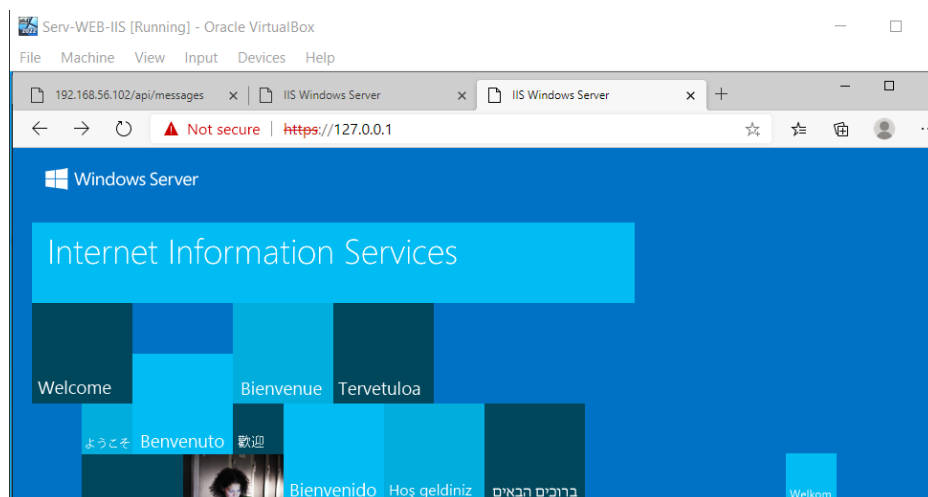


Рис. 3.45-3.48 – Налаштування перенаправлення HTTP-запитів на HTTPS

Окреме правило URL Rewrite виконує постійне перенаправлення з HTTP на HTTPS, якщо запит надійшов незахищеним каналом. Умова правила перевіряє значення HTTPS, а адреса перенаправлення зберігає початковий шлях запиту. Це

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				71
Змн.	Арк.	№ докум.	Підпис	Дата		

забезпечує однаковий захищений канал як для відкриття сторінок IIS, так і для звернень до API.

Умова {HTTPS} matches OFF спрацьовує лише для незахищеного HTTP-запиту. Дія типу Redirect повертає клієнту адресу з протоколом HTTPS, а підстановка {HTTP_HOST} зберігає ім'я або IP вебвузла. Позначення {R:1} переносить початковий шлях, тому запит до /api/messages після перенаправлення не перетворюється на головну сторінку.

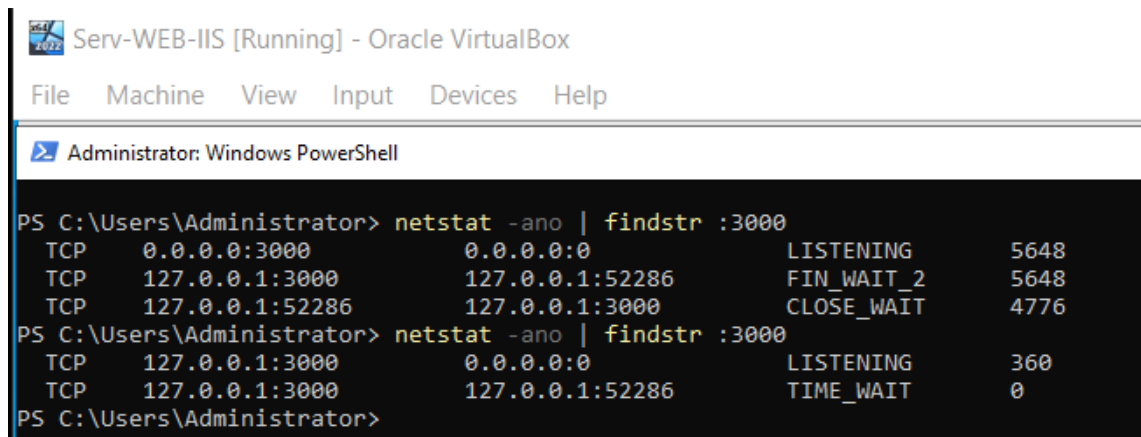


Рис. 3.49 – Перевірка локальної прив'язки Node.js API

Виведення netstat показує, що прикладний сервер слухає порт 3000 лише на адресі 127.0.0.1. Отже, процес доступний IIS на тому самому вузлі, але не приймає прямі з'єднання на мережевій адресі 192.168.56.102. Така прив'язка не дозволяє клієнту обійти реверс-проксі та звернутися до API напряму.

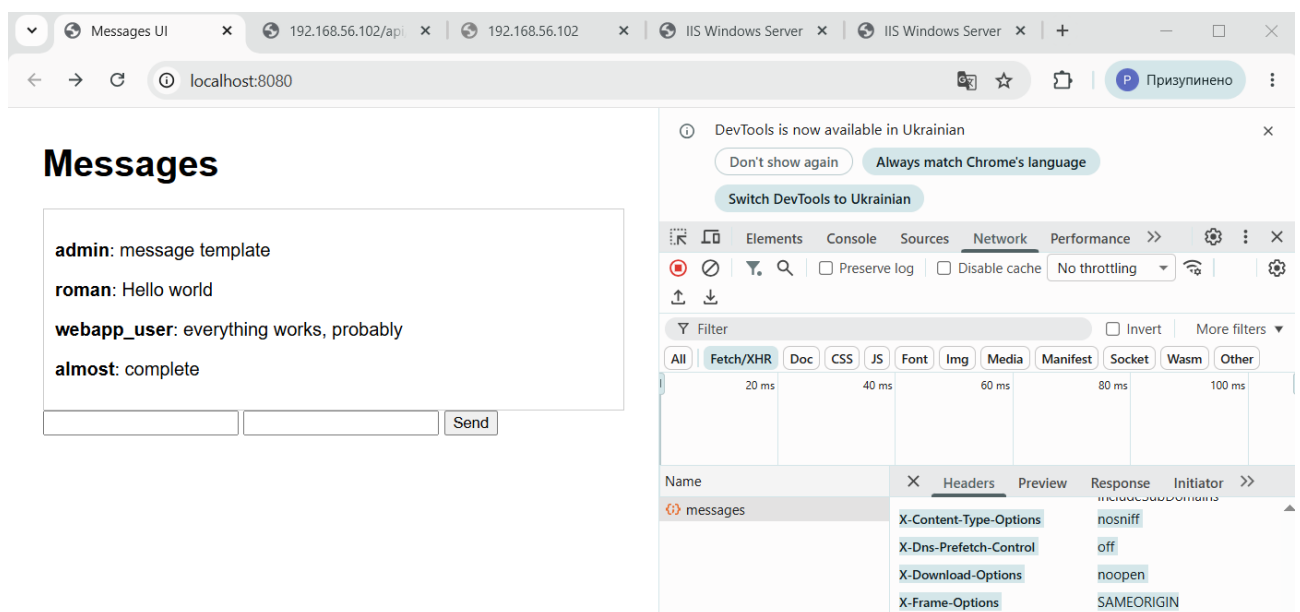


Рис. 3.50 – Робота вебінтерфейсу та захисні HTTP-заголовки відповіді

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				72
Змн.	Арк.	№ докум.	Підпис	Дата		

Після завершення маршрутизації вебінтерфейс отримує повідомлення через зовнішню адресу IIS. У засобах розробника браузера видно заголовки, додані Helmet, зокрема X-Content-Type-Options і X-Frame-Options.

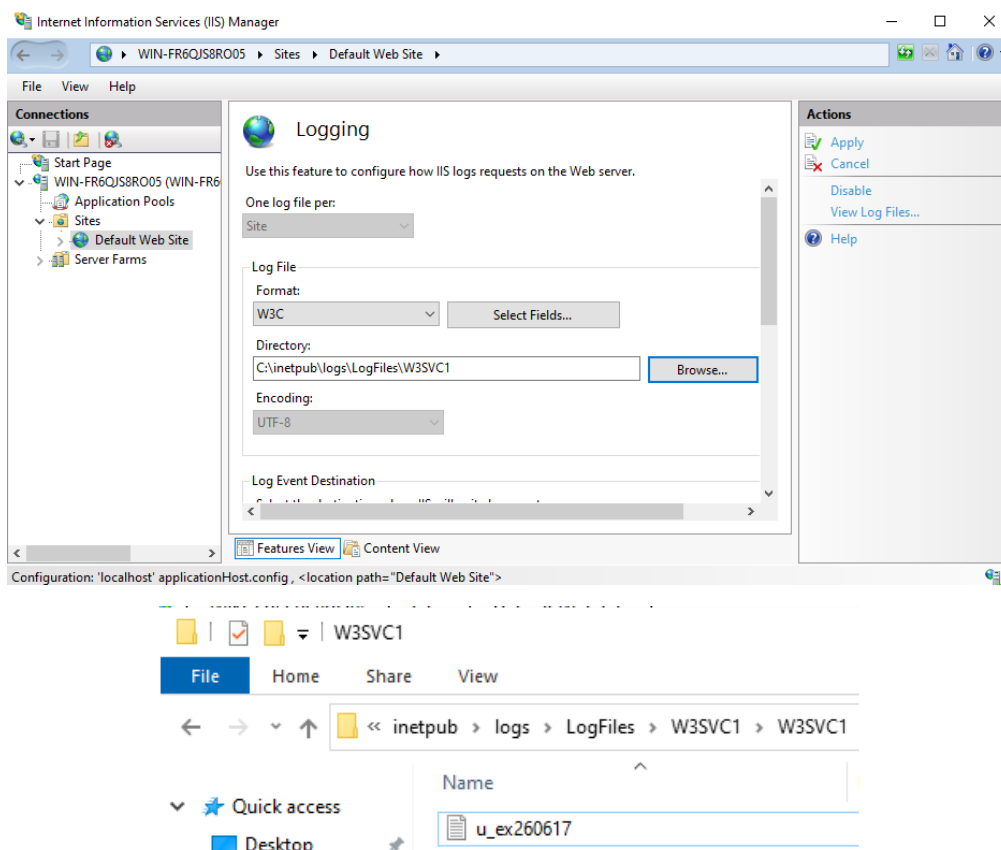


Рис. 3.51-3.52 – Налаштування журналювання запитів IIS

Для завершення практичної частини на IIS потрібно ввімкнути стандартне журналювання у форматі W3C та зафіксувати каталог збереження файлів. У журналах доцільно залишити дату, час, метод, URI, код стану, адресу клієнта та час обробки.

3.3 Перевірка стійкості до типових атак

Під час тестування досліджено реакцію системи на некоректні вхідні дані, спроби SQL-ін'єкції та XSS, звернення до неіснуючих маршрутів, запити з недозволених джерел, перевищення встановленої частоти звернень і спроби прямого доступу до внутрішніх служб.

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		73

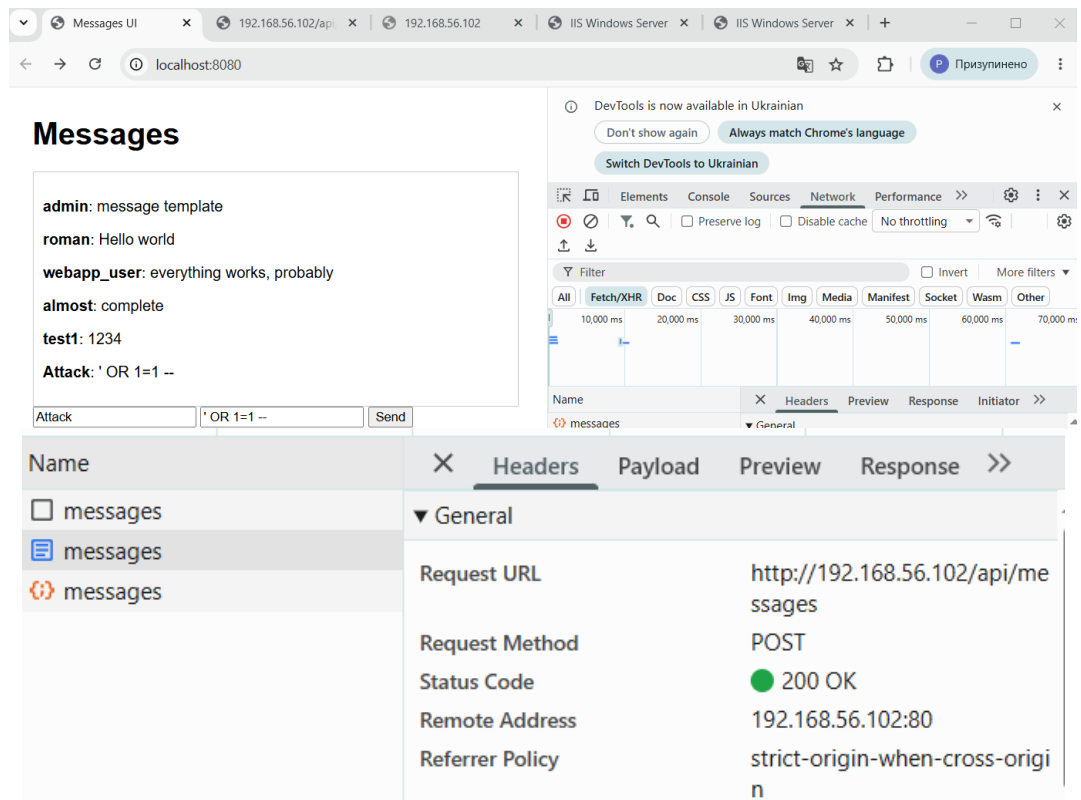


Рис. 3.53-3.54 – Перевірка спроби SQL-ін'єкції через поле повідомлення

До поля повідомлення введено рядок, характерний для спроби SQL-ін'єкції. Запит завершився звичайною відповіддю API, а введений текст був збережений як дані, а не виконаний як частина SQL-команди. Такий результат підтверджує використання параметризованого INSERT і відсутність прямого об'єднання користувацького вводу з текстом запиту.

```
PS C:\Users\zarut> Test-NetConnection -ComputerName 192.168.56.102 -Port 3000
WARNING: TCP connect to (192.168.56.102 : 3000) failed
WARNING: Ping to 192.168.56.102 failed with status: TimedOut

ComputerName           : 192.168.56.102
RemoteAddress          : 192.168.56.102
RemotePort             : 3000
InterfaceAlias         : Ethernet 2
SourceAddress          : 192.168.56.1
PingSucceeded          : False
PingReplyDetails (RTT) : 0 ms
TcpTestSucceeded       : False
```

Рис. 3.55 – Перевірка неможливості прямого доступу до порту Node.js

З основного вузла необхідно перевірити порт 3000 мережевої адреси Serv- WEB-IIS. Очікується негативний результат, оскільки Node.js слухає лише loopback-

інтерфейс. Успішне блокування підтвердить, що зовнішні клієнти можуть взаємодіяти з API тільки через ІІС.

```

PS C:\Users\zarut> 1..120 | ForEach-Object {
>>     $result = curl.exe -k -s -o NUL -w "%{http_code}" `
>>         "https://192.168.56.102/api/messages"
>>     Write-Host "$_ : $result"
>> }
1 : 200
2 : 200
3 : 200
4 : 200
5 : 200
6 : 200
7 : 200
8 : 200
9 : 200
99 : 200
100 : 429
101 : 429
102 : 429
103 : 429
104 : 429
105 : 429
106 : 429
107 : 429
108 : 429
109 : 429
110 : 429
111 : 429
112 : 429
113 : 429
114 : 429
115 : 429
116 : 429
117 : 429
118 : 429
119 : 429
120 : 429
PS C:\Users\zarut>
  
```

Рис. 3.56-3.57 – Перевірка обмеження частоти запитів

Для перевірки express-rate-limit потрібно надіслати серію повторних запитів у межах установленого проміжку часу. Після перевищення ліміту API має повернути код 429 Too Many Requests. Це демонструє, що один клієнт не може необмежено створювати навантаження на прикладний сервер.

```

PS C:\Users\zarut> curl.exe -k -i `
>> -H "Origin: http://example-attacker.local" `
>> https://192.168.56.102/api/messages
HTTP/1.1 403 Forbidden
Keep-Alive: timeout=5
Content-Type: text/html; charset=utf-8
ETag: W/"1a-8gebz6ffB0MINj4rN80gMm+oGWY"
X-Powered-By: ARR/3.0
Date: Wed, 17 Jun 2026 16:52:20 GMT
Content-Length: 26

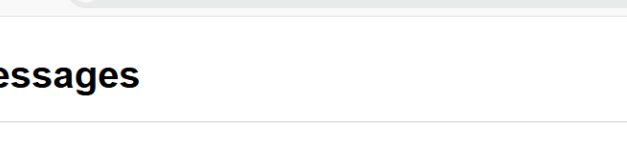
CORS origin is not allowed
PS C:\Users\zarut>
  
```

Рис. 3.58 – Перевірка політики CORS

Запит із джерела, яке не входить до дозволеного списку, повинен бути заблокований політикою CORS у браузері. Для скріншота доцільно використати

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				75
Змн.	Арк.	№ докум.	Підпис	Дата		

консоль або вкладку Network із повідомленням про відсутність дозволу для заданого Origin. Ця перевірка стосується саме браузерної взаємодії й не замінює серверних обмежень доступу.



← → ↻ ⓘ localhost:8080

Messages

admin: message template

roman: Hello world

TestUser: TEST

Arthur: Morgan

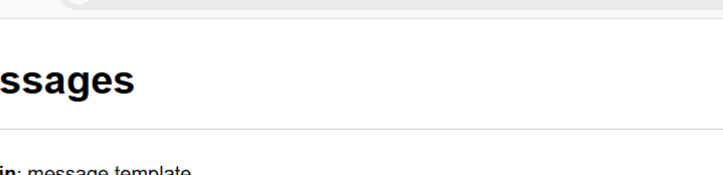
google: com

aaaaaaaaaaaaaaaaaaaaa

[illegible]

Рис. 3.59-3.60 – Перевірка відхилення некоректних даних

Порожнє повідомлення або значення, що перевищує встановлену довжину, не повинно передаватися до SQL Server. Backend API отримує запит, але не передає його далі до бази даних.



← → ↻ ⓘ localhost:8080

Messages

admin: message template

roman: Hello world

TestUser: TEST

Arthur: Morgan

google: com

hacker:

hacker

Рис. 3.61 – Перевірка відображення потенційно небезпечного коду

Для перевірки клієнтського відображення до повідомлення вводиться рядок із HTML- або JavaScript-кодом, але він не спрацьовує і не відображається.

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				76
Змн.	Арк.	№ докум.	Підпис	Дата		

Висновки до розділу

У межах практичної реалізації розгорнуто клієнтський компонент у Docker, вебвузол з IIS і Node.js/Express та окремий сервер SQL Server 2022 Express. Налаштовано ізольовану мережу Host-Only, створено базу QualificationDB, службовий обліковий запис застосунку, TCP-доступ до SQL Server лише з вебвузла, реверс-проксі IIS, самопідписаний сертифікат і перенаправлення на HTTPS.

Працездатність системи підтверджено завантаженням вебінтерфейсу, виконанням GET- і POST-запитів через IIS та підключенням Backend API до бази даних. Проведені контрольні перевірки показали, що параметризовані SQL-запити не допускають виконання введених даних як частини SQL-команди, серверна валідація відхиляє порожні та надмірно довгі значення, політика CORS обмежує браузерні запити з недозволених джерел, а механізм express-rate-limit повертає код 429 після перевищення встановленої кількості звернень. Локальна прив'язка Node.js і правила брандмауера не дозволяють клієнтському вузлу напрямку звертатися до Backend API та SQL Server. Отже, реалізована інфраструктура забезпечує багаторівневий захист, у якому мережеві, серверні та прикладні механізми доповнюють один одного.

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		77

ВИСНОВКИ

У кваліфікаційній роботі розв'язано завдання проєктування, реалізації та перевірки безпечної багатокомпонентної вебінфраструктури. Поставленої мети досягнуто шляхом створення лабораторного стенда, у якому функції взаємодії з користувачем, обробки запитів і зберігання даних розподілено між окремими компонентами з визначеними повноваженнями та контрольованими каналами зв'язку.

У теоретичній частині досліджено основні принципи побудови захищених багаторівневих систем. Розглянуто сегментацію, ізоляцію компонентів, довірчі межі, мінімізацію площі атаки та принцип найменших привілеїв. Проаналізовано особливості контейнеризації, застосування вебсервера як єдиної зовнішньої точки входу, організації прикладного програмного інтерфейсу та захисту рівня даних. На основі порівняння альтернативних технологій обґрунтовано вибір програмних засобів для реалізації розробленої системи.

У процесі проєктування визначено структуру інфраструктури, функції її компонентів, дозволені напрями передавання даних і засоби обмеження небажаних з'єднань. Запропонована архітектура передбачає відокремлення клієнтського, прикладного та інформаційного рівнів, що зменшує кількість прямих зв'язків між компонентами та спрощує контроль за проходженням запитів.

У практичній частині розгорнуто клієнтський вебінтерфейс, зовнішній вебсервер, прикладний сервер і окремий сервер бази даних. Реалізовано захищене передавання запитів, серверну перевірку вхідних даних, параметризоване виконання SQL-команд, обмеження частоти звернень, контроль браузерних джерел і розмежування прав службового облікового запису. Внутрішні служби не використовуються як самостійні зовнішні точки доступу, а взаємодія між рівнями здійснюється відповідно до визначеної архітектури.

Проведені функціональні та контрольні перевірки підтвердили працездатність основних компонентів і реалізованих механізмів захисту. Система коректно виконує дозволені операції, відхиляє некоректні вхідні дані, обмежує надмірну кількість запитів і не допускає виконання введених користувачем значень

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				78
Змн.	Арк.	№ докум.	Підпис	Дата		

як частини SQL-команд. Результати тестування також підтвердили правильність мережевого розмежування та контрольованого доступу до внутрішніх ресурсів.

Практичним результатом роботи є реалізована багатокомпонентна вебінфраструктура з розмежованими функціями, контрольованою мережевою взаємодією та обмеженими повноваженнями її компонентів. Розроблене рішення може використовуватися як основа для навчання, дослідження механізмів кіберзахисту та подальшого вдосконалення вебсистем. Значення отриманого результату полягає в тому, що безпека розглядається не як окрема функція одного компонента, а як властивість усієї архітектури, у якій кожний рівень виконує визначені завдання та не отримує надмірного доступу до інших частин системи.

Таким чином, поставлені завдання виконано, а отримані результати підтвердили доцільність комплексного застосування мережевих, серверних і прикладних механізмів захисту під час побудови багатокомпонентної вебінфраструктури.

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		79

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. National Institute of Standards and Technology. Application Container Security Guide [Електронний ресурс] – 2017 – Режим доступу: <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-190.pdf>
2. National Institute of Standards and Technology. Least Privilege [Електронний ресурс] – Режим доступу: https://csrc.nist.gov/glossary/term/least_privilege
3. Microsoft Learn. Containers vs. Virtual Machines [Електронний ресурс] – 2025 – Режим доступу: <https://learn.microsoft.com/en-us/virtualization/windowscontainers/about/containers-vs-vm>
4. OWASP Cheat Sheet Series. Logging Cheat Sheet [Електронний ресурс] – Режим доступу: https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html
5. Docker Docs. What is Docker? [Електронний ресурс] – Режим доступу: <https://docs.docker.com/get-started/docker-overview/>
6. Docker Docs. What is an Image? [Електронний ресурс] – Режим доступу: <https://docs.docker.com/get-started/docker-concepts/the-basics/what-is-an-image/>
7. Docker Docs. Writing a Dockerfile [Електронний ресурс] – Режим доступу: <https://docs.docker.com/get-started/docker-concepts/building-images/writing-a-dockerfile/>
8. Docker Docs. Building Best Practices [Електронний ресурс] – Режим доступу: <https://docs.docker.com/build/building/best-practices/>
9. Docker Docs. Publishing and Exposing Ports [Електронний ресурс] – Режим доступу: <https://docs.docker.com/get-started/docker-concepts/running-containers/publishing-ports/>
10. Docker Docs. Docker Desktop WSL 2 Backend on Windows [Електронний ресурс] – Режим доступу: <https://docs.docker.com/desktop/features/wsl/>
11. Microsoft Learn. Introduction to IIS Architectures [Електронний ресурс] – 2023 – Режим доступу: <https://learn.microsoft.com/en-us/iis/get-started/introduction-to-iis/introduction-to-iis-architecture>

		Заруцький Р.В.			<i>ККІКБ.КР.Б – 125 – 5 – ПЗ</i>	Арк.
		Фальковський І.Г.				80
Змн.	Арк.	№ докум.	Підпис	Дата		

12. Microsoft Learn. Reverse Proxy with URL Rewrite v2 and Application Request Routing [Електронний ресурс] – 2022 – Режим доступу: <https://learn.microsoft.com/en-us/iis/extensions/url-rewrite-module/reverse-proxy-with-url-rewrite-v2-and-application-request-routing>
13. Microsoft Learn. URL Rewrite Module Configuration Reference [Електронний ресурс] – 2020 – Режим доступу: <https://learn.microsoft.com/en-us/iis/extensions/url-rewrite-module/url-rewrite-module-configuration-reference>
14. OWASP Cheat Sheet Series. Transport Layer Security Cheat Sheet [Електронний ресурс] – Режим доступу: https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Security_Cheat_Sheet.html
15. Microsoft Learn. Binding <binding> [Електронний ресурс] – 2025 – Режим доступу: <https://learn.microsoft.com/en-us/iis/configuration/system.applicationhost/sites/site/bindings/binding>
16. Microsoft Learn. Configure Logging in IIS [Електронний ресурс] – 2020 – Режим доступу: <https://learn.microsoft.com/en-us/iis/manage/provisioning-and-managing-iis/configure-logging-in-iis>
17. Node.js. Introduction to Node.js [Електронний ресурс] – Режим доступу: <https://nodejs.org/learn/getting-started/introduction-to-nodejs>
18. npm Docs. About npm [Електронний ресурс] – Режим доступу: <https://docs.npmjs.com/about-npm>
19. OWASP Cheat Sheet Series. REST Security Cheat Sheet [Електронний ресурс] – Режим доступу: https://cheatsheetseries.owasp.org/cheatsheets/REST_Security_Cheat_Sheet.html
20. Microsoft Learn. What Is SQL Server? [Електронний ресурс] – 2026 – Режим доступу: <https://learn.microsoft.com/en-us/sql/sql-server/what-is-sql-server?view=sql-server-ver17>

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				81
Змн.	Арк.	№ докум.	Підпис	Дата		

ДОДАТКИ

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				82
Змн.	Арк.	№ докум.	Підпис	Дата		

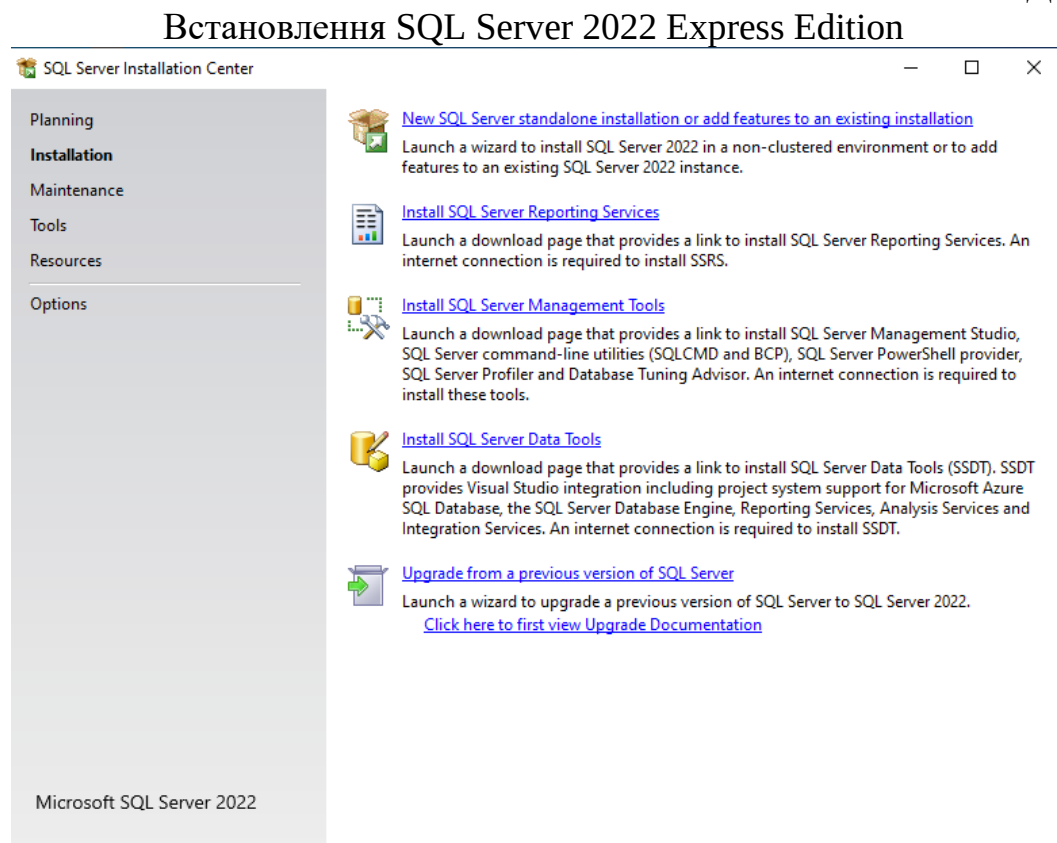


Рис. А.1 – Installation

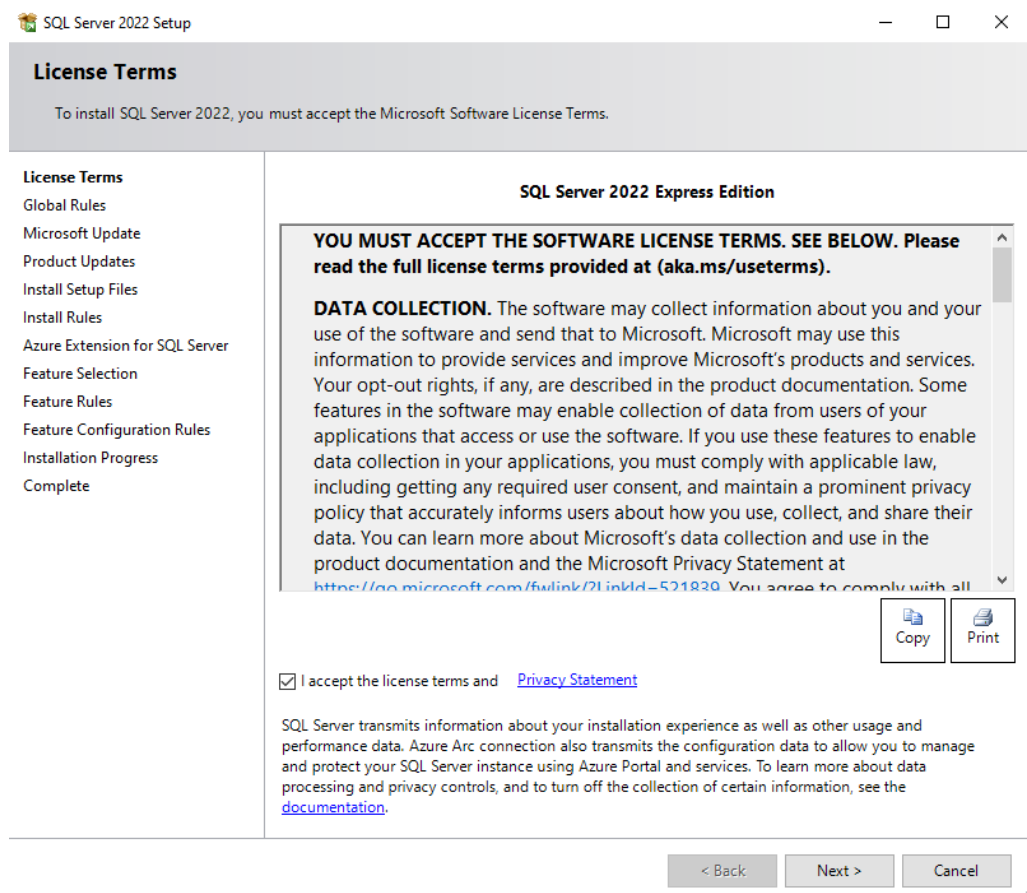


Рис. А.2 – License Terms

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				83
Змн.	Арк.	№ докум.	Підпис	Дата		

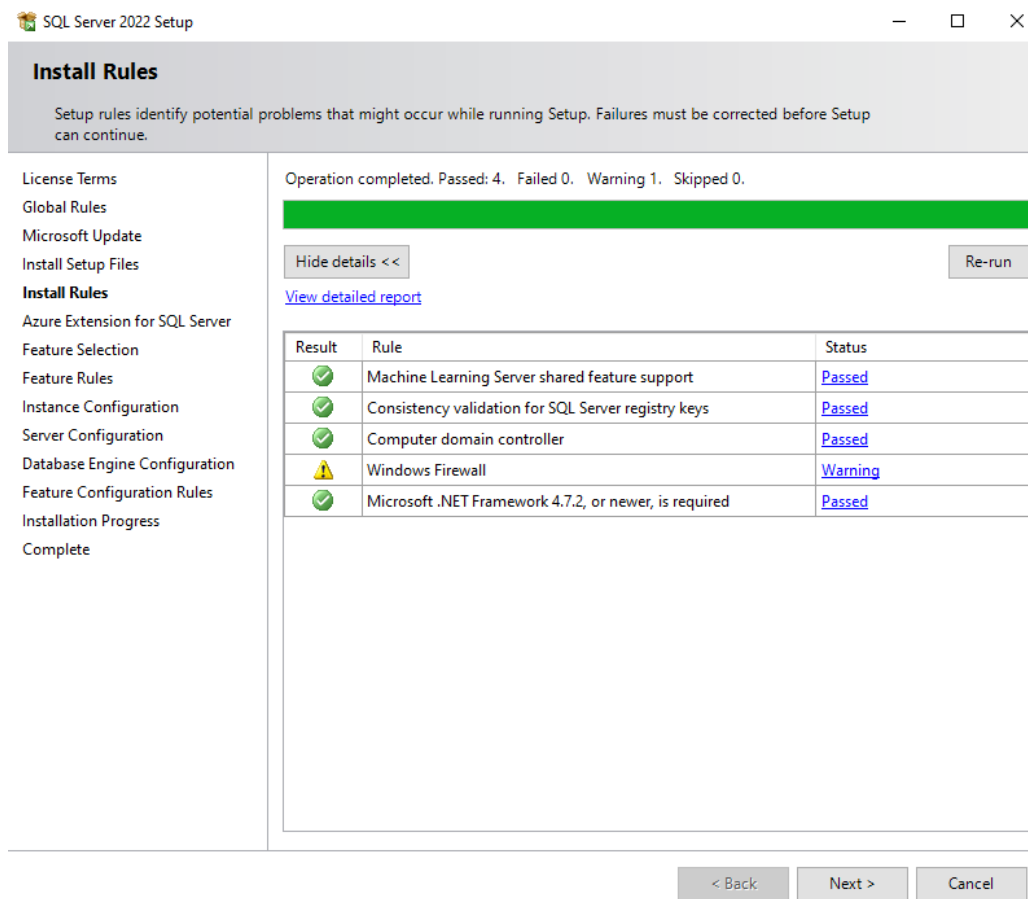


Рис. А.3 – Install Rules

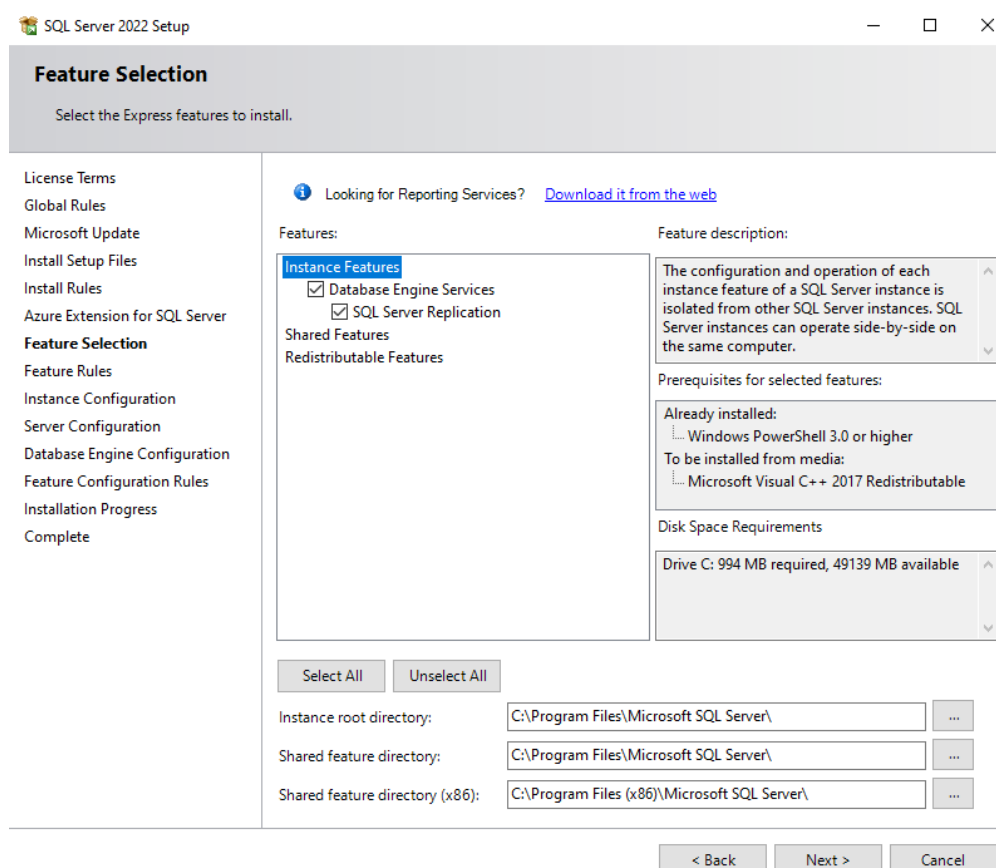


Рис. А.4 – Feature Selection

SQL Server 2022 Setup

Instance Configuration

Specify the name and instance ID for the instance of SQL Server. Instance ID becomes part of the installation path.

License Terms

Global Rules

Microsoft Update

Install Setup Files

Install Rules

Azure Extension for SQL Server

Feature Selection

Feature Rules

Instance Configuration

Server Configuration

Database Engine Configuration

Feature Configuration Rules

Installation Progress

Complete

☐ Default instance

☒ Named instance: *

SQLExpress

Instance ID:

SQLEXPRESS

SQL Server directory:

C:\Program Files\Microsoft SQL Server\MSSQL16.SQLEXPRESS

Installed instances:

Instance Name	Instance ID	Features	Edition	Version
---------------	-------------	----------	---------	---------

< Back

Next >

Cancel

SQL Server 2022 Setup

Server Configuration

Specify the service accounts and collation configuration.

License Terms

Global Rules

Microsoft Update

Install Setup Files

Install Rules

Azure Extension for SQL Server

Feature Selection

Feature Rules

Instance Configuration

Server Configuration

Database Engine Configuration

Feature Configuration Rules

Installation Progress

Complete

Service Accounts

Collation

Microsoft recommends that you use a separate account for each SQL Server service.

Service	Account Name	Password	Startup Type
SQL Server Database Engine	NT Service\MSSQL\$SQL...		Automatic
SQL Server Browser	NT AUTHORITY\LOCAL ...		Disabled

☐ Grant Perform Volume Maintenance Tasks privilege to SQL Server Database Engine Service

This privilege enables instant file initialization by avoiding zeroing of data pages. This may lead to information disclosure by allowing deleted content to be accessed.

[Click here for details](#)

< Back

Next >

Cancel

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				85
Змн.	Арк.	№ докум.	Підпис	Дата		

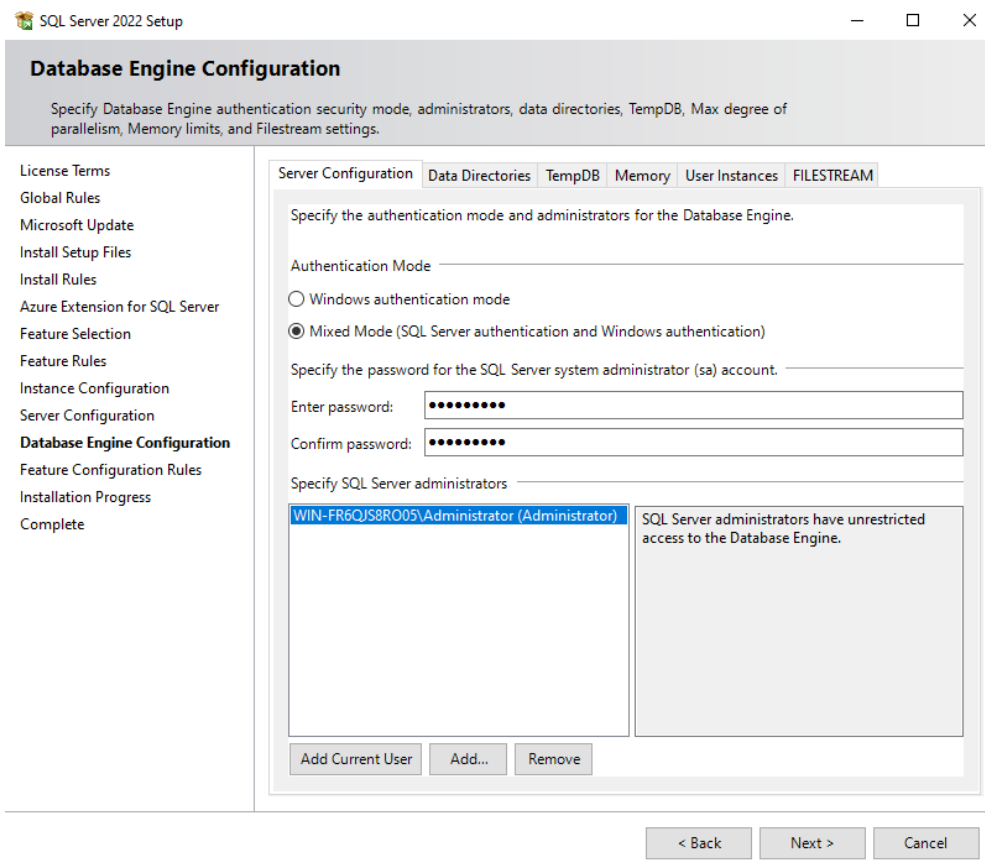


Рис. А.5 - А.7 – Configuration

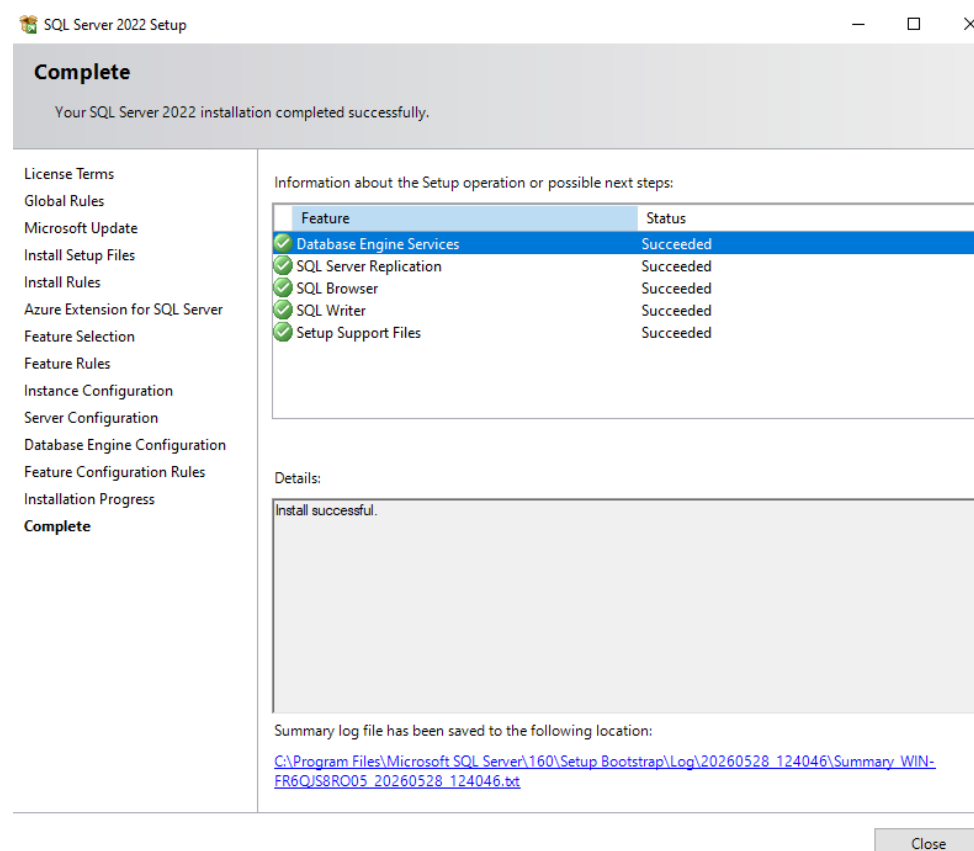


Рис. А.8 – Completion

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		86

Встановлення IIS

 Add Roles and Features Wizard — □ ×

Before you begin DESTINATION SERVER
WIN-FR6QJ58RO05

Before You Begin

Installation Type
Server Selection
Server Roles
Features
Confirmation
Results

This wizard helps you install roles, role services, or features. You determine which roles, role services, or features to install based on the computing needs of your organization, such as sharing documents, or hosting a website.

To remove roles, role services, or features:
[Start the Remove Roles and Features Wizard](#)

Before you continue, verify that the following tasks have been completed:

- The Administrator account has a strong password
- Network settings, such as static IP addresses, are configured
- The most current security updates from Windows Update are installed

If you must verify that any of the preceding prerequisites have been completed, close the wizard, complete the steps, and then run the wizard again.

To continue, click Next.

☐ Skip this page by default

< Previous Next > Install Cancel

 Add Roles and Features Wizard — □ ×

Select installation type DESTINATION SERVER
WIN-FR6QJ58RO05

Before You Begin

Installation Type
Server Selection
Server Roles
Features
Confirmation
Results

Select the installation type. You can install roles and features on a running physical computer or virtual machine, or on an offline virtual hard disk (VHD).

☒ **Role-based or feature-based installation**
Configure a single server by adding roles, role services, and features.

☐ **Remote Desktop Services installation**
Install required role services for Virtual Desktop Infrastructure (VDI) to create a virtual machine-based or session-based desktop deployment.

< Previous Next > Install Cancel

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				87
Змн.	Арк.	№ докум.	Підпис	Дата		

Add Roles and Features Wizard

Select destination server

Before You Begin
Installation Type
Server Selection
Server Roles
Features
Confirmation
Results

Select a server or a virtual hard disk on which to install roles and features.

☒ Select a server from the server pool
☐ Select a virtual hard disk

Server Pool

Filter:

Name
IP Address
Operating System
WIN-FR6QJS8R005
192.168.56.102
Microsoft Windows Server 2022 Standard Evaluation

1 Computer(s) found

This page shows servers that are running Windows Server 2012 or a newer release of Windows Server, and that have been added by using the Add Servers command in Server Manager. Offline servers and newly-added servers from which data collection is still incomplete are not shown.

< Previous

Next >

Install

Cancel

Add Roles and Features Wizard

Select server roles

Before You Begin
Installation Type
Server Selection
Server Roles
Features
Confirmation
Results

Select one or more roles to install on the server.

Roles

☐ Active Directory Certificate Services
☐ Active Directory Domain Services
☐ Active Directory Federation Services
☐ Active Directory Lightweight Directory Services
☐ Active Directory Rights Management Services
☐ Device Health Attestation
☐ DHCP Server
☐ DNS Server
☐ Fax Server
☒ File and Storage Services (1 of 12 roles)
☐ Host Guardian Service
☐ Hyper-V
☐ Network Policy and Access Services
☐ Print and Document Services
☐ Remote Access
☐ Remote Desktop Services
☐ Volume Activation Services
☒ **Web Server (IIS)**
☐ Windows Deployment Services
☐ Windows Server Update Services

Add features that are required for Web Server (IIS)?

The following tools are required to manage this feature, but do not have to be installed on the same server.

Web Server (IIS)
Management Tools
[Tools] IIS Management Console

☒ Include management tools (if applicable)

Add Features

Cancel

< Previous

Next >

Install

Cancel

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				88
Змн.	Арк.	№ докум.	Підпис	Дата		

Select features

DESTINATION SERVER
WIN-FR6QJ58R005

Before You Begin

Installation Type

Server Selection

Server Roles

Features

Web Server Role (IIS)

Role Services

Confirmation

Results

Select one or more features to install on the selected server.

Features

- ☐ .NET Framework 3.5 Features
- ☒ .NET Framework 4.8 Features (2 of 7 installed)
 - ☒ .NET Framework 4.8 (Installed)
 - ☐ ASP.NET 4.8
- ☒ WCF Services (1 of 5 installed)
 - ☒ Background Intelligent Transfer Service (BITS)
 - ☒ IIS Server Extension
 - ☐ Compact Server
- ☐ BitLocker Drive Encryption
- ☐ BitLocker Network Unlock
- ☐ BranchCache
- ☐ Client for NFS
- ☒ Containers
- ☐ Data Center Bridging
- ☐ Direct Play
- ☐ Enhanced Storage
- ☐ Failover Clustering
- ☐ Group Policy Management
- ☐ Host Guardian Hyper-V Support

Description

Provides services and tools to create and manage Windows Server Containers and their resources.

< Previous

Next >

Install

Cancel

DESTINATION SERVER

Select features

Before You Begin

Installation Type

Server Selection

Server Roles

Features

Web Server Role (IIS)

Role Services

Confirmation

Results

Select one or more features to install on the selected server.

Features

- ☐ .NET Framework 3.5 Features
- ☒ .NET Framework 4.8 Features (2 of 7 installed)
 - ☒ .NET Framework 4.8 (Installed)
 - ☐ ASP.NET 4.8
- ☒ Background Intelligent Transfer Service (BITS)
 - ☒ IIS Server Extension
 - ☐ Compact Server
- ☐ BitLocker Drive Encryption
- ☐ BitLocker Network Unlock
- ☐ BranchCache
- ☐ Client for NFS
- ☐ Containers
- ☐ Data Center Bridging
- ☐ Direct Play
- ☐ Enhanced Storage
- ☐ Failover Clustering
- ☐ Group Policy Management
- ☐ Host Guardian Hyper-V Support
- ☐ I/O Quality of Service
- ☐ IIS Hostable Web Core
- ☐ Internet Printing Client

Add features that are required for IIS Server Extension?

You cannot install IIS Server Extension unless the following role services or features are also installed.

- Background Intelligent Transfer Service (BITS)
 - Remote Server Administration Tools
 - Feature Administration Tools
 - [Tools] BITS Server Extensions Tools
- Web Server (IIS)
 - Management Tools
 - IIS 6 Management Compatibility
 - IIS 6 Metabase Compatibility
 - Web Server
 - Common HTTP Features
 - HTTP Redirection
 - Health and Diagnostics

☒ Include management tools (if applicable)

Add Features

Cancel

< Previous

Next >

Install

Cancel

		Заруцький Р.В.		
		Фальковський І.Г.		
Змн.	Арк.	№ докум.	Підпис	Дата

ККІКБ.КР.Б – 125 – 5 – ПЗ

Арк.

89

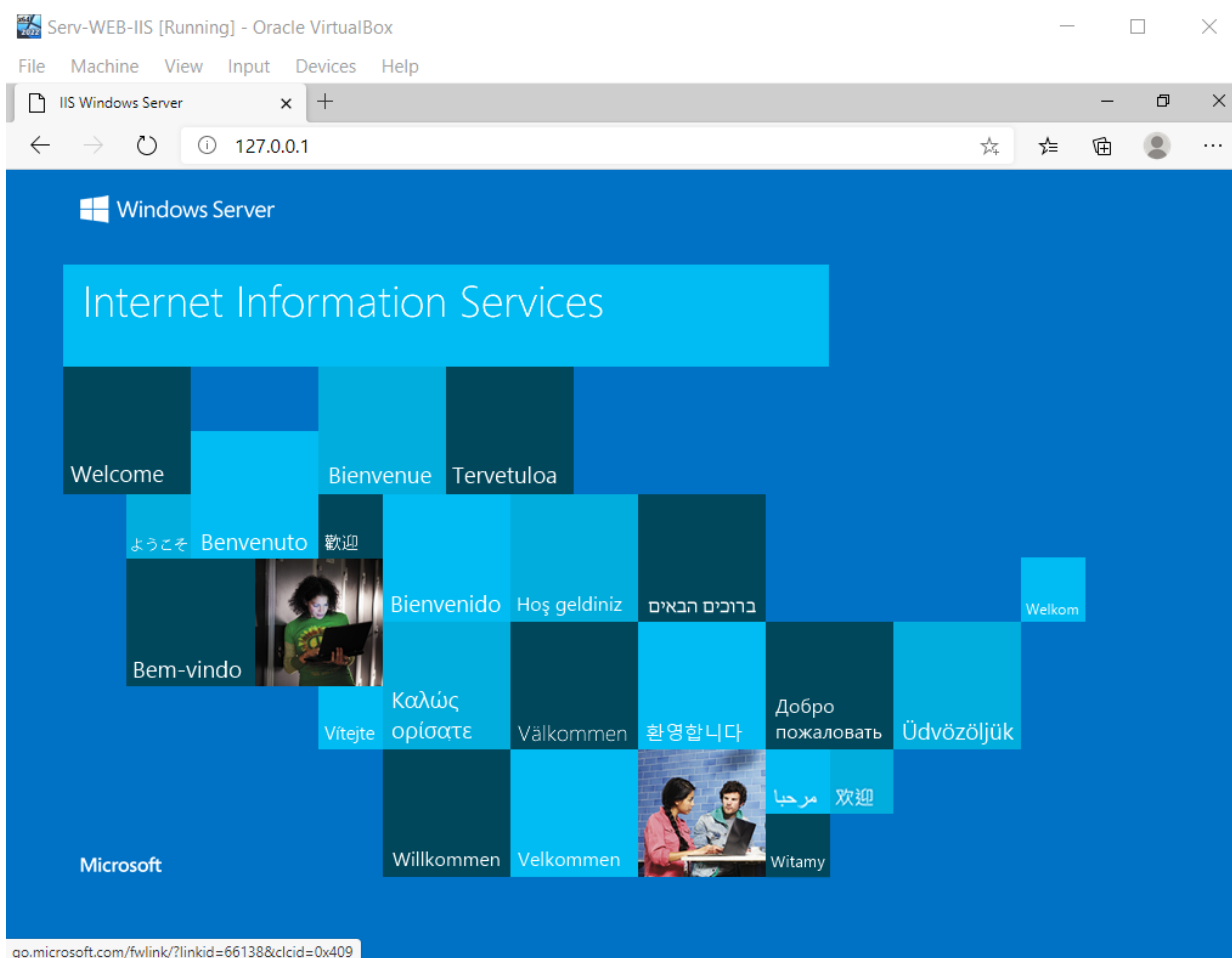
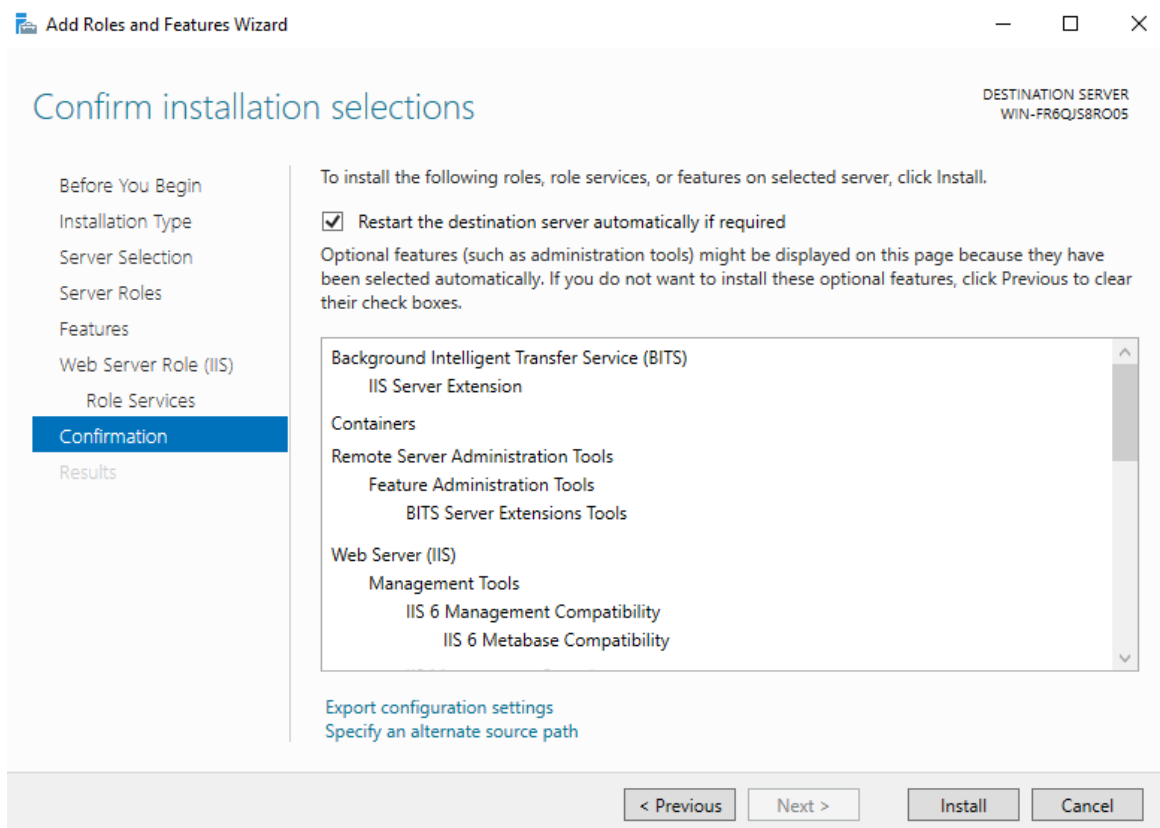


Рис. Б.1-Б.8 – Повний процес встановлення

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				90
Змн.	Арк.	№ докум.	Підпис	Дата		

Фрагменти програмного коду реалізованої системи

Лістинг index.html(основний

вузол):

```

<!DOCTYPE html>
<html>
<head>
  <title>Messages
UI</title>
  <link rel="stylesheet"
href="styles.css">
</head>
<body>
<h1>Messages</h1>
<div id="list"></div>
<input id="user">
<input id="msg">
<button
onclick="send()">Send</button>
<script
src="app.js"></script>
</body>
</html>

```

Лістинг styles.css(основний

вузол):

```

body {
  font-family: Arial;
  margin: 30px;
}
#list {
  margin-top: 20px;
  padding: 10px;
  border: 1px solid #ccc;
}
Лістинг app.js(основний вузол):
const API =
"http://192.168.56.102/api/message
s";
  async function load() {
    const res = await
fetch(API);
    const data = await
res.json();
    list.innerHTML =
data.map(x =>
      `<p><b>${x.Username}<
/b>: ${x.MessageText}</p>`
    ).join('');

```

```

}
  async function send() {
    await fetch(API, {
      method: "POST",
      headers: { "Content-
Type": "application/json" },
      body:
JSON.stringify({
        Username:
user.value,
        MessageText:
msg.value
      })
    });
    load();
  }
  load();

```

Лістинг Dockerfile(основний

вузол):

```

FROM node:20
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN chown -R node:node /app
USER node
EXPOSE 80
CMD ["node", "server.js"]
Лістинг package.json(основний

```

вузол):

```

{
  "name": "docker-ui",
  "version": "1.0.0",
  "dependencies": {
    "express": "^4.18.2"
  }
}

```

Лістинг server.js(основний

вузол):

```

const express =
require('express');
const app = express();
app.use(express.static('publi
c'));
app.listen(80, () => {

```

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				91
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        console.log('UI
running');
    });
    ЛІСТИНГ server.js(Serv-WEB-IIS):

    require('dotenv').config();
    const express =
require('express');
    const sql = require('mssql');
    const helmet =
require('helmet');
    const { rateLimit,
ipKeyGenerator } =
require('express-rate-limit');
    const cors = require('cors');
    const app = express();
    app.set('trust proxy',
'loopback');
    app.disable('x-powered-by');
    const allowedOrigins = [
        'http://localhost:8080'
    ];
    app.use(cors({
        origin(origin, callback)
    {
        if (!origin ||
allowedOrigins.includes(origin)) {
            return
callback(null, true);
        }
        return callback(new
Error('CORS_ORIGIN_NOT_ALLOWED'));
    },
        methods: ['GET', 'POST',
'OPTIONS'],
        allowedHeaders:
['Content-Type']
    })));
    app.use(express.json());
    app.use(helmet());
    app.use(rateLimit({
        windowMs: 15 * 60 * 1000,
        max: 100,
        standardHeaders: true,
        legacyHeaders: false,
        keyGenerator: req => {
            let ip = req.ip ||
req.socket.remoteAddress ||
'unknown';

            const ipv4WithPort =
ip.match(

```

```

/^(\d{1,3}(\?:\.\d{1,3}){3}):\d+$/
        );
        if (ipv4WithPort) {
            ip =
ipv4WithPort[1];
        }
        const ipv6WithPort =
ip.match(
        /^\[([0-9a-fA-
F:]+)\]:\d+$/
        );
        if (ipv6WithPort) {
            ip =
ipv6WithPort[1];
        }
        return
ipKeyGenerator(ip);
    },
    message: 'Too many
requests'
    }));
    const dbConfig = {
        user:
process.env.DB_USER,
        password:
process.env.DB_PASSWORD,
        server:
process.env.DB_SERVER,
        database:
process.env.DB_DATABASE,
        options: {

trustServerCertificate: true
        }
    };
    app.get('/api/messages',
async (req, res) => {
        try {
            const pool = await
sql.connect(dbConfig);
            const result = await
pool.request().query(`
                SELECT Username,
                MessageText FROM MessagesTable
            `);
            return
res.json(result.recordset);
        } catch (err) {
            console.error(err);
        }
    }

```

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		92

```

        return
res.status(500).send('Server
error');
    }
    });
    app.post('/api/messages',
async (req, res) => {
    console.log('POST
WORKS:', req.body);
    const username =
        typeof
req.body?.Username === 'string'
        ?
req.body.Username.trim()
        : '';
    const messageText =
        typeof
req.body?.MessageText === 'string'
        ?
req.body.MessageText.trim()
        : '';
    if (username.length < 1
|| username.length > 50) {
        return
res.status(400).send(
            'Username must
contain from 1 to 50 characters'
        );
    }
    if (messageText.length <
1 || messageText.length > 100) {
        return
res.status(400).send(
            'MessageText must
contain from 1 to 100 characters'
        );
    }
    try {
        const pool = await
sql.connect(dbConfig);
        await pool.request()

.input('Username',
sql.NVarChar(50), username)

.input('MessageText',
sql.NVarChar(100), messageText)
        .query(`
            INSERT INTO
MessagesTable (Username,
MessageText)

```

```

VALUES
(@Username, @MessageText)
        `);
        return
res.status(200).send('Inserted');
    } catch (err) {
        console.error(err);
        return
res.status(500).send('DB error');
    }
    });
    app.use((err, req, res, next)
=> {
        if (err?.message ===
'CORS_ORIGIN_NOT_ALLOWED') {
            return
res.status(403).send('CORS origin
is not allowed');
        }
        if (err instanceof
SyntaxError && err.status === 400
&& 'body' in err) {
            return
res.status(400).send('Invalid
JSON');
        }
        console.error(err);
        return
res.status(500).send('Server
error');
    });
    console.log('PORT from env:',
process.env.PORT);
    app.listen(process.env.PORT,
'127.0.0.1', () => {
        console.log('API running
on port', process.env.PORT);
    });
    Лістинг package.json(Serv-WEB-
IIS):
    {
        "name": "Backend API",
        "version": "1.0.0",
        "description": "",
        "main": "index.js",
        "scripts": {
            "test": "echo \"Error: no
test specified\" && exit 1"
        },
        "keywords": [],

```

```

"author": "",
"license": "ISC",
"type": "commonjs",
"dependencies": {
  "cors": "^2.8.6",
  "dotenv": "^17.4.2",
  "express": "^5.2.1",
  "express-rate-limit":
"^8.5.2",
  "helmet": "^8.2.0",

```

```

    "mssql": "^12.5.5"
  }
}
Лістинг .ENV(Serv-WEB-IIS):
DB_USER=webapp_user
DB_PASSWORD=WebPass1234*
DB_SERVER=192.168.56.101
DB_DATABASE=QualificationDB
PORT=3000

```

		Заруцький Р.В.			ККІКБ.КР.Б – 125 – 5 – ПЗ	Арк.
		Фальковський І.Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		94

Схема мережевої взаємодії компонентів

