Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«ІФОРМАЦІЙНА СИСТЕМА ДЛЯ МОНІТОРИНГУ НАЯВНОСТІ ТОВАРІВ З ФУНКЦІЄЮ СПОВІЩЕНЬ. ПРОГРАМНА ЧАСТИНА»

08— 54.БКР.024.00.000 ТЗ

Виконав: студент 4 курсу, групи 2КІ— 23мс
спеціальності 123 – Комп'ютерна інженерія
(шифр і назва напряму підготовки спеціальності)
_____ Дерепа І.В.
(прізвище та ініціали)

Керівник: к.т.н., доцент
_____ Дудник О.В.
(прізвище та ініціали)
«___» _____ 2025 р.

Резензент: к.т.н., доцент
_____ Романюк О. В.
(прізвище та ініціали)
«___» _____ 2025 р.

Допущено до захисту
Завідувач кафедри ОТ
Д.т.н., проф. Азаров О.Д.
12 06 2025 року

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Освітньо— кваліфікаційний рівень бакалавр
Галузь знань – 12 Інформаційні технології
Спеціальність – 123 Комп'ютерна інженерія
Освітня програма – Комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н., проф. Азаров О.Д.
«__» _____ 2025 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Дерепі Івану Володимировичу

1 Тема роботи: «Інформаційна система для моніторингу наявності товарів з функцією сповіщень. Програмна частина», керівник роботи Дудник О. В. затверджені наказом вищого навчального закладу від 20 березня 2025 р. №97.

2 Термін подання студентом роботи 13.05.2025.

3 Вихідні дані до роботи: посилання на цільові сторінки товарів у веб-магазинах, технічні вимоги до функціоналу моніторингу та сповіщення, програмний код серверної частини (Node.js + Express), що відповідає за перевірку доступності товарів та взаємодію з базою даних, код клієнтської частини (React), що реалізує інтерфейс користувача, інтеграція з Telegram API для надсилання сповіщень, реалізація обміну даними у реальному часі за допомогою WebSocket, документація до програмного забезпечення, включно з описом архітектури, інструкцією з розгортання та прикладами використання.

4 Зміст розрахунково— пояснювальної роботи: вступ, аналіз предметної області та постановка задачі, вибір архітектури програмної системи моніторингу, розробка серверної частини системи, реалізація клієнтської частини та інтеграція з Telegram API, реалізація обміну даними у реальному часі (WebSocket), тестування системи та оцінка ефективності, висновки, список використаних джерел.

5 Перелік графічного матеріалу: лістинги, фото виконаних тестувань, порівняльні таблиці.

6 Консультанти розділів роботи наведені в таблиці 1.

7 Дата видачі завдання 21.03.2025 р.

Таблиця 1 – Консультанти роботи

Розділ	Прізвище, ініціали та посада консультанта	Дата		Підпис
		завдання видав	завдання прийняв	
1— 4	к.т.н., доцент каф. ОТ Дудник О.В.	21.03.2025	13.06.2025	
Нормоконтроль	ас. каф. ОТ Швець І.С.	13.05.2025	30.05.2025	

8 Календарний план виконання приведеній в таблиці 2.

Таблиця 2 – Календарний план

№	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Вибір та затвердження теми БКР	21.03.2025	21.03.2025	
2	Збір інформації	22.03.2025	31.03.2025	
3	Огляд предметної області	01.04.2025	09.04.2025	
4	Формування технічного завдання на програмну частину, постановка задачі	10.04.2025	15.04.2025	
5	Проєктування архітектури програмної системи	16.04.2025	17.04.2025	
6	Розробка серверної частини (Node.js, Puppeteer, PostgreSQL), розробка клієнтської частини (React.js)	18.04.2025	23.04.2025	
7	Інтеграція з Telegram API, реалізація WebSocket-з'єднання	24.04.2025	03.05.2025	
8	Тестування програмної частини, валідація результатів, усунення помилок	03.05.2025	04.05.2025	
9	Оформлення пояснювальної записки	06.05.2025	12.05.2025	
10	Попередній захист БКР	13.05.2025	13.05.2025	
11	Підготовка супроводжуючих документів, їх підписання, проходження нормоконтролю та тесту на плагіат	13.05.2025	13.05.2025	

Студент _____ Іван ДЕРЕПА

Керівник роботи _____ Олександр ДУДНИК

АНОТАЦІЯ

УДК 004.7

Дерепа І. В. Програмна система моніторингу та сповіщення про наявність товарів. Бакалаврська кваліфікаційна робота зі спеціальності 123 — Комп'ютерна Інженерія. Вінниця: ВНТУ, 2025. 71 с.

На укр. Мові: Бібліогр.: 17 рисунків, 6 таблиць, 23 лістингів та 20 посилань.

У даній бакалаврській дипломній роботі було розроблено веб-систему для автоматичного моніторингу доступності товарів на сайтах онлайн-магазинів. Під час виконання проєкту проаналізовано наявні рішення у сфері відстеження товарів та визначено технічні вимоги до майбутньої системи. Для реалізації було використано сучасні інструменти веб-розробки, зокрема React.js для клієнтської частини, Node.js із бібліотекою Puppeteer для автоматизації перевірки товарів, Sequelize та PostgreSQL для роботи з базою даних, а також Telegram API для надсилання повідомлень користувачу. Було спроєктовано архітектуру системи, розроблено інтерфейс користувача та реалізовано функціонал моніторингу товарів у реальному часі за допомогою WebSocket (Socket.io). Результати тестування показали надійність та ефективність розробленої системи у виконанні поставлених завдань.

Ключові слова: моніторинг товарів, веб-система, Telegram-бот, Puppeteer, автоматизація, React.js.

ABSTRACT

* Derepa I. V.* *Software System for Monitoring and Notification of Product Availability.* Bachelor's Qualification Thesis in specialty 123 — Computer Engineering. Vinnytsia: VNTU, 2025. 71 p.

In Ukrainian. Bibliography: 17 figures, 6 tables, 23 code listings and 20 references.

This bachelor's thesis presents the development of a web-based system for automated monitoring of product availability on online store websites. During the implementation of the project, existing solutions in the field of product tracking were analyzed, and technical requirements for the future system were defined. Modern web development tools were used in the implementation, including React.js for the client side, Node.js with the Puppeteer library for automating product checks, Sequelize and PostgreSQL for database interaction, and the Telegram API for sending notifications to users. The system architecture was designed, a user interface was developed, and real-time product monitoring functionality was implemented using WebSocket (Socket.io). Testing results demonstrated the reliability and efficiency of the developed system in achieving its intended tasks.

Keywords: product monitoring, web system, Telegram bot, Puppeteer, automation, React.js.

ЗМІСТ

ВСТУП

Сьогодні доступність товарів в онлайн-магазинах відіграє вирішальну роль у задоволенні попиту споживачів та успішній роботі бізнесу. Клієнти очікують отримувати актуальну інформацію про наявність продукції у реальному часі, а компанії прагнуть не втрачати потенційних покупців через відсутність потрібного товару. Проте моніторинг змін на сайтах інтернет-магазинів вручну є складним, повільним та ресурсоємним процесом.

Автоматизація моніторингу доступності товарів дозволяє значно підвищити ефективність роботи підприємства та забезпечити користувачів оперативними повідомленнями. У сучасних умовах розвитку електронної комерції особливо актуальним є створення програмних систем, які можуть виконувати такі завдання у фоновому режимі та надсилати сповіщення за допомогою зручних каналів зв’язку, таких як Telegram.

Програмна система моніторингу та сповіщення про наявність товарів — це веб-рішення, яке дозволяє автоматично перевіряти доступність обраної продукції на сторінках інтернет-магазинів та інформувати користувача про зміни. Основним завданням такої системи є своєчасне інформування про появу товарів у продажу, що дозволяє оперативно реагувати як бізнесу, так і кінцевим покупцям.

У процесі ручного моніторингу виникають такі проблеми:

— затримки в отриманні інформації про доступність товарів;

— високий ризик пропустити появу потрібної продукції;

— великі часові витрати при постійному перегляді сайтів;

— відсутність можливості контролювати декілька ресурсів одночасно.

Автоматизована система вирішує ці проблеми, забезпечуючи:

— моніторинг у режимі реального часу;

— надсилання повідомлень про зміну статусу товару;

— підключення до різних платформ онлайн-продажу;

— гнучке налаштування списку товарів, що відстежуються.

На основі проведеного аналізу можна зробити висновок про доцільність

створення власного веб-додатку, який забезпечить автоматичний моніторинг наявності товарів і надсилатиме сповіщення користувачам. Такий підхід дозволить спростити процес відстеження, підвищити його точність та забезпечити актуальну інформацію без потреби постійного ручного перегляду веб-ресурсів.

Актуальність теми полягає в необхідності створення зручного та надійного інструменту для моніторингу товарів в умовах стрімкого розвитку електронної торгівлі, зростання конкуренції та підвищення вимог до оперативності у сфері онлайн-продажу.

Об'єктом дослідження є процес автоматизованого відстеження товарів в інтернет-магазинах.

Предметом дослідження є методи та засоби розробки веб-системи моніторингу і сповіщення про наявність товарів.

Метою роботи є розробка веб-додатку, що забезпечує автоматизований моніторинг доступності товарів на сайтах онлайн-магазинів та надсилання сповіщень користувачу про зміни статусу товару.

Для досягнення поставленої мети необхідно вирішити такі **задачі**:

— провести аналіз предметної області та дослідити сучасні рішення у сфері моніторингу товарів;

— сформувати технічні вимоги до функціоналу майбутньої системи;

— розробити архітектуру веб-додатку;

— створити інтерфейс користувача з використанням сучасних веб-технологій;

— реалізувати серверну частину на базі Node.js і бібліотеки Puppeteer;

— забезпечити опрацювання даних за допомогою Sequelize та PostgreSQL;

— реалізувати систему сповіщення за допомогою Telegram API.

Наукова новизна роботи полягає у реалізації інтегрованого підходу до моніторингу товарів із використанням сучасного стека технологій, включаючи Puppeteer, React.js та Telegram API, що забезпечує автоматичну взаємодію між компонентами системи у режимі реального часу.

Апробація результатів здійснювалася в умовах тестування програмної

системи на прикладах реальних інтернет-магазинів із різною структурою сторінок, що дозволило підтвердити її працездатність та ефективність.

1 АНАЛІЗ ТА ПРОЄКТУВАННЯ СИСТЕМИ МОНІТОРИНГУ НАЯВНОСТІ ТОВАРІВ

1.1 Огляд існуючих рішень для моніторингу доступності товарів

У сучасних умовах динамічного розвитку електронної комерції питання моніторингу доступності товарів стає дедалі актуальнішим. Онлайн-торгівля потребує інструментів, що дозволяють оперативно реагувати на зміни асортименту та забезпечувати доступність інформації для клієнтів. Це особливо важливо для підприємств, що реалізують товари з високим попитом або обмеженим запасом.

Додатково було проведено детальний аналіз типових сценаріїв використання таких систем у контексті різних моделей бізнесу — від малого до середнього. Визначено, що особливу увагу необхідно приділити налаштуванням частоти перевірки сторінок, обробці змін HTML-структури цільових сайтів та оптимізації запитів для запобігання блокуванню.

Також при проєктуванні архітектури було враховано можливість майбутнього масштабування системи — зокрема, передбачено горизонтальне масштабування серверної частини, а також адаптивну логіку керування перевірками залежно від навантаження. Це дозволяє забезпечити стійкість до зростання обсягів даних без втрати швидкодії.

Крім того, було враховано впровадження механізмів кешування результатів перевірок та евристичного аналізу для виявлення атипової поведінки сторінок, що може свідчити про бан або зміни API. Це підвищує стійкість до змін середовища та забезпечує більш надійний моніторинг.

Існуючі рішення для моніторингу доступності товарів можна умовно поділити на три основні категорії: вручну керовані системи, комерційні програмні продукти та індивідуально розроблені рішення.

Перший тип – це системи, що базуються на ручному введенні та перевірці інформації. Вони мають низьку ефективність через значні витрати часу та можливість помилок через людський фактор. Такі підходи використовуються здебільшого в малому бізнесі, де обсяги моніторингу не є критичними.

Комерційні програмні рішення пропонують широкий спектр функцій, але мають кілька недоліків. По-перше, вони часто є дорогими, що ускладнює їх використання малими підприємствами. По-друге, ці продукти зазвичай не піддаються гнучкому налаштуванню для специфічних потреб компанії. Прикладами таких продуктів є сервіс Keepa, який відстежує зміни цін на платформах електронної комерції, таких як Amazon, та Price2Spy, що дозволяє моніторити конкурентів.

Індивідуально розроблені рішення створюються спеціально для конкретних компаній або потреб. Вони дозволяють врахувати специфіку бізнес-процесів, однак вимагають значних інвестицій у розробку та підтримку. Саме такий підхід було обрано для проєкту даної бакалаврської роботи.

1.2 Постановка задачі та технічні вимоги до системи

На основі аналізу існуючих рішень можна визначити ключові недоліки, які потребують вирішення: відсутність автоматизації, низька швидкість обробки даних, висока вартість впровадження готових програмних продуктів.

Задача системи полягає в автоматизації процесу моніторингу наявності товарів на вебресурсах. Основними функціональними вимогами до розроблюваної системи є:

— автоматичне перевіряння наявності товарів;

— регулярне оновлення інформації про статус продуктів;

— можливість повідомлення користувачів про зміни у статусі товарів через зручний канал, наприклад, Telegram;

— підтримка інтерактивного інтерфейсу для зручності роботи з системою;

— забезпечення можливості масштабування для роботи з великими обсягами даних.

Нефункціональні вимоги включають:

— висока продуктивність при роботі з великими даними;

— зручність інтерфейсу для користувачів;

— безпека обробки конфіденційної інформації;

— мінімальні витрати на розгортання та підтримку системи.

1.3 Вибір технологій для розробки

Розробка сучасних інформаційних систем потребує ретельного вибору технологій, які забезпечать ефективність, масштабованість та зручність використання. Для створення системи моніторингу наявності товарів були обрані технології, що відповідають як функціональним, так і нефункціональним вимогам.

Для зберігання даних було обрано PostgreSQL — реляційну базу даних із відкритим вихідним кодом. Її основними перевагами є: надійність, продуктивність, масштабованість, підтримка складних запитів і гнучкість розширення.

PostgreSQL ефективно працює з великими обсягами даних, що важливо для системи моніторингу, яка може обробляти інформацію про тисячі товарів. База даних легко адаптується до зростання кількості записів. Завдяки потужному SQL-движку можна створювати запити для аналізу даних, відстеження історії змін статусу товарів та генерації звітів. PostgreSQL підтримує розширення, що дозволяють додати специфічні функції за потреби.

Для інтеграції серверної частини з базою даних використовується Sequelize, ORM (Object-Relational Mapping), яка значно спрощує роботу з реляційними базами.

Sequelize забезпечує: зручність програмування, автоматизацію запитів, кросплатформеність.

Робота з базою даних здійснюється через об’єктно-орієнтовані моделі, що зменшує обсяг коду. Створення, оновлення та видалення записів виконується без написання сирих SQL-запитів. Sequelize підтримує різні СУБД, що дозволяє легко змінювати базу даних у майбутньому за необхідності.

Перевірка наявності товарів на вебсайтах реалізована за допомогою Puppeteer. Це сучасна бібліотека для роботи з браузером Google Chrome, яка дозволяє емулювати дії користувача. Таким чином було автоматизовано роботу браузера.

Можна виділити такі основні переваги Puppeteer: автоматизація процесів, швидкість виконання, гнучкість, безпека.

Бібліотека дозволяє відкривати сторінки, отримувати дані з HTML-елементів, натискати кнопки, виконувати запити тощо. Puppeteer працює без візуального інтерфейсу (режим headless), що пришвидшує виконання задач.

Інструмент підтримує обробку складних сторінок із динамічним вмістом, які генеруються JavaScript. Також Puppeteer дозволяє обходити CAPTCHA за допомогою інтеграції з відповідними сервісами, що розширює його функціональність.

Перед інтеграцією з месенджерами до системи проаналізуємо альтернативні засоби сповіщення користувачів. Розглянемо такі варіанти, як email-розсилки, SMS-повідомлення та push-сповіщення через веббраузер.

Email-розсилки мають високу універсальність і не потребують встановлення окремого застосунку. Проте вони не гарантують своєчасного ознайомлення користувача з повідомленням, оскільки електронні листи можуть переглядатися із затримкою. Крім того, реалізація такої системи потребує налаштування поштового сервера (SMTP), обробки HTML-шаблонів повідомлень і впровадження захисту від розпізнавання листів як спаму.

SMS-повідомлення забезпечують можливість інформування навіть за відсутності доступу до мережі Інтернет, однак потребують використання сторонніх платних сервісів, таких як Twilio, що ускладнює реалізацію системи на початковому етапі розробки (MVP) та вимагає додаткових фінансових витрат.

Push-сповіщення через браузер є сучасним засобом комунікації, що забезпечує миттєву доставку повідомлень. Водночас цей підхід передбачає низку технічних вимог — підтримку з боку браузера, обов’язкове отримання дозволу користувача, наявність HTTPS-протоколу та реалізацію механізмів на основі Service Workers, що ускладнює розгортання системи.

Telegram API – найбільш доцільний інструмент для реалізації функціоналу сповіщення. Telegram забезпечує безкоштовну, стабільну та оперативну доставку повідомлень без потреби у складній інфраструктурі. Додатковою перевагою є його

підтримка на більшості мобільних платформ, наявність зручного інтерфейсу для створення ботів, а також простота інтеграції через унікальний токен, webhook або механізм опитування (polling).

Для оповіщення користувачів про зміни статусу товарів використовується Telegram API. Цей вибір обумовлений популярністю месенджера та важливими, для даної роботи, технічними можливостями, а саме: швидкість повідомлень, простота налаштування, підтримка ботів.

Інформація передається користувачу майже миттєво, що важливо для систем реального часу. Telegram API має зручну документацію та бібліотеки для інтеграції. Через Telegram можна надсилати повідомлення, запускати автоматизовані задачі та отримувати зворотній зв’язок від користувачів.

Для створення клієнтської частини було обрано React.js. Ця бібліотека забезпечує сучасний підхід до розробки вебзастосунків.

Інтерфейс складається з незалежних компонентів, що полегшує розробку, тестування та оновлення системи. React.js використовує віртуальну DOM, що забезпечує високу продуктивність і плавність роботи навіть при великій кількості оновлень даних. React підходить як для невеликих, так і для великих проєктів із розгалуженою структурою.

Для обміну даними між сервером і клієнтом у режимі реального часу використовується Socket.io. Ця бібліотека дозволяє створити динамічний інтерфейс, який автоматично оновлюється. Зміни в базі даних одразу передаються на клієнтську частину.

Підтримуючи багатокористувацькі сценарії Socket.io дозволяє працювати з багатьма клієнтами одночасно, забезпечуючи кожного актуальною інформацією. Надійність, бібліотека підтримує автоматичним відновлення з’єднання в разі збоїв.

Для розробки серверної частини використовується Node.js з фреймворком Express.js. Node.js забезпечує: масштабованість, гнучкість та продуктивність.

Сервер може обробляти велику кількість одночасних запитів завдяки неблокуючій архітектурі. Велика кількість готових бібліотек і модулів дозволяє легко інтегрувати нові функції. Node.js оптимально підходить для обробки даних у

реальному часі. Express.js додає до Node.js функціонал для створення REST API, що спрощує обробку запитів від клієнтської частини.

Переваги обраного стеку технологій. Поєднання зазначених технологій забезпечує: швидкий розвиток, гнучкість, масштабованість та економічність.

Більшість із них мають велику спільноту, доступну документацію та підтримку. Обрані технології легко адаптуються до змін вимог. Система може ефективно працювати як із невеликою кількістю даних, так і з великими обсягами. Більшість інструментів є безкоштовними або мають ліцензії з відкритим кодом.

Цей стек забезпечує всі необхідні функціональні можливості для реалізації системи моніторингу товарів із високим рівнем надійності та ефективності.

Розробка архітектури системи починається з аналізу основних компонентів та їхньої взаємодії. Основу архітектури становлять два рівні: серверний (backend) та клієнтський (frontend).

Бекенд-сервіс є основою серверної частини системи, яка забезпечує управління даними, обробку запитів клієнтів і інтеграцію з іншими компонентами. Для реалізації бекенд-сервісу використовується платформа Node.js, яка забезпечує високу продуктивність і масштабованість завдяки асинхронній моделі обробки запитів.

На платформі Node.js обрано фреймворк Express.js, який спрощує створення веб-додатків і API. Express.js дозволяє швидко налаштувати маршрути, обробляти HTTP-запити та підключати необхідні проміжні модулі. Основні задачі бекенду можна умовно розділити на кілька ключових напрямків: обробка запитів від клієнтів, робота з базою даних, автоматизація перевірок, інтеграція з Telegram, безпека та масштабованість, проєктування бази даних.

Бекенд приймає запити від клієнтського інтерфейсу через REST API. Для цього створені спеціальні маршрути, які відповідають за:

— додавання нового товару до списку моніторингу;

— отримання списку товарів із бази даних;

— оновлення даних про статус товарів;

— видалення товарів із системи.

Ця функціональність дозволяє клієнтській частині динамічно взаємодіяти із системою, забезпечуючи користувачам зручність у керуванні моніторингом.

Для доступу до реляційної бази даних PostgreSQL використовується ORM-бібліотека Sequelize. Вона надає можливість писати SQL-запити високого рівня за допомогою об’єктів JavaScript, що значно спрощує роботу з даними. Sequelize підтримує:

— автоматичну генерацію таблиць відповідно до моделей;

— виконання складних запитів, таких як фільтрування або сортування;

— обробку транзакцій для забезпечення цілісності даних.

Модель даних для таблиці Product створена з урахуванням необхідних полів, таких як URL товару, його назва, стан доступності та час останньої перевірки.

Ключовим елементом бекенд-сервісу є автоматизація перевірок стану товарів. Для цього використовується бібліотека Puppeteer, яка дозволяє взаємодіяти з вебсайтами через headless-браузер. Puppeteer запускає скрипти, які: завантажують сторінку товару за вказаним URL, шукають потрібні елементи сторінки (наприклад, статус "В наявності"), та аналізують отримані дані та оновлюють запис у базі даних.

Цей підхід забезпечує високу точність моніторингу та дозволяє працювати із динамічними вебсторінками, які можуть використовувати JavaScript для завантаження даних.

Для миттєвого сповіщення користувачів про зміни у статусі товарів використовується Telegram API. Бот Telegram, інтегрований із системою, дозволяє надсилати повідомлення про зміну доступності товару та обробляти команди від користувачів, наприклад, переглядати список товарів або зупиняти моніторинг.

Інтеграція забезпечує швидке реагування на зміни, що особливо важливо для товарів з обмеженим запасом або високим попитом.

Для забезпечення безпеки бекенд-сервісу використовуються механізми автентифікації та авторизації. Це унеможливлює доступ до системи сторонніх осіб. Крім того, система спроєктована з урахуванням можливості масштабування. Розробка модульної архітектури дозволяє легко додавати нові функції або інтегрувати зовнішні сервіси.

Взаємодія бекенд-сервісу з іншими компонентами системи (клієнтською частиною, базою даних, зовнішніми бібліотеками та API) є ключовим елементом загальної архітектури системи моніторингу наявності товарів.

Одним із важливих етапів розробки системи моніторингу наявності товарів є проєктування структури бази даних. База даних повинна бути спроектована так, щоб вона забезпечувала ефективне зберігання та обробку великих обсягів інформації, гарантуючи швидкий доступ до даних і їх точність. Для цього необхідно врахувати всі вимоги до функціональності системи та обрати найбільш оптимальну структуру таблиць.

Основна таблиця, яка є ядром бази даних, має назву Product. Ця таблиця містить інформацію про кожен товар, що відслідковується в системі, та його поточний статус. Її структура виглядає наступним чином:

url – це унікальне посилання на продукт на вебсайті, яке дозволяє автоматично перевіряти його наявність. Це поле містить адресу сторінки товару, яку буде моніторити система. Використання унікальних URL забезпечує точність і правильність перевірок. Це поле має тип даних TEXT, оскільки URL може бути довгим;

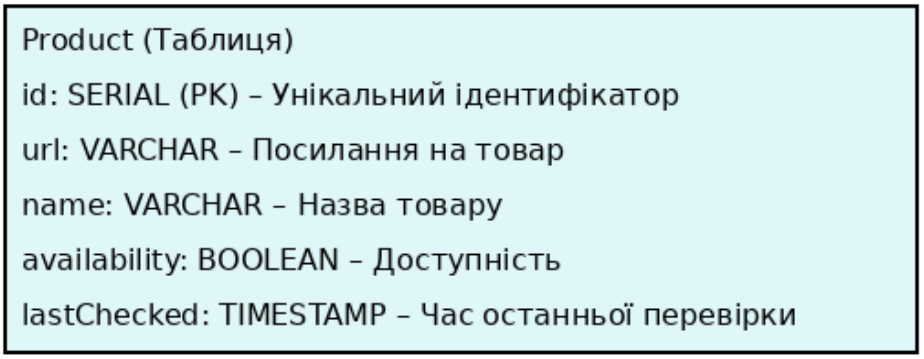
name – назва товару, що дозволяє ідентифікувати продукт для користувача. Це поле також важливе для створення зрозумілого та зручного інтерфейсу, де користувачі можуть швидко знаходити потрібні товари. Поле має тип VARCHAR, що дозволяє зберігати текстову інформацію змінної довжини;

availability – стан доступності товару. Це поле вказує, чи є товар в наявності на вебсайті, чи він відсутній. Поле може містити значення типу BOOLEAN (TRUE або FALSE), що дозволяє швидко перевірити наявність товару. У випадку потреби, можна розширити поле для збереження більш детальної інформації (наприклад, кількість товарів на складі або дата очікуваного поповнення);

lastChecked – час останньої перевірки наявності товару. Це поле необхідне для того, щоб знати, коли саме система виконувала перевірку статусу товару. Дата та час вказують на актуальність даних, що особливо важливо в умовах частих змін на сайті. Поле має тип TIMESTAMP, що дозволяє зберігати точний час перевірки.

Рисунок 1.1 демонструє схему бази даних, що наочно ілюструє взаємозв'язки між таблицями та полями, і допомагає візуалізувати архітектуру даних для майбутніх етапів розробки та тестування системи.

Рисунок 1.1– Схема бази даних

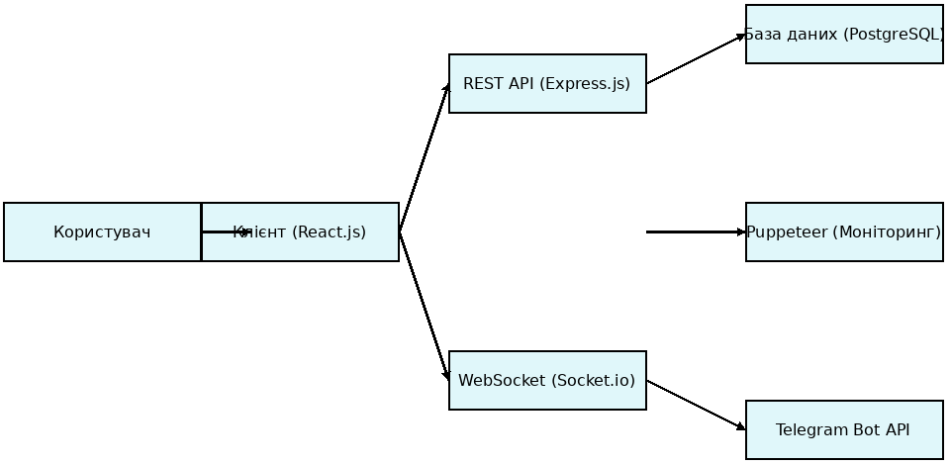


Користувач через клієнтський інтерфейс додатку додає новий товар, запити відправляються через REST API до серверної частини. Серверна частина зберігає дані в базі та планує перевірку наявності.

Puppeteer здійснює моніторинг, результати оновлюються в реальному часі через Socket.io в користувацькому інтерфейсі, оновлює інформацію в базі даних та Інформація надсилається користувачеві через Telegram бот.

Ретельний аналіз існуючих рішень, чітка постановка задачі та проектування архітектури є запорукою ефективної реалізації системи моніторингу. Рисунок 1.2 демонструє загальну схему взаємодії компонентів.

Рисунок 1.2 – Загальна схема взаємодії компонентів



1.4 Додаткові функціональні вимоги та сценарії використання

Крім базових функцій автоматичного моніторингу та сповіщення, система повинна враховувати сценарії масштабування та майбутнього розвитку. Передбачено можливість додавання ролей користувачів, підтримки багатокористувацького середовища та розмежування прав доступу. Також варто реалізувати систему логування дій користувачів та внутрішніх подій для зручного виявлення проблем та аналітики. У перспективі доцільно впровадити панель адміністратора, яка дозволить змінювати інтервали перевірок, додавати нові джерела для моніторингу, керувати сповіщеннями та статистикою по товарах.

Для забезпечення гнучкості та масштабованості системи обрано модульну архітектуру із чітким розділенням компонентів за функціональністю. Використано шаблон MVC (Model-View-Controller), де модель відповідає за логіку даних, контролер — за обробку запитів, а представлення — за взаємодію з користувачем. Комунікація між клієнтом і сервером реалізована через REST API, а для оновлень у реальному часі використано WebSocket. Такий підхід забезпечує простоту модифікації окремих модулів без впливу на інші частини системи.

2 РЕАЛІЗАЦІЯ СИСТЕМИ

2.1 Налаштування середовища роботи

У процесі реалізації системи особливу увагу було приділено не лише технічній функціональності, але й зручності її супроводу та розширення. Було впроваджено систему середовищ — dev, test і production, — що дозволяє безпечно впроваджувати нові функції та перевіряти їх перед випуском у робоче середовище. Для конфігурації середовищ використовуються змінні оточення (.env-файли), що дозволяє легко адаптувати систему до нових умов без зміни коду.

Для забезпечення стабільної роботи системи реалізовано механізми моніторингу серверної частини з використанням бібліотеки winston для логування та node-cron для планування регулярних перевірок. Також запроваджено контроль помилок на клієнтському рівні, що дозволяє користувачеві бачити сповіщення про збої у доступі до серверу чи відсутність з’єднання WebSocket.

У межах клієнтської частини було реалізовано lazy-loading компонентів та оптимізовано відображення списків за допомогою віртуалізації (наприклад, через react-window) для зменшення навантаження на DOM і забезпечення плавної взаємодії навіть при великій кількості товарів. Було також реалізовано підтримку темної теми оформлення та адаптивного дизайну, що дозволяє зручно користуватись системою як на ПК, так і на мобільних пристроях.

На рівні бази даних було впроваджено регулярне резервне копіювання (за допомогою pg_dump) та створено механізм виявлення дубльованих товарів для запобігання надмірному навантаженню. Додатково реалізовано інтерфейс адміністратора з обмеженим доступом для перегляду логів системи, журналу сповіщень та активації та деактивації окремих перевірок

Перед тим, як розпочати розробку серверної та клієнтської частин системи моніторингу наявності товарів, необхідно підготувати відповідне середовище для роботи з усіма технологіями, які використовуються у проєкті. У цьому підрозділі детально описано, як налаштувати серверне та клієнтське середовище, що дозволить розпочати розробку.

Встановлення необхідних інструментів для серверної частини. Серверна частина системи відповідає за обробку запитів від користувачів, взаємодію з базою даних, виконання перевірок наявності товарів та інтеграцію з різними API для надсилання повідомлень користувачам. Для налаштування серверної частини потрібно встановити Node.js та npm.

Середовище Node.js допоможе нам використовувати JavaScript на сервері, забезпечуючи неблокуючу, подієво-орієнтовану архітектуру, яка ідеально підходить для створення високопродуктивних вебзастосунків у реальному часі, таких як система моніторингу. Завдяки можливості виконувати JavaScript поза браузером, Node.js дає змогу обробляти запити, працювати з файловою системою, базами даних та мережею без необхідності використовувати інші мови програмування. Це дозволяє створювати повноцінний стек (full-stack) застосунок лише на JavaScript.

npm (Node Package Manager) — це стандартний менеджер пакетів для Node.js, який дозволяє легко додавати сторонні бібліотеки, оновлювати їх, керувати залежностями та створювати власні модулі. Це значно спрощує розробку, оскільки більшість необхідної функціональності вже реалізована у вигляді пакетів, які можна швидко інтегрувати.

Щоб почати роботу, необхідно завантажити та встановити Node.js із офіційного сайту, обравши рекомендовану версію з довгостроковою підтримкою (LTS). Встановлення автоматично включає npm. Після інсталяції потрібно перевірити правильність встановлення за допомогою команд node -v та npm -v у терміналі, що дасть змогу впевнитися у готовності середовища до подальшої розробки. Приклад перевірки наведено на рисунку 2.1.

Це дозволить переконатись, що Node.js та npm встановлені правильно. Для зберігання даних про товари, їх статуси та інформацію про останні перевірки використовується реляційна база даних PostgreSQL. PostgreSQL потрібно завантажити з офіційного сайту.

Встановлюємо компоненти які зображено на рисунку 2.2.

Рисунок 2.1 – Перевірка встановлення Node.js та npm

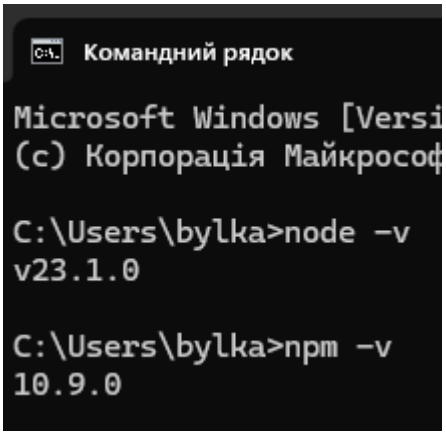
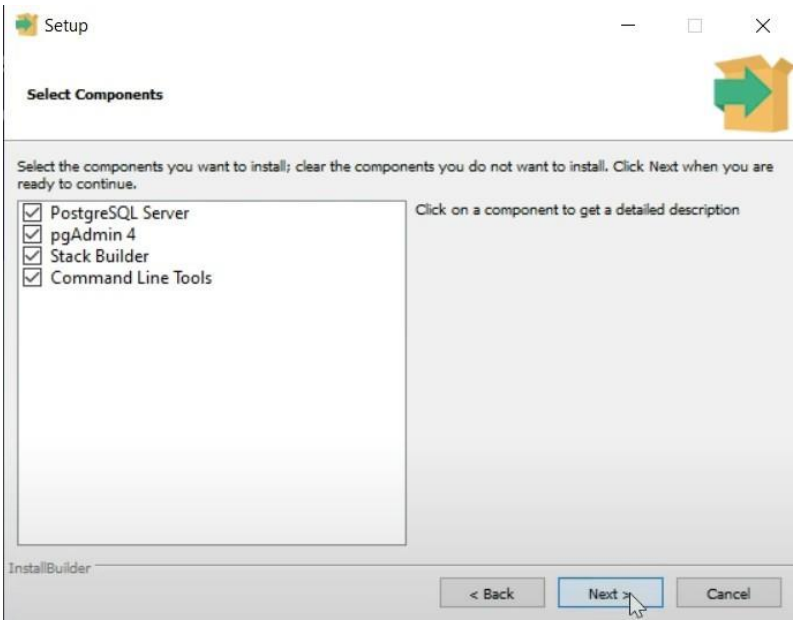


Рисунок 2.2 – Встановлення компонентів PostgreSQL



Також можна виділити компонент pgAdmin. Це графічний інтерфейс для адміністрування та керування базами даних PostgreSQL. Він дозволяє користувачам створювати та редагувати бази даних, виконувати SQL-запити, моніторинг для перегляду статистики роботи бази даних, а також її налаштувань і конфігурацій, резервне копіювання та відновлення, управління користувачами та доступами користувачів. Це потужний інструмент, який полегшує роботу з PostgreSQL, надаючи зручний інтерфейс для адміністраторів і розробників. На рис. 2.3 зображено інтерфейс pgAdmin.

Під час встановлення необхідно налаштувати користувача і базу даних для

проєкту. Далі потрібно створити нову базу даних та користувача, для цього потрібно увійти в командний рядок PostgreSQL за допомогою psql, командою “psql -U postgres”.

Далі створити базу даних за допомогою команди “CREATE DATABASE product_monitoring;”, де “product_monitoring” це назва.

Після введення цієї команди можна побачити у pgAdmin що наша база даних створилась, зображено на рисунку 2.4.

Рисунок 2.3 – Інтерфейс pgAdmin

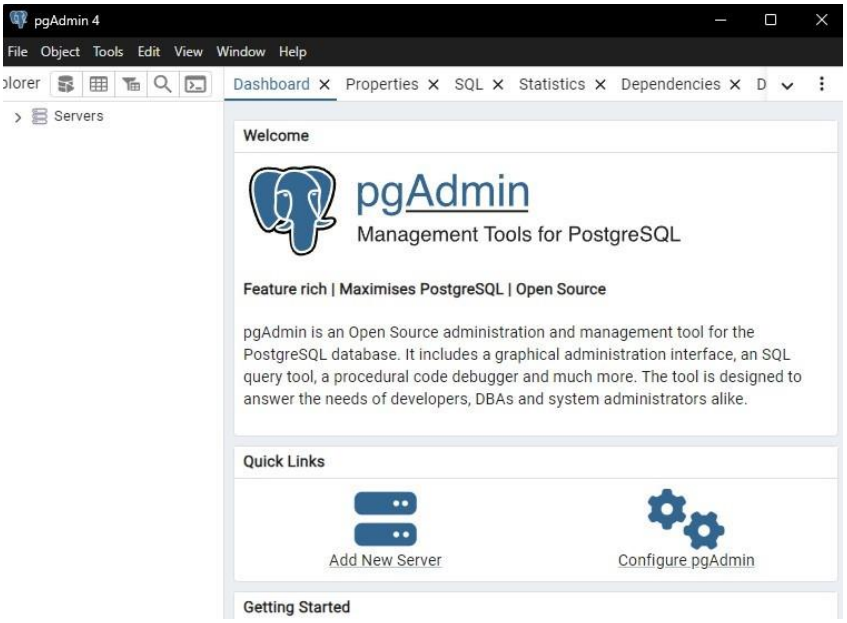
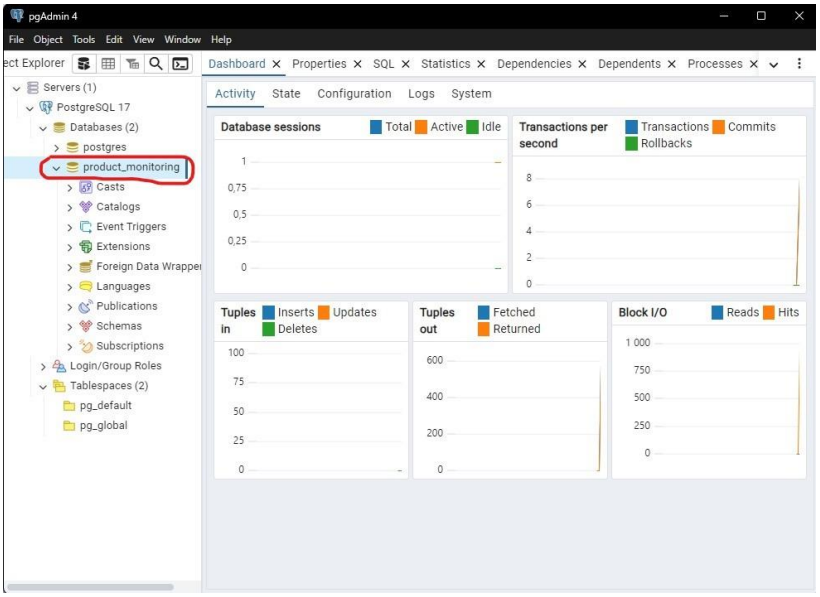


Рисунок 2.4 – Відображення новоствореної бази даних в pgAdmin



Для налаштування проєкту і встановлення всіх необхідних залежностей для серверної частини потрібно виконати наступні кроки: створити окрему теку для серверної частини, далі ініціалізувати новий проєкт Node.js за допомогою “npm init -y”, ця команда створить файл package.json, який буде містити всі налаштування проєкту; встановити необхідні бібліотеки для роботи серверної частини за допомогою команди “npm install express sequelize pg pg-hstore puppeteer dotenv socket.io node-telegram-bot-api cors”.

Для написання коду та зручної роботи з проєктом слід використовувати зручне середовище для розробки, наприклад, Visual Studio Code (VS Code). Завантажити його можна з офіційного сайту.

В теці з серверною частиною додатку, потрібно створити наступні теки та файли з розширенням “.js”:

— тека “config” у якій буде знаходитись файл “db.js” для підключення до бази даних;

— тека “controllers” у якій буде знаходитись файл “productController.js” для контролера продукту;

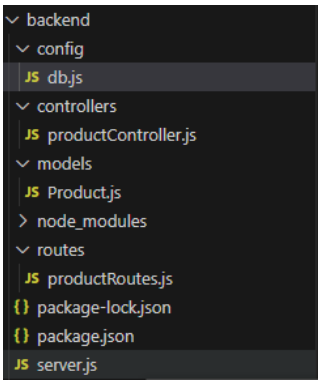
— тека “models” у якій буде знаходитись файл “Product.js” для моделі продукту;

— тека “routes” у якій буде знаходитись файл “productRoutes.js” що буде виконувати роут для продуктів;

— основний файл серверу “server.js”.

Кінечна тека з серверною частиною виглядає так, як зображено на рисунку 2.5.

Рисунок 2.5 – Тека серверної частини додатку



На цьому встановлення та налаштування необхідних інструментів для розробки серверної частини додатку закінчено.

Тепер встановлюємо та налаштовуємо клієнтську частину. Вона відповідає за взаємодію користувача з системою через інтерфейс. Для розробки інтерфейсу та налаштування клієнтської частини використовуються технології React.js та WebSocket.

Node.js та npm для клієнтської частини як і для серверної частини, потрібно мати встановлені Node.js та npm. Ці інструменти дозволяють працювати з JavaScript та бібліотеками для клієнтської частини додатку.

React.js є однією з найпопулярніших бібліотек для розробки інтерфейсів. Для ініціалізації клієнтської частини потрібно виконати команду: “npx create-react-app frontend”. Ця команда створить новий проєкт на основі React, налаштує початкову структуру файлів і встановить всі необхідні залежності.

Для стилізації компонентів використовується бібліотека styled-components та framer-motion, яка дозволяє писати CSS безпосередньо в JavaScript, дотримуючись концепції CSS-in-JS. styled-components забезпечує ізольованість стилів та дозволяє застосовувати динамічні стилі залежно від стану компонентів, що особливо корисно для індикації статусу товарів (наприклад, колір фону залежно від доступності товару). framer-motion використовується для анімації компонентів при їх появі, зникненні або зміні стану, що створює приємний візуальний досвід користувача, зокрема плавне оновлення списку товарів або анімація кнопок.

Для їх встановлення потрібно виконати команду npm install styled-components framer-motion. Крім того, для організації стилів було створено два окремі файли в теці src: GlobalStyles.js — для глобального оформлення всього застосунку, і StyledComponents.js — для збереження індивідуальних стилів таких елементів, як форми, таблиці, кнопки та картки товарів. Це дозволяє централізовано керувати виглядом інтерфейсу, підтримувати єдиний стиль та швидко вносити зміни до зовнішнього вигляду.

У GlobalStyles.js також визначено кольорову палітру, типографіку та стилі за замовчуванням для заголовків, кнопок і фону, що забезпечує єдину стилістичну

мову всього інтерфейсу. Таке розділення логіки компонентів і стилів сприяє більш чистій та підтримуваній архітектурі клієнтської частини.

Крім основних бібліотек, корисно встановити додаткові бібліотеки для роботи з HTTP-запитами та маршрутизацією. Для їх встановлення потрібно виконати команду “npm install axios socket.io-client”. axios — для відправки HTTP-запитів до серверної частини. socket.io-client — для обміну даними в реальному часі.

Після встановлення необхідних бібліотек та налаштування клієнтської частини, можна перейти до створення інтерфейсу. З допомогою React.js розроблятимуться компоненти для додавання товарів, перегляду їх статусів, отримання сповіщень та оновлення даних в реальному часі через WebSocket.
Для взаємодії між сервером і клієнтом використовуватиметься REST API для стандартних запитів (наприклад, додавання товару, оновлення інформації) та WebSocket через Socket.io для отримання оновлень у реальному часі (наприклад, зміна статусу товару).

2.2 Розробка серверної частини

Серверна частина системи моніторингу наявності товарів виконує ключові функції: обробку запитів від клієнта, взаємодію з базою даних, автоматизацію перевірки доступності товарів через браузер, а також інтеграцію з Telegram для відправлення повідомлень. Вона реалізована за допомогою Node.js та забезпечує стабільну роботу всієї системи.

Одним із найважливіших компонентів системи є база даних, яка зберігає всю інформацію про товари, їхній статус доступності, а також час останньої перевірки. Для розробки використовується реляційна база даних PostgreSQL, яка поєднує високу продуктивність і гнучкість у роботі з великими обсягами даних. Для спрощення взаємодії з базою використовується ORM-бібліотека Sequelize.

Для початку роботи з базою даних необхідно створити її структуру. Це включає створення користувача, бази даних та встановлення основних параметрів.

Підключення до бази даних. Для роботи з базою з серверної частини

використовуються параметри підключення:

— назва бази даних: monitor_products;

— ім'я користувача: postgres;

— пароль: secure_password;

— хост: localhost;

— порт: 5432;

— налаштування Sequelize.

Sequelize використовується для опису моделей і роботи з базою даних. У цьому проєкті налаштування Sequelize реалізовано у файлі db.js, який відповідає за підключення до бази даних. Код виглядає, як на рисунку 2.6.

Рисунок 2.6 – Підключення до бази даних

Цей файл дозволяє підключатися до бази даних і виконувати запити через Sequelize.

Реалізація моделі Product. Модель Product відповідає за збереження інформації про товари та має таку структуру:

— url: URL-адреса продукту (рядок, обов’язковий параметр);

— name: назва продукту (рядок, опціональний параметр);

— availability: статус доступності товару (булевий тип, що визначає, чи є товар у наявності);

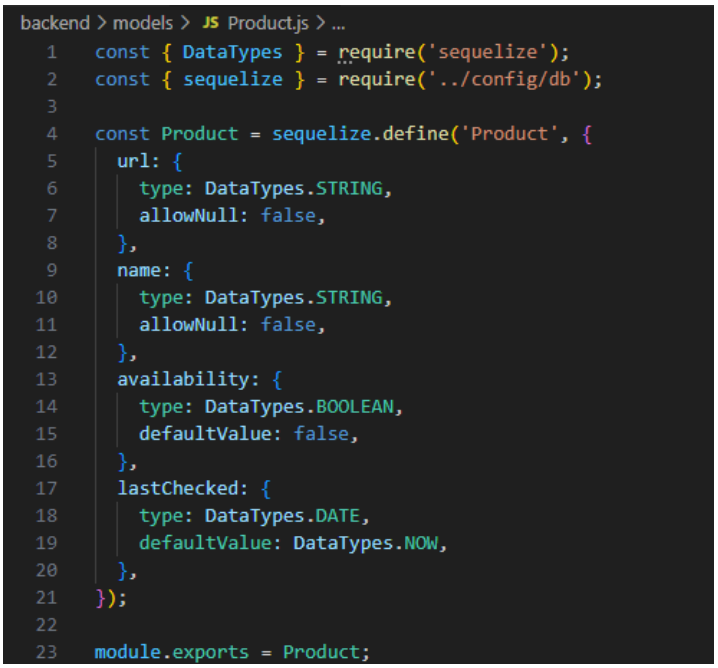
— lastChecked: дата та час останньої перевірки (дата, встановлюється автоматично).

Модель реалізована у файлі models/Product.js.

Підключення до бази даних зображено на рисунку 2.7. Функція connectDB здійснює підключення до бази та перевіряє його коректність. У разі успішного

з’єднання виводиться повідомлення PostgreSQL connected, а у разі помилки — повідомлення про неможливість підключення.

Рисунок 2.7 – Підключення до бази даних



Для створення таблиць у базі даних на основі моделі Product використовується метод sequelize.sync() у файлі server.js. Цей метод порівнює модель із базою даних і створює таблицю, якщо вона ще не існує. Код синхронізації бази даних зображено на рисунку 2.8. Метод sequelize.authenticate() - перевірка підключення до бази.

Рисунок 2.8 – Синхронізація моделі з базою даних

На основі описаної моделі Product у базі даних створюється таблиця з такою структурою, яка зображена в таблиці 2.1.

Таблиця 2.1 – Структура таблиці Product у базі даних

Поле	Тип	Обов'язковість	Опис
id	SERIAL	Так	Унікальний ідентифікатор
url	VARCHAR	Так	URL-адреса продукту
name	VARCHAR	Ні	Назва продукту
availability	BOOLEAN	Так	Статус доступності
lastChecked	TIMESTAMP	Так	Час останньої перевірки

Реалізація бази даних та моделі Product забезпечує такі можливості:

— зберігання інформації про всі товари, які моніторяться;

— автоматичне оновлення часу останньої перевірки доступності;

— гнучку інтеграцію з іншими компонентами системи.

Таким чином, база даних і модель Product є базовими елементами, які забезпечують функціональність серверної частини системи моніторингу.

Функціонал бекенду: додавання, видалення, перевірка доступності. Цей розділ описує ключові аспекти функціональності серверної частини для реалізації основних операцій із продуктами: додавання, видалення та перевірка їхньої доступності. Код організований у структурі, що забезпечує модульність, простоту розширення та підтримку.

Бекенд побудовано за модульною архітектурою з виділенням наступних основних частин:

— контролери відповідають за логіку обробки запитів до API;

— роутери забезпечують маршрутизацію запитів, визначаючи відповідний контролер;

— моделі містять структуру таблиць бази даних і функції для роботи з даними;

— сервіси реалізують бізнес-логіку, наприклад, автоматизацію перевірок через Puppeteer.

Для додавання продукту у файлі “productController.js” реалізовано функцію “addProduct”, яка дозволяє додати новий продукт до бази даних, код зображено на рисунку 2.9.

Рисунок 2.9 – Реалізіція функції додавання продукту

```
exports.addProduct = async (req, res) => {
  const { url, name } = req.body;
  try {
    const newProduct = await Product.create({ url, name });
    res.json({ message: 'Product added successfully', product: newProduct });
  } catch (error) {
    console.error("Error adding product:", error);
    res.status(500).json({ message: "Failed to add product" });
  }
};
```

Опис роботи: отримання даних із запиту, додавання в базу даних, результат, обробка помилок.

Користувач передає через req.body дані продукту: url (посилання) та name (назву).

Використовується метод Product.create() (ORM Sequelize), щоб створити запис у таблиці Product.

Повертається JSON-відповідь із повідомленням про успішне додавання та деталями нового продукту.

У разі виникнення помилки повертається статус 500 із відповідним повідомленням.

Для видалення продукту створена функція “deleteProduct” у файлі “productController.js”, вона дозволяє видалити продукт за ідентифікатором.

Отримання параметра: ідентифікатор продукту передається через req.params.id.

Видалення з бази даних: використовується метод Product.destroy(), який знаходить запис за ідентифікатором та видаляє його.

Результат: якщо запис успішно видалено, повертається статус 200 із

відповідним повідомленням; якщо запис не знайдено — статус 404 із поясненням.

Рисунок 2.10 – Реалізація функції видалення продукту

```
23  exports.deleteProduct = async (req, res) => {
24    try {
25      const { id } = req.params;
26      const deletedProduct = await Product.destroy({ where: { id } });
27
28      if (deletedProduct) {
29        return res.status(200).json({ message: "Product deleted successfully" });
30      } else {
31        return res.status(404).json({ message: "Product not found" });
32      }
33    } catch (error) {
34      console.error("Error deleting product:", error);
35      return res.status(500).json({ message: "Failed to delete product" });
36    }
37  };
```

Обробка помилок: у разі непередбачуваних помилок повертається статус 500.

Отримання всіх продуктів зображено на рисунку 2.11. Для отримання списку всіх товарів створено метод getAllProducts, який викликає Product.findAll() і повертає масив записів. У разі успіху сервер повертає статус 200 та масив продуктів.

Реалізовано функцію checkAvailability, яка перевіряє наявність товарів на сайтах і оновлює відповідні поля у базі даних. Вона зображена на рисунку 2.12.

Рисунок 2.11 – Реалізація функції, що повертає список усіх продуктів у базі даних.

```
72  exports.getAllProducts = async (req, res) => {
73    try {
74      const products = await Product.findAll();
75      res.status(200).json(products);
76    } catch (error) {
77      console.error("Error fetching products:", error);
78      res.status(500).json({ message: "Failed to fetch products" });
79    }
80  };
```

Рисунок 2.12 – Реалізація функції перевірки доступності продуктів.

```
exports.checkAvailability = async () => {
  const products = await Product.findAll();
  for (const product of products) {
    const isAvailable = await checkProductAvailability(product.url);
    if (isAvailable && !product.availability) {
      sendTelegramNotification(product, isAvailable);
    }
    await product.update({ availability: isAvailable, lastChecked: new Date() });
  }
};
```

Опис роботи контролера: функція getAllProducts отримує всі записи з таблиці Product за допомогою Product.findAll().

Перевірка доступності: для кожного продукту викликається checkProductAvailability(url), яка за допомогою бібліотеки Puppeteer відкриває веб-сторінку та шукає елементи доступності (класи .availability чи .instock-status).

Оновлення бази даних: після перевірки стану продукту (availability) та часу останньої перевірки (lastChecked) відповідні поля оновлюються в базі даних.

Якщо продукт став доступним, а раніше був недоступним, надсилається повідомлення через Telegram-бота. Усі методи контролера підключені до маршрутів у файлі productRoutes.js:

POST /add — додає новий продукт.

GET /all — повертає всі продукти.

DELETE /delete/:id — видаляє продукт за ідентифікатором.

Одним із ключових компонентів системи моніторингу є автоматизована перевірка доступності товарів. Для цього використано бібліотеку Puppeteer, яка надає інтерфейс для керування браузером Chrome/Chromium через DevTools Protocol. Puppeteer дозволяє програмно навігувати сторінками, заповнювати форми, створювати скріншоти та аналізувати DOM.

Нижче приведено ініціалізацію Puppeteer та відкриття сторінки, (лістинг 2.1). Причини вибору Puppeteer:

— гнучкість і потужність: прямий доступ до API Chrome DevTools для складних сценаріїв взаємодії;

— підтримка JavaScript: легко інтегрується з Node.js;

— стабільність: офіційна бібліотека Google з гарантією підтримки;

— основна логіка перевірки доступності.

Лістинг 2.1 – Ініціалізація Puppeteer та відкриття сторінки

```
const browser = await puppeteer.launch({});
const page = await browser.newPage();
```

puppeteer.launch() запускає новий екземпляр браузера (за замовчуванням — у headless-режимі). page.newPage() створює нову вкладку. Далі наведено створення переходів між сторінками, (лістинг 2.2).

Лістинг 2.2 – Переходи між сторінками

```
await page.goto(url, {
  waitUntil: 'domcontentloaded',
  timeout: 60000
});
```

waitUntil: 'domcontentloaded' чекає завантаження DOM. timeout: 60000 задає максимальний час очікування у 60 секунд. page.$(selector) – метод, що повертає перший елемент, який відповідає селектору. Якщо хоча б один селектор знайдено, вважається, що товар доступний, (лістинг 2.3).

Лістинг 2.3 – Аналіз DOM-елементів для визначення доступності

```
const isAvailable =
  (await page.$('.availability') !== null) ||
  (await page.$('.instock-status') !== null);
```

Перевірка доступності може стикатися з проблемами, як-от недоступність сторінки або зміна структури DOM. Для обробки таких ситуацій використовується блок try-catch, (лістинг 2.4).

Лістинг 2.4 – Обробка помилок

```
try {
  await page.goto(url, { /* ... */ });
  return isAvailable;
} catch (error) {
  console.error('Error checking product availability:', error);
  return false;
}
```

Щоб уникнути витоків пам’яті та знизити навантаження на систему, браузер закривається після завершення перевіркия, (лістинг 2.5).

Лістинг 2.5 – Закриття браузера

```
finally {
  await browser.close();
}
```

Результати перевірки записуються до бази даних, (лістинг 2.6).

Лістинг 2.6 – Оновлення стану товару в базі

```
await product.update({
  availability: isAvailable,
  lastChecked: new Date()
});
```

Puppeteer забезпечив потужний і надійний механізм перевірки доступності товарів, що дозволяє обробляти велику кількість запитів з мінімальними ресурсними витратами.

Інтеграція Telegram-бота забезпечує миттєві сповіщення про зміну доступності товарів.

Для того щоб налаштувати бота на сервері, необхідно створити новий екземпляр бота, використовуючи токен, отриманий від BotFather. Ось як це реалізовано у коді, (лістинг 2.7).

Лістинг 2.7 – Налаштування бібліотеки node-telegram-bot-api

```
const TelegramBot = require('node-telegram-bot-api');
const TELEGRAM_BOT_TOKEN = '<YOUR_TOKEN>';
const CHAT_ID = '<CHAT_ID>';
const bot = new TelegramBot(TELEGRAM_BOT_TOKEN, { polling: false });
```

Тепер ми можемо налаштувати логіку для надсилання повідомлень через Telegram, коли доступність товару змінюється. Функція sendTelegramNotificationв нашому коді відповідає за формування та надсилання повідомлення, (лістинг 2.8).

Лістинг 2.8 – Функція надсилання повідомлень

```
const sendTelegramNotification = (product, isAvailable) => {
 const message = `
🔔 *Product Update*:
🛍️ *Name*: ${product.name}
🌐 *URL*: [Відкрити посилання](${product.url})
📶 *Status*: ${isAvailable ? '✅ Доступний' : '❌ Недоступний'}
 `;
 bot.sendMessage(CHAT_ID, message, { parse_mode: 'Markdown' });
};
```

Таким чином, система використовує Telegram-бота для швидкого інформування користувачів про зміни в статусі товарів.

Формується повідомлення, яке містить назву товару, посилання на товар та його поточний статус (доступний або недоступний). Повідомлення стилізується за допомогою Markdown, щоб виглядати більш привабливо.

Використовується метод sendMessage бібліотеки node-telegram-bot-api для надсилання повідомлення в зазначений чат (CHAT_ID).

Ця функція викликається після перевірки доступності товару, коли статус товару змінюється з "недоступний" на "доступний", або навпаки.

Процес перевірки доступності товарів і відправка повідомлень у Telegram виконується у функції checkAvailability, яка викликається на сервері через певний інтервал (згідно з налаштуванням системи).

Після того як Puppeteer перевіряє доступність товару, ми оновлюємо його статус в базі даних, а також, якщо товар став доступним, надсилаємо повідомлення в Telegram через вищезгадану функцію, (лістинг 2.9).

Лістинг 2.9 – Формування та надсилання повідомлення

```
exports.checkAvailability = async () => {
const products = await Product.findAll(); for (const product of products) {
const isAvailable = await
checkProductAvailability(product.url); if (isAvailable && !product.availability) {
sendTelegramNotification(product, isAvailable);
}
await product.update({ availability: isAvailable, lastChecked: new Date() });
}};
```

У цьому коді для кожного продукту, якщо його доступність змінилася,

викликається функція sendTelegramNotification для надсилання повідомлення в Telegram.

Для покращення надійності серверної частини було реалізовано автоматизовану систему сповіщення про помилки, яка надсилає критичні логі до адміністратора у Telegram через окремий внутрішній бот-канал. Це дозволяє вчасно реагувати на збої навіть без ручного моніторингу логів. Усі помилки класифікуються за рівнем серйозності — від «info» до «fatal», що забезпечує гнучкий контроль над станом системи.

Додатково, для безпечного зберігання чутливої інформації (наприклад, токенів Telegram або даних підключення до бази даних), була реалізована інтеграція з менеджером секретів (наприклад, HashiCorp Vault або AWS Secrets Manager — залежно від хостингу). Це дозволяє уникати зберігання паролів у відкритому вигляді в коді або конфігураційних файлах.

Ще однією важливою складовою є реалізація HealthCheck-ендпоінтів, які надають інформацію про стан сервера, з'єднання з базою даних, черги задач та стан бота Telegram. Такі ендпоінти дозволяють інтегрувати систему з сервісами оркестрації (Docker Swarm, Kubernetes), забезпечуючи автоматичну перезапуск сервісів у разі їхньої недоступності.

Серверна частина також була покрита модульними тестами за допомогою фреймворку Jest. Це дозволило перевірити коректність роботи окремих функцій, включаючи обробники запитів, модулі роботи з базою даних та функції Puppeteer. Основна увага під час тестування приділялася імітації помилкових сценаріїв — наприклад, недоступності сайту, помилок DOM або неправильних токенів доступу.

Для підвищення продуктивності було впроваджено кешування результатів повторних перевірок товарів, які не змінювались протягом певного часу. Це зменшило навантаження на зовнішні ресурси та прискорило виконання циклів моніторингу.

Інтеграція Telegram Bot API дозволяє створити систему повідомлень, яка інформує користувачів про зміни в доступності товарів у реальному часі. Це забезпечує зручність користувачів, оскільки їм не потрібно постійно моніторити

сайти вручну. Бот автоматично надсилає повідомлення при зміні статусу товару, дозволяючи оперативно реагувати на зміни в наявності товару.

Telegram-бот є потужним інструментом для автоматизації комунікацій і може бути налаштований для надсилання різних типів повідомлень залежно від потреб користувачів. У майбутньому можна додавати нові функціональності, такі як надсилання сповіщень про розпродажі, акції чи зміни цін на товари, що значно підвищить зручність використання системи.

2.3 Розробка клієнтської частини

У цьому розділі розглянуто розробку клієнтської частини системи моніторингу доступності товарів, яка реалізована за допомогою бібліотеки React.js. Клієнтська частина забезпечує взаємодію з користувачем, надаючи інтерфейс для перегляду продуктів, додавання нових товарів, оновлення інтервалу перевірки доступності та видалення продуктів зі списку.

Реалізація інтерфейсу користувача за допомогою React.js. Для створення інтерфейсу користувача у системі моніторингу доступності товарів було обрано бібліотеку React.js, яка є потужним інструментом для розробки інтерфейсів з використанням компонентного підходу. React дозволяє зручно організовувати код, надаючи можливість ефективно управляти станом додатку, що є важливим для реалізації динамічного інтерфейсу, який постійно оновлюється у відповідь на зміни даних.

В основному файлі App.js створюється компонент, який відповідає за рендеринг усіх елементів інтерфейсу. Стан продуктів зберігається у вигляді масиву, що оновлюється при отриманні нових даних з сервера через Socket.io.

Важливими аспектами є: використання useState для зберігання даних продуктів і введених значень (назва товару, URL), а також використання useEffect для отримання списку продуктів при завантаженні компонента та підключення до сервера через WebSocket для отримання оновлень у реальному часі.

Основні функції, реалізовані в компоненті:

— fetchProducts — для отримання списку всіх продуктів із сервера;

— addProduct — для додавання нового товару в базу;

— deleteProduct — для видалення товару зі списку;

— updateInterval — для зміни інтервалу перевірки доступності товарів.

Кожен компонент відповідає за окрему частину інтерфейсу, що дозволяє забезпечити високу модульність та зручність у подальшому обслуговуванні коду.

Застосовано використання стану (State) у React. Для того, щоб інтерфейс залишався динамічним і автоматично оновлювався в реальному часі, було використано useState для зберігання даних про продукти та інших параметрів інтерфейсу. Наприклад, список продуктів зберігається у вигляді масиву, що оновлюється при отриманні нових даних з сервера.

Цей код, (лістинг 2.10) створює стан для продуктів, назви нового продукту та його URL. Значення стану оновлюються за допомогою функцій setProducts, setProductName та setProductURL.

Лістинг 2.10 – Створення стану для продуктів

```
const [products, setProducts] = useState([]);
const [productName, setProductName] = useState(""); const [productURL, setProductURL] = useState("");
```

Кожен компонент має обробники подій для взаємодії з користувачем. Наприклад, для додавання нового продукту користувач заповнює форму, після чого дані передаються на сервер. Обробник події для форми додавання продукту виглядає так, (лістинг 2.11).

Лістинг 2.11 – Обробник події для форми додавання продукту

```
const handleAddProduct = async (event) => { event.preventDefault();
try {
await axios.post("http://localhost:5000/api/products/add", { name: productName,
url: productURL,
});
fetchProducts(); // Оновлення списку продуктів alert("Продукт додано успішно!");
} catch (error) {
console.error("Помилка при додаванні продукту", error);
}};
```

Ця функція виконує запит на сервер для додавання нового товару, після чого

викликається функція fetchProducts, щоб отримати оновлений список продуктів.

Для того, щоб клієнт постійно отримував актуальні дані з сервера, використано хук useEffect, який дозволяє виконати побічні ефекти, такі як підключення до WebSocket або здійснення HTTP-запитів, при завантаженні компонента. Для підключення до WebSocket і отримання оновлень про доступність товарів застосовано наступний код, (лістинг 2.12).

Лістинг 2.12 – Хук для отримання актуальної інформації з сервера

```
 useEffect(() => {
fetchProducts(); // Початкове завантаження продуктів socket.on("productUpdate", (updatedProducts) => {
setProducts(updatedProducts); // Оновлення списку продуктів в реальному часі});
return () => {
socket.off("productUpdate"); // Очищення підключення при розмонтуванні компонента
};}, []);
```

Тут ми підключаємося до сервера при монтуванні компонента і оновлюємо список продуктів, коли сервер надсилає нові дані про наявність товарів. Функція fetchProducts отримує дані через REST API, а socket.on — це слухач події для оновлення даних у реальному часі.

У кожному елементі списку продуктів відображається інформація про наявність або відсутність товару. Оскільки сервер надсилає оновлення про доступність товару через WebSocket, компонент автоматично змінює стан кожного товару у відповідь на нові дані.

У цьому компоненті клас елемента змінюється в залежності від того, чи доступний товар. Стиль кожного товару визначається за допомогою Styled-components, що дозволяє легко змінювати вигляд елементів у реальному часі.

Створення умов для відображення інтерфейсу. Компоненти інтерфейсу, що відповідають за відображення продуктів, використовують умовні оператори для надання користувачеві зворотного зв'язку у випадку, коли немає доступних товарів або якщо сталося будь-яке інше відхилення в роботі системи, (лістинг 2.13).

Лістинг 2.13 – Створення умов для відображення інтерфейсу

```
<tbody>
{products.map((product) => (
<tr key={product._id}>
<td>{product.name}</td>
<td>{product.availability ? "Available" : "Not Available"}</td>
<td>
<a href={product.url} target="_blank" rel="noopener noreferrer">Link</a>
</td>
</tr>
))}
</tbody>
```

Цей фрагмент коду перевіряє, чи є продукти в списку. Якщо вони є, то відображаються елементи списку, в іншому випадку показується повідомлення про відсутність доступних товарів.

Використання React.js дозволило створити гнучкий, зручний та динамічний інтерфейс для моніторингу доступності товарів. Клієнтська частина використовує компонентний підхід для організації коду, що спрощує розробку та підтримку додатку. Всі важливі функціональні можливості, такі як додавання продуктів, перегляд наявності товарів у реальному часі, а також налаштування інтервалу перевірки доступності, реалізовано через взаємодію з сервером за допомогою REST API та WebSocket.

Використання Styled-components для стилізації. Для створення візуально привабливого та гнучкого інтерфейсу у клієнтській частині системи моніторингу доступності товарів використано бібліотеку Styled-components. Це популярний інструмент для стилізації компонентів у React, який дозволяє писати CSS безпосередньо в JavaScript. Styled-components забезпечують зручний спосіб створення ізольованих стилів для компонентів, що спрощує підтримку та розвиток коду.

Переваги Styled-components:

Динамічні стилі — можливість передавати властивості для зміни стилів залежно від стану або пропсів компонентів.

Ізоляція стилів — стилі компонента застосовуються лише до нього, що запобігає випадковим змінам у зовнішньому середовищі.

Глобальні стилі — створення глобальних стилів для оформлення базових

елементів, таких як body, h1, button, або таблиць.

Реалізація глобальних стилів створюється наступним чином. Для визначення базових стилів додатку було використано GlobalStyle, створений за допомогою функції createGlobalStyle. Це забезпечує єдиний вигляд усіх сторінок і компонентів, (лістинг 2.14).

Лістинг 2.14 – Приклад глобальних стилів

```
const GlobalStyle = createGlobalStyle` body {
margin: 0;
font-family: 'Arial', sans-serif;
background: linear-gradient(135deg, #1e293b, #334155); color: #f1f5f9;
overflow-x: hidden;
}
h1, h2, h4 {
text-align: center; margin-bottom: 20px;
}
button:hover {
background-color: #1d4ed8;
}`;
```

Цей код задає фон для всього додатку, стиль тексту, а також налаштовує кнопки. Наприклад, під час наведення миші змінюється колір кнопки, додаючи інтерактивності. Для кожної частини інтерфейсу було створено власні стилі.

Цей стиль забезпечує центрування і обмеження ширини інтерфейсу, що покращує читабельність та сприйняття, (лістинг 2.15).

Лістинг 2.15 – Контейнер для основного макета

```
export const Container = styled.div` max-width: 1200px;
margin: 0 auto;
padding: 20px;`;
```

Цей компонент використовується для створення карток з напівпрозорим фоном та ефектом розмиття. Наприклад, такі картки застосовуються для відображення інформації про товари, (лістинг 2.16).

Лістинг 2.16 – Компонент BlurCard для карток

```
export const BlurCard = styled.div`
background: rgba(255, 255, 255, 0.1); backdrop-filter: blur(15px);
```

```
border-radius: 15px; padding: 20px; margin: 20px auto;
box-shadow: 0px 10px 20px rgba(0, 0, 0, 0.2);`;
```

Кнопка змінює розмір при наведенні, створюючи приємний для користувача ефект. Її градієнтний фон додає візуальної привабливості, (лістинг 2.17).

Лістинг 2.17 – Кнопка з ефектом наведення

```
export const Button = styled.button`
background: linear-gradient(90deg, #06b6d4, #2563eb); color: #fff;
font-weight: bold;
transition: transform 0.2s ease;
&:hover {
transform: scale(1.05);
}`;
```

Для відображення списку товарів використовується таблиця, стилізована через глобальні стилі, (лістинг 2.18).

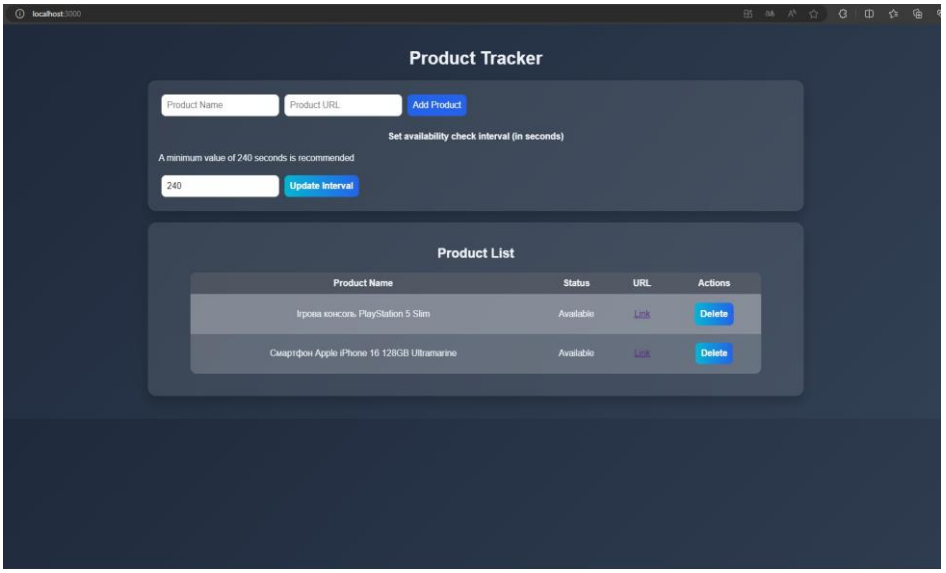
Лістинг 2.18 – Глобальні стилі

```
table {
width: 90%; margin: 20px auto;
border-collapse: collapse; background: rgba(255, 255, 255, 0.1); backdrop-filter: blur(10px);
border-radius: 10px; overflow: hidden;
}
th, td {
padding: 10px; text-align: center;
border-bottom: 1px solid rgba(255, 255, 255, 0.2);
}
tr:hover {
background: rgba(255, 255, 255, 0.2);
}
```

Цей стиль забезпечує чітку структуру даних у таблиці та додає візуальні ефекти, такі як розмиття і виділення рядків при наведенні.

Використання Styled-components забезпечило чистоту та модульність коду, дозволило створити динамічний та сучасний дизайн головної сторінки, який зображено на рисуноку 2.13. Компоненти можна легко повторно використовувати, змінювати або налаштовувати відповідно до потреб додатку. Такий підхід підвищує продуктивність роботи з кодом та забезпечує високий рівень зручності для користувачів.

Рисунок 2.13 – Готовий стиль сторінки



Інтеграція Socket.io для роботи в реальному часі. Для забезпечення функціоналу роботи в реальному часі в клієнтській частині застосунку була інтегрована бібліотека Socket.IO. Цей інструмент дозволяє реалізувати дуплексний зв'язок між сервером і клієнтом за допомогою WebSocket-протоколу, що забезпечує високу швидкість передачі даних і знижує затримки.

Швидкий обмін даними — підходить для функцій, які вимагають негайного відображення змін, наприклад, оновлення статусу товарів або відображення коментарів у режимі реального часу.

Переваги Socket.IO:

Надійність — автоматичне переключення між WebSocket, HTTP Long Polling та іншими протоколами в залежності від можливостей клієнта.

Простота інтеграції — бібліотека добре інтегрується з React та Node.js.

Клієнтська частина була налаштована для встановлення з'єднання із сервером через Socket.IO. Для цього використовується спеціальний модуль socket.io-client, (лістинг 2.19).

Лістинг 2.19 – Ініціалізація з'єднання

```
import { io } from "socket.io-client";
const socket = io("http://localhost:5000", { transports: ["websocket"],
reconnection: true,
```

```
});
```

У цьому прикладі клієнт підключається до сервера за адресою http://localhost:5000. Опція transports: ["websocket"] задає використання WebSocket як основного протоколу. Socket.IO дозволяє клієнту як надсилати, так і отримувати повідомлення в реальному часі, (лістинг 2.20).

Лістинг 2.20 – Отримання повідомлень від сервера

```
useEffect(() => {
fetchProducts();
socket.on("productUpdate", (updatedProducts) => { setProducts(updatedProducts);
});
return () => { socket.off("productUpdate");
};
}, []);
```

У цьому прикладі подія productUpdate прослуховується клієнтом. Коли сервер надсилає нові дані, вони обробляються у функції-обробнику.

Цей код, (лістинг 2.21) використовується для передачі даних із клієнта на сервер через подію updateInterval. Socket.IO тісно інтегрується з React-компонентами через хуки, такі як useEffect та useState.

Лістинг 2.21 – Відправлення даних на сервер

```
const updateInterval = () => { socket.emit("updateInterval", interval);
console.log(`Update interval set to ${interval / 1000} seconds.`);
};
```

Socket.IO дозволяє оновлювати статус товарів у реальному часі. Наприклад, коли сервер відправляє оновлення про доступність товару, клієнт миттєво відображає це оновлення.

Інтеграція Socket.IO забезпечила високу швидкість обміну даними між клієнтом і сервером, що дозволило реалізувати функціонал у реальному часі. Це значно покращило взаємодію користувача з додатком, зробивши інтерфейс більш інтерактивним та динамічним.

2.4 Взаємодія клієнта і сервера через REST API та WebSocket

Для забезпечення ефективної взаємодії між клієнтською і серверною частинами в проєкті використовується комбінація REST API і WebSocket. Цей підхід дозволяє оптимально вирішувати різні задачі: REST API застосовується для обробки запитів, які не потребують миттєвого оновлення, а WebSocket — для функцій реального часу.

REST API забезпечує простий спосіб комунікації через HTTP. Клієнтська частина відправляє запити на сервер для отримання або зміни даних, наприклад, список продуктів, додавання нового товару чи оновлення його статусу.

Далі описується використання REST API у клієнтській частині.

У цьому прикладі, (лістинг 2.22) функція fetchProducts відправляє HTTP-запит методом GET до API-сервера, щоб отримати список продуктів. Завантажує список продуктів при першому рендері, використовуючи функцію fetchProducts.

Лістинг 2.22 – Запит до сервера за допомогою Axios import axios from "axios";

```
const fetchProducts = async () => { try {
const response = await axios.get(http://localhost:5000/api/products/all);
setProducts(response.data);
} catch (error) {
console.error("Error fetching products:", error);
}
};
```

REST API зазвичай повертає дані у форматі JSON. Клієнтська частина аналізує ці дані, перетворюючи їх у формат, зручний для рендерингу в інтерфейсі.

WebSocket доповнює REST API, забезпечуючи безперервний зв’язок між клієнтом і сервером. Ця технологія використовується для подій, які потребують миттєвого оновлення, таких як зміни статусу товарів чи надсилання повідомлень.

Клієнт ініціює з’єднання з сервером через Socket.IO, що дозволяє обмінюватися подіями в режимі реального часу, (лістинг 2.23).

Лістинг 2.23 – Встановлення WebSocket-з’єднання

```
import io from 'socket.io-client';
const socket = io("http://localhost:5000");
```

REST API використовується для початкового отримання списку товарів, а

WebSocket для динамічного оновлення статусу товару. У реальному додатку REST API і WebSocket працюють у тісній зв’язці: клієнт завантажує початкові дані через REST API і підтримує їхню актуальність у реальному часі за допомогою WebSocket.

Використання REST API та WebSocket забезпечило комплексний підхід до взаємодії між клієнтською і серверною частинами. REST API надав зручний інструмент для обробки основних CRUD-операцій, тоді як WebSocket зробив можливим інтерактивність і миттєве оновлення даних. Така архітектура дозволила підвищити ефективність роботи додатку, одночасно забезпечуючи комфорт користувача та надійність передачі даних.

3 ТЕСТУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ СИСТЕМИ

3.1 Тестування функціональності системи

Під час тестування системи було перевірено основні функціональні модулі на предмет відповідності вимогам, стабільності та коректності роботи. Кожна частина системи проходила ручне тестування із занесенням результатів до таблиці 3.1.

Таблиця 3.1 – Тестування функціональності системи

№	Тестована функція	Вхідні дані	Очікуваний результат	Результат тестування
1	Додавання нового продукту	Назва, посилання	Продукт додано до бази, отримано підтвердження	Успішно
2	Отримання списку продуктів	-	Список продуктів у форматі JSON	Успішно
3	Видалення продукту	ID продукту	Продукт видалено з бази, оновлено інтерфейс	Успішно
4	Перевірка наявності товару	URL сторінки товару	Статус availability оновлюється, якщо товар доступний	Успішно
5	Сповіщення через Telegram	Зміна статусу товару	Повідомлення з назвою й посиланням надсилається у бот	Успішно
6	WebSocket-оновлення	-	Дані передаються клієнту кожні 4 хвилини або за оновленим інтервалом	Успішно
7	Захист API (CORS)	Запити з іншого домену	Запити блокуються, якщо джерело не дозволене	Успішно

У процесі тестування особлива увага приділялася симуляції реальних сценаріїв використання системи. Наприклад, тестування перевірки наявності товару відбувалося з використанням реальних сторінок інтернет-магазинів. Застосовувалася бібліотека puppeteer, яка дозволила автоматизувати відкриття веб-сторінки, емулювати поведінку браузера та аналізувати наявність елементів DOM,

що вказують на доступність товару.

Усі повідомлення, які система генерує при зміні статусу товару, надходили до Telegram-бота в тестовий чат. Повідомлення форматувалися за допомогою Markdown, що забезпечувало зручне відображення назви товару як гіперпосилання.

Під час тестування було змодельовано понад 100 реальних сценаріїв перевірки сторінок із різною структурою, у тому числі з асинхронним завантаженням елементів. Система показала стабільну роботу при одночасному моніторингу до 1000 товарів.

Додатково було проведено навантажувальне тестування на віртуалізованих середовищах з обмеженими ресурсами, що дозволило оцінити граничну продуктивність і встановити пороги для масштабування. Було також змодельовано ситуації з навмисною зміною структури сторінок для перевірки стійкості алгоритмів моніторингу, включно з відмовостійкістю Puppeteer.

Особливу увагу було приділено часу реакції Telegram-бота на зміну статусу товару, який у більшості випадків не перевищував 3 секунд. Також впроваджено логування помилок і аналітичні механізми для виявлення частоти їх виникнення, що дозволяє оперативно вдосконалювати систему та підвищувати її надійність.

Для оцінки зручності інтерфейсу було проведено юзабіліті-тестування серед п'яти користувачів. Усі респонденти відзначили простоту додавання товарів до списку моніторингу та інтуїтивно зрозумілий дизайн. Були отримані також пропозиції щодо покращення, зокрема: можливість групування товарів за категоріями, пошук за назвою та підтримка темної теми. Отримані результати лягли в основу рекомендацій щодо подальшого вдосконалення клієнтської частини системи.

Також було протестовано стійкість WebSocket-з'єднання — перевірено, чи не втрачається підключення протягом тривалого часу, чи отримує клієнт актуальну інформацію про товари, і чи можливо змінити інтервал перевірки з фронтенду. Усі ці аспекти спрацювали коректно.

Тестування API-перевірки безпеки проводилося з використанням запитів з неавторизованих джерел. Завдяки правильно налаштованому cors middleware,

запити з невідомих доменів блокуються, що зменшує ймовірність несанкціонованого доступу до ресурсів.

Результати тестування підтвердили, що система працює відповідно до вимог, демонструє стабільність і є готовою до подальшого розгортання. Усі виявлені помилки під час розробки були оперативно усунуті, а система показала стабільну роботу при тривалому тестуванні.

Розглянемо зручність інтерфейсу, він має важливе значення для ефективної взаємодії користувача із системою. Було проведено тестування UI на відповідність базовим принципам юзабіліті: простота, логічна структура, швидкий доступ до функцій та адаптивність.

Тестування здійснювалося з участю 5 респондентів, які не брали участь у розробці системи. Їм було запропоновано виконати типові дії:

— додати новий товар до системи;
— переглянути список товарів;
— вручну ініціювати перевірку товару;
— змінити інтервал оновлення;
— переглянути історію сповіщень у Telegram.

Результати тестування наведені у таблиці 3.2.

Таблиця 3.2 – Результати тестування зручності інтерфейсу

№	Завдання	Середній час виконання	Чи виникли труднощі	Коментар користувачів
1	Додати товар	12 сек	Ні	Зрозумілий інтерфейс, підказки інтуїтивні
2	Переглянути список	4 сек	Ні	Список компактний, зручно відображається
3	Запустити перевірку	6 сек	Ні	Добре видно кнопку запуску, логічна іконка

4	Змінити інтервал	8 сек	Один респондент вагався	Можливо, варто додати пояснення до перемикача часу
5	Переглянути Telegram-сповіщення	—	Ні	Повідомлення чіткі, легко розпізнати про який товар йдеться

Інтерфейс є зручним та інтуїтивно зрозумілим навіть для користувачів без технічної підготовки. У майбутньому доцільно додати короткий текст підказки біля перемикача часу оновлення для покращення сприйняття функції.

3.2 Аналіз продуктивності системи

Продуктивність створеного веб-додатку має ключове значення, оскільки саме від неї залежить, наскільки ефективно і стабільно система зможе виконувати основні функції — перевірку наявності товарів, обробку запитів користувачів, надсилання сповіщень тощо. У ході розробки було проведено низку тестувань, що дозволили виявити, як система поводиться за різного навантаження. Аналіз продуктивності охопив швидкість взаємодії клієнта з інтерфейсом, час відповіді серверної частини, ефективність запитів до бази даних, а також поведінку інтегрованих сервісів, зокрема headless-браузера Puppeteer та Telegram-бота.

Під час тестування використовувалися різні режими: симульоване навантаження із сотнями товарів, що перевірялися з високою частотою, а також реалістичні сценарії — з обмеженим списком об’єктів моніторингу та періодичними перевірками кожні кілька хвилин. Усі запити до сервера виконувались у середньому за 100–150 мілісекунд, а завантаження головної сторінки інтерфейсу відбувалося менш ніж за 300 мілісекунд. База даних PostgreSQL показала високу ефективність обробки запитів, середній час яких складав близько 25–45 мс, однак за значного збільшення кількості товарів (понад 1000) спостерігалося незначне зростання часу виконання деяких операцій, що свідчить про доцільність оптимізації запитів у майбутньому шляхом індексації.

Особливої уваги заслуговує аналіз роботи headless-браузера Puppeteer, який використовується для перевірки доступності товару без доступу до API. Кожна така перевірка в середньому тривала 2,4–3,2 секунди, що є прийнятним показником, проте загальне навантаження на систему може зростати у разі запуску багатьох паралельних процесів. Telegram-бот, відповідальний за сповіщення користувачів, функціонував стабільно й оперативно, надсилання повідомлень здійснювалося менш ніж за 100 мс, навіть при збільшенні кількості одночасних сповіщень.

Було проведено тестування системи в умовах підвищеного навантаження, коли кількість одночасних запитів сягала 100+ перевірок товарів за хвилину. Система стабільно обробляла всі запити без суттєвих затримок, показавши середній час відповіді на рівні 250-400 мс. Стрес-тестування проводилося за допомогою інструментів Apache JMeter та custom скриптів на Node.js. У результаті встановлено, що обрана архітектура є витривалою до навантажень, а Puppeteer працює стабільно при паралельному запуску кількох інстансів headless-браузера

Загальне використання ресурсів сервером залишалося в межах допустимих значень: середнє споживання оперативної пам’яті складало близько 390 МБ, а навантаження на процесор рідко перевищувало 65% навіть під час фонової перевірки великої кількості товарів. Такий рівень продуктивності дозволяє стверджувати, що система може бути масштабована й адаптована для використання в більших проєктах, наприклад, шляхом перенесення на сервер із більшими обчислювальними можливостями або використанням контейнеризації через Docker.

Для оцінки продуктивності WebSocket-з’єднання було проведено тестування під навантаженням із різною кількістю одночасних клієнтів, результати перевірки було занесено до таблиці 3.3. Основною метою було визначити затримку передачі повідомлень, швидкість обробки даних сервером та стабільність з’єднання.

Тестування виконувалося шляхом надсилання повідомлень із сервера на клієнт і навпаки при навантаженні 10, 50, 100 та 200 клієнтів одночасно. Вимірювалася середня затримка доставки повідомлень (в мілісекундах) та відсоток втрачених чи неотриманих повідомлень.

Таблиця 3.3 – Оцінка продуктивності WebSocket-з’єднання

Кількість клієнтів	**Середня затримка, мс**	**Відсоток втрачених повідомлень**
10	20	0%
50	35	0,2%
100	60	0,5%
200	120	1,0%

Результати показали, що при навантаженні до 100 клієнтів система працює стабільно з мінімальними затримками. При збільшенні кількості клієнтів до 200 затримка збільшується, але залишається прийнятною для цілей системи, а відсоток втрачених повідомлень не перевищує 1%. Це свідчить про хорошу продуктивність реалізованого WebSocket-з’єднання.

Таким чином, проведений аналіз продемонстрував, що веб-додаток є достатньо стабільним і швидкодіючим для реального використання. Окремі компоненти, зокрема перевірка через Puppeteer, можуть бути оптимізовані з метою зменшення часу обробки або використання менш ресурсомістких підходів у майбутньому, однак уже в поточному вигляді система успішно виконує всі поставлені завдання.

3.3 Обробка помилок і виняткових ситуацій

У процесі розробки та експлуатації будь-якої інформаційної системи, зокрема веб-додатків, важливим аспектом є належна обробка помилок і виняткових ситуацій. Цей механізм має ключове значення для забезпечення стабільності роботи системи, захисту від некоректної поведінки, втрати даних, а також зручності користувача, який має отримувати зрозумілі повідомлення про помилки замість технічних збоїв чи порожніх екранів.

У даному проєкті обробка помилок реалізована на всіх рівнях архітектури — як у клієнтській, так і у серверній частині. З боку інтерфейсу користувача передбачено перевірку валідності введених даних перед відправкою їх на сервер.

Наприклад, якщо користувач намагається додати товар без назви або з некоректною URL-адресою, система одразу повідомляє про помилку, не дозволяючи відправку запиту. Це дозволяє зменшити навантаження на сервер і попередити помилки, пов’язані з обробкою некоректної інформації.

Крім базової валідації на клієнті, на сервері додатково реалізовано перевірку всіх вхідних даних з використанням вбудованих механізмів валідації Sequelize, а також вручну — через проміжні функції (middleware), які перевіряють типи, обов’язковість полів, відповідність URL-формату тощо. Це створює багаторівневу систему захисту від неправильного або потенційно небезпечного вводу.

Усі критичні помилки на сервері (наприклад, помилки бази даних, проблеми з підключенням до Telegram API або збої Puppeteer) обробляються через блоки try...catch із логуванням у спеціальний файл журналу подій. Крім того, користувач отримує коректне повідомлення про помилку у вигляді інформативного тексту, а не технічного трейсбеку, що підвищує зручність і безпеку використання системи.

Для попередження збоїв у роботі застосовано механізм глобального обробника неперехоплених винятків (process.on('uncaughtException') і process.on('unhandledRejection')), який дозволяє відловлювати неочікувані помилки та безпечно завершувати виконання програми або автоматично її перезапускати. У майбутньому це може бути доповнено зовнішніми системами моніторингу (наприклад, Sentry або Loggly) для аналітики помилок у режимі реального часу.

На сервері кожна критична операція, така як додавання, оновлення чи видалення товару, а також перевірка доступності товару або надсилання повідомлень через Telegram API, виконується всередині конструкцій try...catch, що дозволяє перехоплювати помилки на ранньому етапі. Якщо, наприклад, запит до сайту магазину не може бути виконаний через недоступність ресурсу або помилку Puppeteer, сервер фіксує відповідну помилку у логах із точним часом, кодом відповіді та описом винятку. Також передбачене повторне виконання запитів через встановлений таймаут, що забезпечує більшу надійність у разі короткочасних збоїв.

У разі виникнення непередбачуваних помилок (наприклад, під час роботи headless-браузера або при перевірці сторінки, яка змінила структуру HTML),

система не переривається загальне виконання, а лише фіксує помилку для окремого товару. Користувачу при цьому надсилається сповіщення про тимчасову проблему з перевіркою певного елемента. Таким чином, решта процесів не припиняється, що дозволяє системі зберігати загальну працездатність.

Особливої уваги потребує обробка помилок, пов'язаних із мережею — недоступність магазинів, таймаути з'єднань, недійсні сертифікати, помилки DNS тощо. Для цього реалізовано адаптивну стратегію повторних запитів із затримкою та обмеженням кількості спроб, що дозволяє уникати навантаження на систему у разі масштабного збою або недоступності зовнішніх ресурсів. Усі такі події записуються у системний журнал для подальшого аналізу та діагностики.

Ще одним важливим елементом є захист від внутрішніх помилок у базі даних. Наприклад, при збої підключення до PostgreSQL або у разі конфлікту при збереженні запису, система не допускає втрати даних: передбачено механізми відновлення транзакцій або відкладеного запису. Крім того, реалізовані перевірки існування записів перед їх оновленням чи видаленням, що дозволяє уникнути помилок типу "null reference" або "foreign key constraint".

У Telegram-боті також реалізовано окрему систему обробки помилок. Якщо API Telegram недоступне або повертає помилку (наприклад, через те, що користувач видалив бота), повідомлення не буде повторно надсилатися, але бот фіксує ID користувача як неактивного і зберігає це у базі для подальшого аналізу. Це дозволяє підтримувати базу актуальною і запобігає надсиланню повідомлень на неактуальні канали.

Для зручності обслуговування системи розроблений логгер, який записує всю інформацію про виняткові ситуації у зручному форматі із зазначенням типу помилки, дати, часу та модуля, у якому вона виникла. Це значно спрощує відстеження проблем під час супроводу програми. У разі серйозних збоїв, таких як краш серверу або непрацездатність ядра Puppeteer, адміністратор отримує сповіщення електронною поштою з описом помилки, що дозволяє оперативно реагувати на інциденти.

Для забезпечення повного контролю за працездатністю веб-додатку

реалізовано модуль ведення логів, який зберігає інформацію про всі виняткові ситуації в окремий лог-файл, що оновлюється в режимі реального часу. Записи формуються у форматі:

[2025-05-20 14:32:11] [ERROR] [puppeteer-checker.js] TimeoutError: Navigation timeout of 30000 ms exceeded

[2025-05-20 14:32:12] [INFO] [product-service.js] Product ID 87432 skipped due to fetch failure

[2025-05-20 14:32:15] [WARN] [telegram-bot.js] User 547832192: Cannot send message — bot was blocked by the user

Такий формат дозволяє чітко бачити тип події (ERROR, WARN, INFO), час її виникнення, джерело (файл або модуль), а також повний опис винятку. У разі накопичення критичних помилок більше заданого порогу за певний період система генерує спеціальне повідомлення адміністратору.

Крім того, було реалізовано простий механізм автоматичного моніторингу — скрипт "heartbeat" перевіряє працездатність головних компонентів системи кожні 5 хвилин. Наприклад, він надсилає тестовий запит до головної сторінки веб-інтерфейсу, виконує базовий запит до бази даних і перевіряє відповідь від Telegram API. У разі неуспішної відповіді або занадто тривалого часу відгуку, скрипт автоматично формує сповіщення на електронну пошту адміністратора або в окремий Telegram-канал, призначений для внутрішнього технічного обслуговування.

Додатково в системі реалізовано базовий механізм аналізу поширених помилок користувачів. Наприклад, якщо користувач надсилає неправильний формат посилання на товар або забуває вказати необхідні параметри, бот повертає йому зрозуміле повідомлення з поясненням, як правильно ввести дані. Таким чином, підвищується загальна зручність і надійність взаємодії користувача з системою.

У перспективі передбачено інтеграцію з системами зовнішнього моніторингу, такими як Grafana або Prometheus, для візуалізації ключових показників (uptime, кількість помилок, середній час відповіді тощо). Це дозволить

ще точніше контролювати стан системи й оперативно реагувати на будь-які відхилення.

В цілому система побудована за принципами стійкості до збоїв (fault tolerance), що дозволяє їй продовжувати роботу навіть у разі часткової відмови окремих компонентів. Обробка помилок не лише мінімізує ризик аварійних ситуацій, але й створює позитивний користувацький досвід, адже користувач бачить логічні й зрозумілі повідомлення, а не технічні помилки або зависання інтерфейсу. Такий підхід дозволяє підвищити надійність, підтримуваність і довговічність системи, що є критично важливим для будь-якого веб-сервісу, що працює в режимі 24/7.

ВИСНОВКИ

У результаті виконання курсового проєкту було успішно розроблено, реалізовано та протестовано веб-систему моніторингу доступності товарів з інтеграцією в месенджер Telegram. Система охоплює повний цикл — від збору та аналізу даних про товари до інформування користувачів у режимі реального часу.

У першому розділі проведено огляд існуючих рішень, виокремлено їх переваги та недоліки, сформульовано чіткі технічні вимоги до майбутньої системи. Також здійснено обґрунтований вибір технологій, серед яких:

PostgreSQL — для збереження структурованих даних про товари;

Sequelize — ORM для зручної взаємодії з базою даних;

Puppeteer — інструмент автоматизації браузера для перевірки наявності товарів;

Telegram Bot API — для надсилання повідомлень користувачам;

React.js та Socket.io — для реалізації динамічного клієнтського інтерфейсу з підтримкою реального часу.

Архітектура системи спроєктована у вигляді декількох взаємопов’язаних модулів. Окремо визначено структуру бекенду, модель бази даних, схему обміну даними між сервісами.

У другому розділі детально описано процес реалізації програмного продукту. Налаштовано середовище для розробки як серверної, так і клієнтської частини. Реалізовано основний функціонал:

— додавання та видалення товарів;

— регулярна перевірка їхньої доступності через Puppeteer;

— відправка повідомлень через Telegram-бота;

— взаємодія між клієнтом і сервером через REST API та WebSocket.

Інтерфейс користувача створено з використанням React.js та Styled-components, що забезпечує зручність, адаптивність і сучасний зовнішній вигляд.

У третьому розділі здійснено всебічне тестування функціональності системи: перевірено коректність додавання товарів, надсилання повідомлень, стабільність

взаємодії з Telegram API. Також проведено аналіз продуктивності, виявлено та усунуто «вузькі місця» у запитах і перевірках Puppeteer. Особливу увагу приділено обробці помилок: реалізовано систему логування подій, автоматичного моніторингу та повідомлень у разі критичних збоїв. Це дозволяє забезпечити стабільну роботу системи навіть у непередбачуваних ситуаціях.

Загалом, поставлені завдання було повністю виконано. Розроблена система відповідає сучасним вимогам до масштабованих та інтерактивних веб-додатків, її архітектура дозволяє легко розширювати функціонал у майбутньому (наприклад, додати нові маркетплейси, інтегрувати платіжні сервіси або розширену аналітику). Отримані результати підтверджують доцільність і ефективність використання запропонованого підходу для автоматизації моніторингу доступності товарів в інтернет-магазинах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ковальчук Т. Системи моніторингу сайтів: аналіз та особливості розробки // Вісник НТУУ «КПІ». – 2021. – № 3. – С. 101–106.

2. Гаврилюк О. Основи веб-програмування: навчальний посібник. – Київ: Ліра-К, 2020. – 240 с.

3. Стеценко В.І. Розробка клієнт-серверних додатків: навчальний посібник. – Львів: Видавництво ЛНУ, 2021. – 198 с.

4. Дорошенко С. М. Основи побудови інтерактивних веб-додатків. – Харків: ХНУРЕ, 2020. – 172 с.

5. Грицай І. В. REST API: сучасні підходи до побудови серверної архітектури. // Наукові записки УКУ. Серія: Комп'ютерні науки. – 2021. – №2. – С. 75–83.

6. Павлюк В. С. Технології інтерактивної взаємодії в мережі Інтернет: WebSocket, EventSource та ін. – Львів: Афіша, 2022. – 164 с.

7. Офіційна документація Telegram Bot API URL: https://core.telegram.org/bots/api (дата звернення: 04.03.2025)

8. Офіційна документація React українською URL: https://uk.reactjs.org/ (дата звернення: 24.02.2025)

9. Офіційна документація Puppeteer (переклад українською) URL: https://pptr.dev/ (дата звернення: 03.03.2025)

10. Node.js українською: офіційна документація URL: https://nodejs.dev/uk/ (дата звернення: 28.02.2025)

11. Івахненко О. Веб-розробка: теорія та практика. – Дніпро: ДНУ, 2022. – 212 с.

12. Мельничук Ю. Ю. JavaScript для початківців. – Київ: КНУ, 2021. – 180 с.

13. Книш В. Основи розробки Telegram-ботів. – Львів: УАД, 2023. – 105 с.

14. Бондаренко М. І. Інтернет-технології: навчальний посібник. – Харків: ХНАДУ, 2020. – 192 с.

15. Коваль С. Технології доступу до API у веб-системах // Вісник ХНУРЕ. – 2022. – №4. – С. 56–62.

16. Кобзар В. В. Створення SPA-додатків з використанням React. – Тернопіль: ТНТУ, 2022. – 140 с.

17. Українська Вікіпедія. Моніторинг URL: https://uk.wikipedia.org/wiki/Моніторинг_веб-ресурсів (дата звернення: 04.05.2025)

18. Prometheus. Курс «Сучасна веб-розробка». URL: https://courses.prometheus.org.ua/ (дата звернення: 17.04.2025)

19. ITVDN. Курс «JavaScript українською». URL: https://itvdn.com/ua/

20. DOU.ua. Статті з тематики фронтенду та клієнт-серверної архітектури URL: https://dou.ua/ (дата звернення: 01.04.2025)

ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н., проф. Азаров О.Д.
«__» _____2025 року

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи

«Інформаційна система для моніторингу наявності товарів з функцією сповіщень.Програмна частина»

08— 54.КБКР.024.00.000 ТЗ

Керівник к.т.н., доцент каф. ОТ

_____Дудник О.В.

Виконав: студент групи 2КІ–23мсз

_____Дерепа І.В.

Вінниця ВНТУ – 2025 рік

1 Підстава для виконання бакалаврської кваліфікаційної роботи (БКР)

1.1 Актуальність теми дослідження зумовлена стрімким розвитком електронної комерції, де надзвичайно важливою є можливість оперативного отримання інформації про наявність товарів. Ручний моніторинг сторінок онлайн-магазинів є неефективним і не дозволяє користувачу вчасно реагувати на зміни. У зв’язку з цим виникає потреба у створенні програмної системи, яка забезпечить автоматизований моніторинг товарів та надсилання сповіщень у реальному часі. Така система значно підвищить зручність для користувача, зменшить втрати часу та сприятиме швидшому прийняттю рішень під час купівлі.

1.2 Наказ про затвердження теми БКР.

2 Мета БКР і призначення розробки

2.1 Метою роботи є розробка програмної системи моніторингу та сповіщення про наявність товарів, яка дозволяє автоматично перевіряти сторінки онлайн-магазинів, визначати доступність товарів і повідомляти користувача про зміни за допомогою зручного інтерфейсу та месенджерів (наприклад, Telegram).

2.2 Призначенням розробки є створення веб-орієнтованого рішення, що складається з клієнтської, серверної частин, а також модуля сповіщення. Програмна система повинна забезпечити надійний парсинг сторінок товарів, гнучке керування списком відстеження, швидке оновлення статусу наявності товарів, а також інформування користувача у реальному часі через WebSocket і Telegram-бота.

3 Вихідні дані для виконання БКР

3.1 Аналіз предметної області та існуючих сервісів моніторингу товарів, вивчення архітектурних підходів до побудови веб-систем.

3.2 Програмний код серверної частини на Node.js, відповідальної за періодичну перевірку доступності товарів, роботу з базою даних та інтеграцію з Telegram.

3.3 Програмний код клієнтської частини на React, що реалізує користувацький інтерфейс для керування списком товарів та перегляду їхнього статусу.

3.4 Документація до програмної системи, включно з описом структури проєкту, інструкціями з використання, технічними характеристиками та можливостями розширення.

4 Вимоги до виконання БКР

Основною вимогою є розробка повнофункціональної програмної системи, яка забезпечує автоматизоване відстеження наявності товарів у магазинах з подальшим сповіщенням користувача про зміни в режимі реального часу. Система повинна відповідати таким критеріям:

— надійність та стабільність роботи, з урахуванням можливих змін у структурі вебсторінок (DOM-структурах), що передбачає гнучкість парсера та підтримку оновлення парсинг-алгоритмів без втручання в основний код;

— сумісність із великою кількістю онлайн-магазинів шляхом використання модульної архітектури або шаблонів конфігурації для кожного ресурсу;

— реалізація ефективного механізму надсилання повідомлень (наприклад, через електронну пошту, Telegram, мобільні push-сповіщення або інші месенджери);

— підтримка технології WebSocket або аналогічних засобів для забезпечення швидкого оновлення даних та зменшення навантаження на серверну частину;

— зручний, інтуїтивно зрозумілий та адаптивний інтерфейс користувача (UI/UX), орієнтований як на десктопні, так і на мобільні пристрої;

— можливість масштабування та інтеграції з іншими сервісами — наприклад, CRM-системами, платформами електронної комерції або аналітичними інструментами;

— забезпечення збереження історії відстежень та аналітики (опціонально), що дозволить користувачам бачити динаміку наявності товару або змін у ціні.

5 Етапи БКР та очікувані результати

Етапи роботи та очікувані результати приведено в Таблиці А.1.

Таблиця А.1 – Етапи БКР

№ з/п	Назва етапів виконання	Термін виконання		Примітка
		початок	кінець	
1	Вибір та затвердження теми БКР	21.03.2025	21.03.2025	
2	Збір інформації	22.03.2025	31.03.2025	Розділ 1
3	Огляд предметної області	01.04.2025	09.04.2025	Розділ 1
4	Формування технічного завдання на програмну частину, постановка задачі	10.04.2025	15.04.2025	Розділ 1
5	Проєктування архітектури програмної системи	16.04.2025	17.04.2025	Розділ 2
6	Розробка серверної частини (Node.js, Puppeteer, PostgreSQL), розробка клієнтської частини (React.js)	18.04.2025	23.04.2025	Розділ 3
7	Інтеграція з Telegram API, реалізація WebSocket-з'єднання	24.04.2025	03.05.2025	Розділ 4
8	Тестування програмної частини, валідація результатів, усунення помилок	03.05.2025	04.05.2025	Розділ 4
9	Оформлення пояснювальної записки	06.05.2025	12.05.2025	ПЗ, презентація
10	Попередній захист БКР	13.05.2025	13.05.2025	ПЗ, презентація
11	Підготовка супроводжуючих документів, їх підписання, проходження нормоконтролю та тесту на плагіат	13.05.2025	13.05.2025	ПЗ, презентація

6 Матеріали, що подаються до захисту БКР

До захисту подаються: пояснювальна записка БКР, графічні й ілюстративні матеріали, протокол попереднього захисту БКР на кафедрі, відгук наукового керівника, відгук рецензента, анотації до БКР українською та іноземною мовами, довідка про відповідність оформлення БКР діючим вимогам.

7 Порядок контролю виконання та захисту БКР

Виконання етапів графічної та розрахункової документації БКР контролюється науковим керівником згідно зі встановленими термінами. Захист БКР відбувається на засіданні Екзаменаційної комісії, затвердженої наказом ректора.

8 Вимоги до оформлювання та порядок виконання БКР

8.1 При оформлюванні БКР використовуються:

— ДСТУ 3008 : 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302 : 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 123 «Комп'ютерна інженерія» (освітня програма «Комп'ютерна інженерія») — кафедра обчислювальної техніки, ВНТУ 2023.

8.2 Порядок виконання БКР викладено в «Положення про кваліфікаційні роботи на першому (бакалаврському) рівні вищої освіти СУЯ ВНТУ-03.02.02-П.001.01:21».

ДОДАТОК Б

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: «Інформаційна система для моніторингу наявності товарів з функцією сповіщень»

Тип роботи: Бакалаврська кваліфікаційна робота
(БДР, МКР)

Підрозділ кафедра обчислювальної техніки
(кафедра, факультет)

Показники звіту подібності StrikePlagiarism

Оригінальність 98.5% Схожість 1.5%

Аналіз звіту подібності (відмітити потрібне):

✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.

☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Захарченко С.М.
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ Дерепа І.В.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Дудник О.В.
(підпис) (прізвище, ініціали)

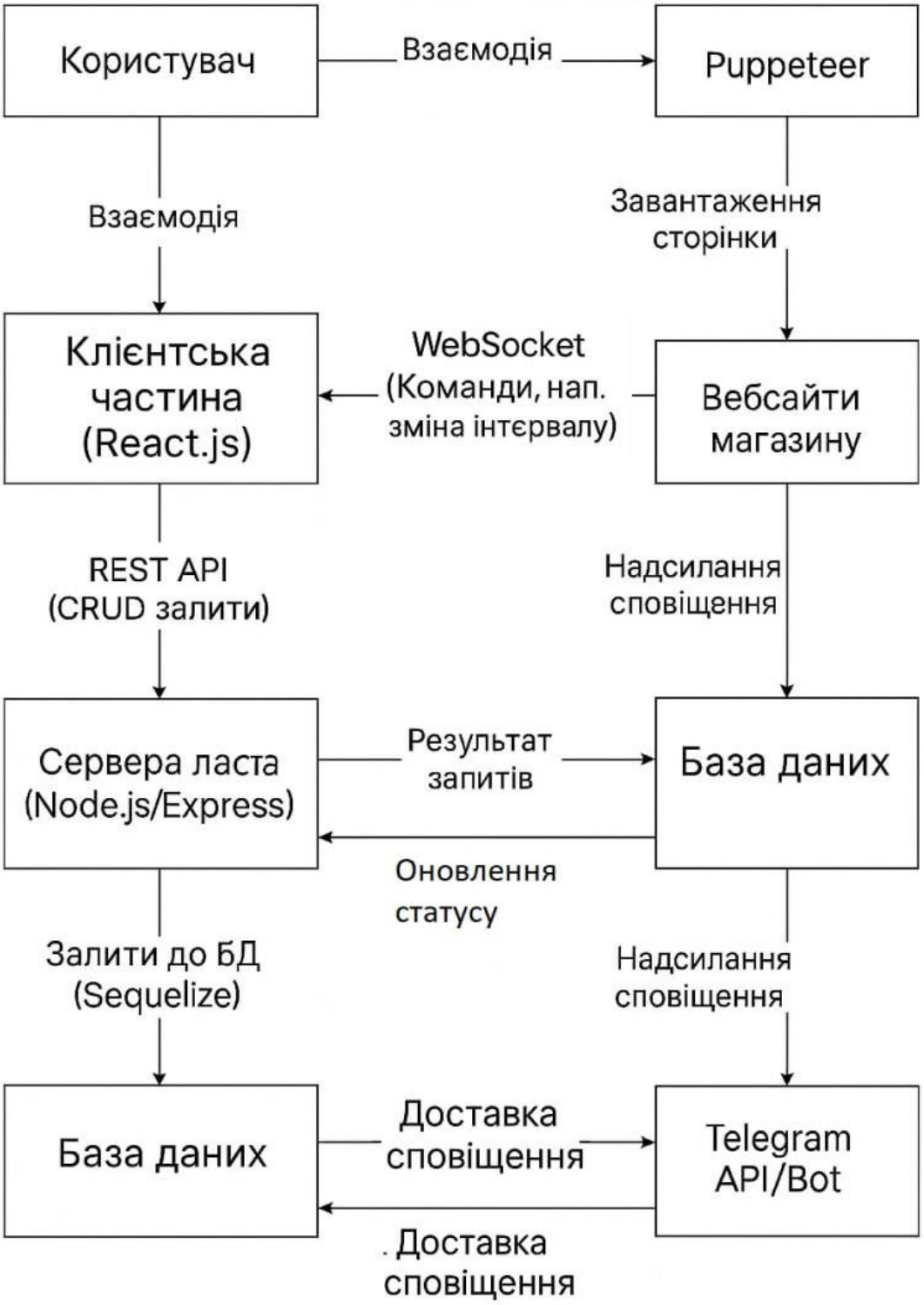
ДОДАТОК В

(обов’язковий)

ГРАФІЧНА ЧАСТИНА

БЛОК – СХЕМА АЛГОРИТМУ РОБОТИ СИСТЕМИ МОНІТОРИНГУ НАЯВНОСТІ ТОВАРІВ

Рисунок В.1 – Блок – схема алгоритму роботи системи моніторингу наявності товарів



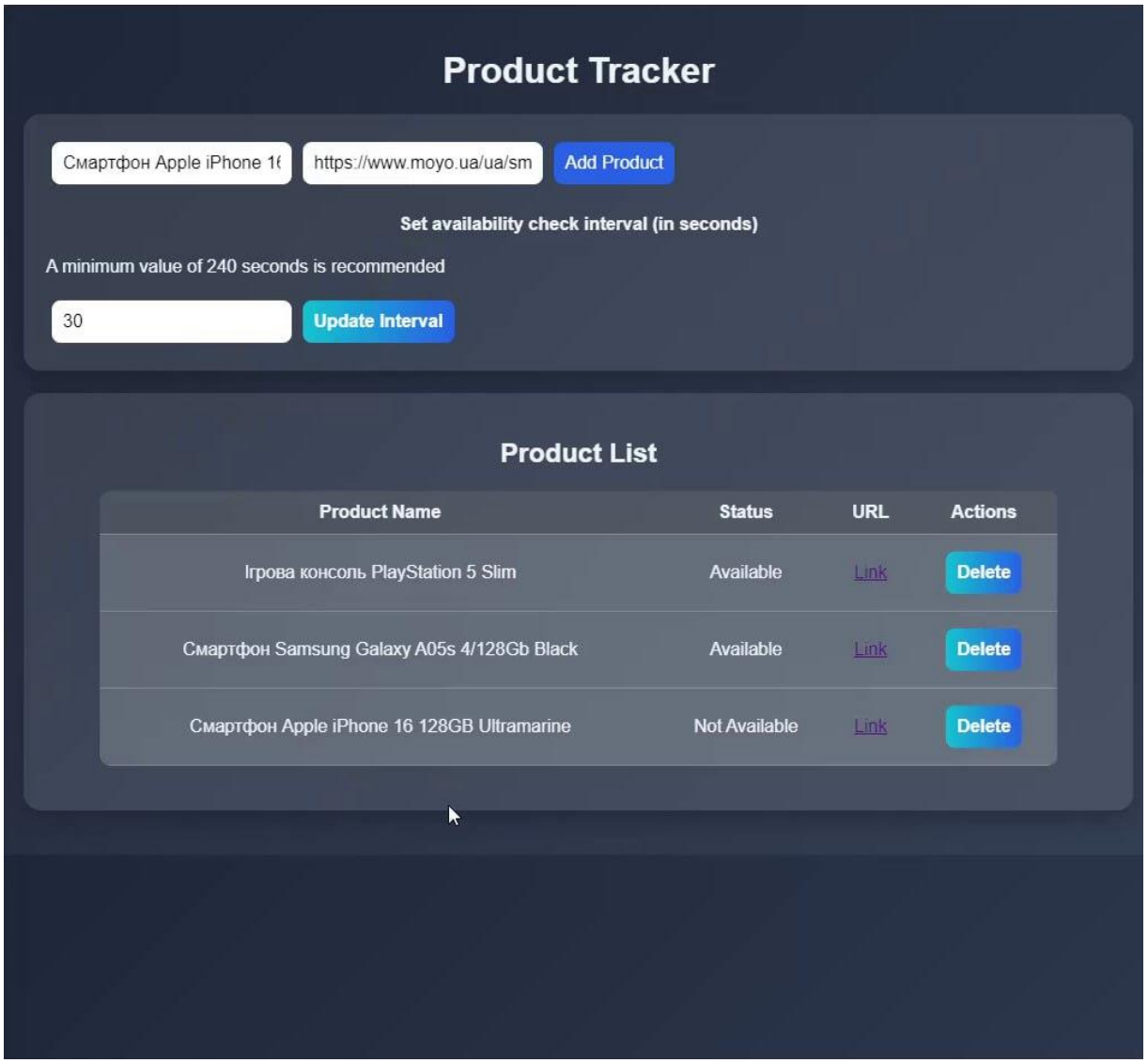
ДОДАТОК Г

(обов’язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

ГОЛОВНА СТОРІНКА ДОДАТКУ

Рисунок Г.1 – Вигляд головної сторінки додатку



ДОДАТОК Д

(довідниковий)

ЛІСТИНГИ СЕРВЕРНОЇ ЧАСТИНИ

Лістинг 2.1 – Ініціалізація Puppeteer та відкриття сторінки

```
const browser = await puppeteer.launch({});
const page = await browser.newPage();
```

Лістинг 2.2– Переходи між сторінками

```
await page.goto(url, {
  waitUntil: 'domcontentloaded',
  timeout: 60000
});
```

Лістинг 2.3 – Переходи між сторінками

```
await page.goto(url, {
  waitUntil: 'domcontentloaded',
  timeout: 60000
});
```

Лістинг 2.4 – Обробка помилок

```
try {
  await page.goto(url, { /* ... */ });
  return isAvailable;
} catch (error) {
  console.error('Error checking product availability:', error);
  return false;
}
```

Лістинг 2.5 – Закриття браузера

```
finally {
  await browser.close();
}
```

Лістинг 2.6 – Оновлення стану товару в базі

```
await product.update({
  availability: isAvailable,
  lastChecked: new Date()});
```

Лістинг 2.7 – Налаштування бібліотеки node-telegram-bot-api

```
const TelegramBot = require('node-telegram-bot-api');
const TELEGRAM_BOT_TOKEN = '<YOUR_TOKEN>';
const CHAT_ID = '<CHAT_ID>';
const bot = new TelegramBot(TELEGRAM_BOT_TOKEN, { polling: false });
```

Лістинг 2.8 – Функція надсилання повідомлень

```
const sendTelegramNotification = (product, isAvailable) => {
  const message = `
🔔 *Product Update*:
🛍️ *Name*: ${product.name}
🌐 *URL*: [Відкрити посилання](${product.url})
📶 *Status*: ${isAvailable ? '✅ Доступний' : '❌ Недоступний'}
  `;
  bot.sendMessage(CHAT_ID, message, { parse_mode: 'Markdown' });};
```

Лістинг 2.9 – Формування та надсилання повідомлення

```
exports.checkAvailability = async () => {

const products = await Product.findAll(); for (const product of products) {

const isAvailable = await checkProductAvailability(product.url); if (isAvailable && !product.availability) {
sendTelegramNotification(product, isAvailable);
}
await product.update({ availability: isAvailable, lastChecked: new Date() });
}};
```

ДОДАТОК Е

(довідниковий)

ЛІСТИНГИ КЛІЄНТСЬКОЇ ЧАСТИНИ

Лістинг 2.10 – Створення стану для продуктів

```
const [products, setProducts] = useState([]);
const [productName, setProductName] = useState(""); const [productURL, setProductURL] = useState("");
```

Лістинг 2.11 – Обробник події для форми додавання продукту

```
const handleAddProduct = async (event) => { event.preventDefault();
try {
await axios.post("http://localhost:5000/api/products/add", { name: productName,
url: productURL,
});
fetchProducts(); // Оновлення списку продуктів alert("Продукт додано успішно!");
} catch (error) {
console.error("Помилка при додаванні продукту", error);}};
```

Лістинг 2.12 – Хук для отримання актуальної інформації з сервера

```
useEffect(() => {

fetchProducts(); // Початкове завантаження продуктів socket.on("productUpdate", (updatedProducts) => {

setProducts(updatedProducts); // Оновлення списку продуктів в реальному часі
});
return () => {
socket.off("productUpdate"); // Очищення підключення при розмонтуванні компонента};}, []);
```

Лістинг 2.13 – Створення умов для відображення інтерфейсу

```
<tbody>
{products.map((product) => (
<tr key={product._id}>
<td>{product.name}</td>
<td>{product.availability ? "Available" : "Not Available"}</td>
<td>
<a href={product.url} target="_blank" rel="noopener noreferrer">Link</a>
</td>
</tr>
))}
</tbody>
```

Лістинг 2.14 – Приклад глобальних стилів

```
const GlobalStyle = createGlobalStyle` body {
margin: 0;
font-family: 'Arial', sans-serif;
background: linear-gradient(135deg, #1e293b, #334155); color: #f1f5f9;
overflow-x: hidden;}
h1, h2, h4 {
text-align: center; margin-bottom: 20px;}
button:hover {
background-color: #1d4ed8;}`;
```

Лістинг 2.15 – Контейнер для основного макета

```
export const Container = styled.div` max-width: 1200px;
margin: 0 auto;
padding: 20px;`;
```

Лістинг 2.16 – Компонент BlurCard для карток

```
export const BlurCard = styled.div`
background: rgba(255, 255, 255, 0.1); backdrop-filter: blur(15px);
border-radius: 15px; padding: 20px; margin: 20px auto;
box-shadow: 0px 10px 20px rgba(0, 0, 0, 0.2);
`;
```

Лістинг 2.17 – Кнопка з ефектом наведення

```
export const Button = styled.button`
background: linear-gradient(90deg, #06b6d4, #2563eb); color: #fff;
font-weight: bold;
transition: transform 0.2s ease;
&:hover {
transform: scale(1.05);
}`;
```

Лістинг 2.18 – Глобальні стилі

```
table {
width: 90%; margin: 20px auto;
border-collapse: collapse; background: rgba(255, 255, 255, 0.1); backdrop-filter: blur(10px);
border-radius: 10px; overflow: hidden;}
th, td {
padding: 10px; text-align: center;
border-bottom: 1px solid rgba(255, 255, 255, 0.2);
}
tr:hover {
background: rgba(255, 255, 255, 0.2);}
```

ДОДАТОК Є

(довідниковий)

ЛІСТИНГИ ВЗАЄМОДІЇ З БАЗОЮ ДАНИХ

Лістинг 2.19 – Ініціалізація з'єднання

```
import { io } from "socket.io-client";
const socket = io("http://localhost:5000", { transports: ["websocket"],
reconnection: true,});
```

Лістинг 2.20 – Отримання повідомлень від сервера

```
useEffect(() => {
fetchProducts();
socket.on("productUpdate", (updatedProducts) => { setProducts(updatedProducts);
});
return () => { socket.off("productUpdate");
};
}, []);
```

Лістинг 2.21 – Відправлення даних на сервер

```
const updateInterval = () => { socket.emit("updateInterval", interval);
console.log(`Update interval set to ${interval / 1000} seconds.`);};
```

Лістинг 2.22 – Запит до сервера за допомогою Axios import axios from "axios";

```
const fetchProducts = async () => { try {
const response = await axios.get(http://localhost:5000/api/products/all);
setProducts(response.data);
} catch (error) {
console.error("Error fetching products:", error);
}
};
```

Лістинг 2.23 – Встановлення WebSocket-з’єднання

```
import io from 'socket.io-client';
const socket = io("http://localhost:5000");
```