МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління

МАКАР Маркіян Олегович

Програмний інтерфейс для комунікаційного сервісу компанії / Software interface for the company's communication service

Спеціальність 122 – Комп'ютерні науки
Освітньо-професійна програма – Комп'ютерні науки

Кваліфікаційна робота

Виконав студент групи КН-43
М.О. Макар

Науковий керівник:
к.т.н., доцент, О.Р. Осолінський

Кваліфікаційну роботу допущено до захисту
«___»_____ 2025 р.

В.о. завідувача кафедри
_____ Н.М.Васильків

ТЕРНОПІЛЬ – 2025

Факультет комп'ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління
Освітній ступінь «бакалавр»
Спеціальність 122 – Комп'ютерні науки
Освітньо-професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Н.М. Васильків
«____» _____2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
МАКАРУ Маркіяну Олеговичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи: Програмний інтерфейс для комунікаційного сервісу компанії / Software interface for the company's communication service
керівник роботи О.Р. Осолінський, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 20 грудня 2024 р. № 938.
2. Строк подання студентом закінченої кваліфікаційної роботи 25 травня 2025 р.
3. Вихідні дані до кваліфікаційної роботи: завдання на кваліфікаційну роботу студента, наукові статті, технічна література.
4. Основні питання, які потрібно розробити:

– провести аналіз предметної області, описати особливості корпоративних комунікаційних систем та проаналізувати сучасні технології для розробки веб-додатків реального часу.
– провести огляд існуючих комунікаційних рішень та їх архітектур;
– обґрунтувати вибір стеку технологій для розробки;
– сформувати постановку задачі та визначення вимоги до системи;
– розробити архітектуру програмного інтерфейсу та серверної частини, описати компоненти системи та їх взаємодію в Docker-середовищі;
– провести проектування бази даних для зберігання інформації про користувачів, чати, повідомлення та з'єднання;
– розробити алгоритмічне забезпечення ключових алгоритмів обробки HTTP-запитів, встановлення та управління WebSocket-з'єднаннями, надсилання та отримання повідомлень, оновлення статусів;
– провести вибір середовища розробки та розгортання;
– розробити програмний інтерфейс та основні сервіси, конфігурації проєкту;
– провести тестування розробленого програмного інтерфейсу за допомогою методики тестування API (Postman), тестування WebSocket-з'єднань, функціональне, та навантажувальне.

5. Перелік графічного матеріалу в роботі:

– архітектура системи в Docker-середовищі;
– схеми алгоритмів обробки та надсилання WebSocket повідомлення.
– схеми алгоритмів обробки та надсилання повідомлень в серверній частині додатку.

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 20 грудня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів кваліфікаційної роботи	Примітка
1	Затвердження теми кваліфікаційної роботи, ознайомлення з літературними джерелами та складання плану роботи	до 01.01. 2025 р.	
2	Написання 1 розділу кваліфікаційної роботи	до 01.02. 2025 р.	
3	Написання 2 розділу кваліфікаційної роботи	до 01.04.2025 р.	
4	Написання 3 розділу кваліфікаційної роботи	до 01.05. 2025 р.	
5	Представлення попереднього варіанту кваліфікаційної роботи, перевірка та внесення змін керівником	до 15.05.2025 р.	
6	Опрацювання зауважень та представлення завершеного варіанту кваліфікаційної роботи. Підготовка супроводжуючих документів	до 25.05.2025 р.	
7	Перевірка кваліфікаційної роботи на оригінальність тексту	до 30.05.2025 р.	
8	Оформлення кваліфікаційної роботи та отримання допуску до захисту	до 10.06.2025 р.	
9	Подання кваліфікаційної роботи до захисту на засіданні атестаційної комісії	до 10.06. 2025 р.	

Студент _____ М.О. Макар
(підпис) (прізвище та ініціали)

Керівник кваліфікаційної роботи _____ О.Р. Осолінський
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кваліфікаційна робота на тему «Програмний інтерфейс для комунікаційного сервісу компанії» на здобуття освітнього ступеня «бакалавр» зі спеціальності 122 «Комп’ютерні науки» освітньої програми «Комп’ютерні науки» написана обсягом в 51 сторінку і містить 12 ілюстрацій, 6 додатків та 31 використане джерело.

Метою роботи є розробка програмного інтерфейсу для чат-сервісу компанії з підтримкою комунікації в реальному часі. У роботі використано методи системного аналізу, архітектурного проектування, об'єктно-орієнтованого програмування, контейнеризації та функціонального тестування.

Запропоновано високопродуктивну архітектуру, що поєднує Laravel 11 із розширенням Swoole для асинхронної обробки запитів, WebSocket для двосторонньої комунікації та Docker для ізольованого розгортання сервісів. Реалізовано ключові компоненти чат-системи: створення чатів, надсилання та читання повідомлень, управління WebSocket-з’єднаннями, кешування та зберігання даних.

Розроблено систему, яка здатна ефективно обробляти велику кількість одночасних з’єднань, забезпечує гнучке масштабування, надійність, ізоляцію сервісів та зручність у розгортанні. Користувачі мають змогу в режимі реального часу обмінюватися повідомленнями через веб-інтерфейс або мобільний додаток, що з'єднується через WebSocket.

Практичне значення розробки полягає у створенні готового до використання серверного інтерфейсу для сучасного комунікаційного сервісу, який може бути адаптований під різні бізнес-задачі.

Ключові слова: LARAVEL, SWOOLE, WEBSOCKET, DOCKER, PHP, АСИНХРОННЕ ПРОГРАМУВАННЯ, REAL-TIME, API, ЧАТ, КОНТЕЙНЕРИЗАЦІЯ, NGINX, MYSQL, REDIS.

ANNOTATION

The qualification work on the topic «Software interface for the company's communication service» for the degree of «bachelor» in specialty 122 «Computer Science» of the educational program «Computer Science» is written in 51 pages and contains 12 illustrations, 6 appendices and 31 sources used.

The aim of this work is to develop a software interface for a company’s chat service that supports real-time communication. The study employs methods of system analysis, architectural design, object-oriented programming, containerization, and functional testing.

A high-performance architecture is proposed, combining Laravel 11 with the Swoole extension for asynchronous request handling, WebSocket for bidirectional communication, and Docker for isolated service deployment. The key components of the chat system have been implemented: chat creation, message sending and reading, WebSocket connection management, caching, and data storage.

The developed system is capable of efficiently handling a large number of concurrent connections, providing flexible scalability, reliability, service isolation, and ease of deployment. Users can exchange messages in real time through a web interface or a mobile application that connects via WebSocket.

The practical value of the project lies in delivering a ready-to-use backend interface for a modern communication service, which can be adapted to various business needs.

Keywords: LARAVEL, SWOOLE, WEBSOCKET, DOCKER, PHP, ASYNCHRONOUS PROGRAMMING, REAL-TIME, API, CHAT, CONTAINERIZATION, NGINX, MYSQL, REDIS.

ЗМІСТ

ВСТУП

Актуальність теми дослідження. Сучасний цифровий світ характеризується стрімким зростанням потреби у миттєвій комунікації. Чат-системи, платформи для спільної роботи, системи сповіщень та інші додатки [1], що функціонують у реальному часі, стали невід'ємною частиною як повсякденного життя, так і бізнес-процесів. Традиційна модель роботи веб-додатків, особливо тих, що розроблені на PHP з використанням підходу PHP-FPM (FastCGI Process Manager) [2], демонструє обмеження при роботі з великою кількістю одночасних довготривалих з'єднань, необхідних для технологій реального часу, таких як WebSocket [3]. Кожен запит у такій моделі зазвичай ініціює повне або часткове завантаження фреймворку та логіки додатку, що призводить до значних затримок та високого споживання серверних ресурсів (CPU, пам'ять), особливо під високим навантаженням.

Для подолання цих обмежень активно розвиваються підходи, засновані на асинхронному програмуванні та подієво-орієнтованих моделях. У світі PHP значної популярності набуло розширення Swoole [4] – високопродуктивний асинхронний мережевий комунікаційний рушій, написаний на C/C++ [5], що дозволяє PHP-додаткам ефективно обробляти тисячі одночасних з'єднань, підтримувати довготривалі процеси, використовувати корутини для неблокуючої обробки операцій вводу/виводу та реалізовувати власні TCP/UDP/WebSocket [1] сервери безпосередньо на PHP. Інтеграція Swoole з популярними фреймворками, такими як Laravel [6], за допомогою адаптерів як hhxsv5/laravel-s [7], відкриває шлях до створення високопродуктивних PHP-додатків реального часу.

Разом з тим, розгортання та управління такими складними системами, що часто складаються з кількох взаємодіючих сервісів (веб-сервер, бекенд, база даних, кеш), зумовлює значні технічні виклики. Технологія контейнеризації Docker [8] надає ефективне рішення цієї проблеми, дозволяючи ізолювати кожен сервіс у власному контейнері, забезпечуючи консистентність середовища на всіх етапах, спрощуючи розгортання та масштабування.

Таким чином, дослідження та розробка програмного інтерфейсу для комунікаційного сервісу з використанням синергії Laravel, Swoole та Docker є актуальною задачею, що дозволяє поєднати швидкість розробки та багату екосистему Laravel з високою продуктивністю та можливостями реального часу Swoole, забезпечуючи при цьому надійність та гнучкість розгортання за допомогою Docker.

Метою кваліфікаційної роботи є розробка та реалізація масштабованого, високоефективного програмного інтерфейсу (API) [9] для чат-сервісу, що підтримує комунікацію в реальному часі, з використанням фреймворку Laravel 11, розширення Swoole (через адаптер hhxsv5/laravel-s) та технології контейнеризації Docker.

Предметом дослідження є технології та методи розробки високопродуктивних програмних інтерфейсів для комунікаційних сервісів з використанням асинхронного програмування (Swoole), веб-сокетів та контейнеризації (Docker) на базі фреймворку Laravel.

Методи дослідження: Для вирішення поставлених завдань у роботі було використано системний аналіз, порівняльний аналіз, метод архітектурного проектування, метод об'єктно-орієнтованого програмування, методи контейнеризації та тестування програмного забезпечення.

Практичне значення одержаних результатів полягає в розробці програмного інтерфейсу та серверної частини для чат-сервісу, що функціонує в реальному часі. Розроблена система, розгорнута у Docker-контейнерах, демонструє високу продуктивність та масштабованість. Вона може бути використана як основа для створення повноцінного комерційного або корпоративного комунікаційного сервісу.

Результати кваліфікаційної роботи апробовані та опубліковані у матеріалах студентської науково-практичної конференції “Стан, досягнення і перспективи інформаційних систем і технологій” м. Одеса, Україна, 17-18 квітня 2025 р.

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

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Особливості корпоративних комунікаційних систем

Корпоративні комунікаційні системи є невід'ємною частиною сучасної бізнес-інфраструктури. Вони призначені для забезпечення ефективної взаємодії між співробітниками, відділами та підрозділами компанії, а також для обміну інформацією з зовнішніми партнерами та клієнтами. Еволюція корпоративних комунікацій [10] пройшла шлях від традиційних засобів, як телефони та електронні пошти, до інтегрованих платформ, що включають обмін миттєвими повідомленнями – чати, відео конференції, спільний доступ до файлів та управління проєктами.

Основні функції корпоративних комунікаційних систем включають обмін текстовими повідомленнями в режимі реального часу (індивідуальні та групові чати), передачу файлів та документів, голосові та відеодзвінки, організацію відеоконференцій, спільний доступ до екрана, інтеграцію з іншими корпоративними системами (CRM, ERP, таск-трекери) [11], управління статусами користувачів та повідомлення про події.

Ключовими вимогами до корпоративних комунікаційних систем є безпека та конфіденційність, надійність, продуктивність та масштабованість, зручність використання та можливість інтеграції з існуючою IT-інфраструктурою компанії. Розробка власного корпоративного комунікаційного сервісу, орієнтованого на специфічні потреби компанії, дозволяє досягти високого рівня безпеки, гнучкості та контролю над даними, що є критично важливим для багатьох організацій.

1.2 Аналіз сучасних технологій для розробки веб-додатків реального часу

Розробка веб-додатків, що працюють у режимі реального часу, вимагає використання спеціалізованих технологій та архітектурних підходів, які відрізняються від традиційної моделі "запит-відповідь" [9]. Для забезпечення миттєвого обміну даними між сервером та клієнтом без необхідності постійного

опитування, використовуються такі технології, як WebSocket, Server-Sent Events (SSE) [12] та асинхронні сервери.

1.2.1 Порівняння PHP-FPM та асинхронних серверів (Swoole)

Традиційна архітектура веб-серверів, що використовують PHP (наприклад, Nginx [2] з PHP-FPM), базується на моделі "один запит - один процес/потік". Кожен вхідний HTTP-запит обробляється окремим процесом PHP-FPM, який виконує скрипт і повертає результат. Ця модель добре підходить для більшості веб-сайтів та додатків, але має суттєві обмеження при роботі з довготривалими з'єднаннями або великою кількістю одночасних запитів, характерних для додатків реального часу. У таких сценаріях, утримання великої кількості процесів PHP-FPM може призвести до значного споживання ресурсів та обмеження масштабованості [13].

На противагу цьому, асинхронні сервери, такі як Swoole, використовують подієво-орієнтовану архітектуру (event-driven). Swoole є розширенням для PHP, яке надає можливості для створення високопродуктивних асинхронних серверів, включаючи HTTP, WebSocket, TCP та UDP сервери. Замість створення нового процесу для кожного з'єднання, Swoole використовує неблокуючий I/O та пул воркерів, що дозволяє одному процесу обробляти тисячі одночасних з'єднань. Це робить Swoole ідеальним вибором для розробки додатків реального часу, де необхідно підтримувати велику кількість активних WebSocket-з'єднань.

Основні переваги Swoole над традиційним PHP-FPM для додатків реального часу включають високу продуктивність завдяки асинхронній обробці, вбудовану підтримку протоколу WebSocket, ефективне управління довготривалими з'єднаннями та менше споживання ресурсів. Використання Swoole спільно з фреймворком Laravel за допомогою інтеграційного пакету hhxsv5/laravel-s дозволяє поєднати переваги швидкої розробки на Laravel з високою продуктивністю асинхронного сервера Swoole.

1.2.2 Технологія WebSocket

WebSocket – це протокол зв'язку, що забезпечує повнодуплексний канал зв'язку через одне TCP-з'єднання. На відміну від традиційного HTTP, де клієнт надсилає запит, а сервер повертає відповідь, WebSocket дозволяє як клієнту, так і серверу надсилати дані один одному в будь-який момент часу після встановлення з'єднання [1]. Це робить WebSocket ідеальним рішенням для додатків, що вимагають обміну даними в реальному часі, таких як онлайн-чати, багатокористувацькі ігри, системи моніторингу та фінансові додатки.

Переваги використання WebSocket [3]:

- Низька затримка: Дані передаються миттєво без необхідності встановлення нового з'єднання для кожного повідомлення.
- Ефективне використання ресурсів: Зменшується навантаження на сервер та мережу порівняно з polling-підходами.
- Повнодуплексний зв'язок: Можливість одночасної передачі даних в обох напрямках.

1.2.3 Контейнеризація за допомогою Docker

Docker – це платформа для розробки, доставки та запуску додатків за допомогою контейнерів. Контейнеризація дозволяє упаковувати додаток та всі його залежності (бібліотеки, інструменти, код, середовище виконання) в ізольований контейнер, який може бути легко перенесений та запущений на будь-якій системі, що підтримує Docker. Це забезпечує консистентність середовища на всіх етапах життєвого циклу розробки (розробка, тестування, продакшн) та спрощує процеси розгортання та масштабування. Використання Docker дозволяє [8]:

- Ізолювати компоненти системи, тобто кожен сервіс (Nginx, PHP/Swoole, MySQL, Redis) працює у власному контейнері, що зменшує конфлікти залежностей та спрощує управління.

- Спростити налаштування середовища розробки для швидкого розгортання повного робочого середовища за допомогою одного файлу docker-compose.yml.
- Забезпечення переносимості контейнерів між різними середовищами такими як локальна машина розробника, staging, production сервери.
- Спрощення масштабування для збільшення кількості екземплярів контейнерів для певних сервісів.

1.3 Огляд існуючих комунікаційних рішень та їх архітектур

На ринку існує велика кількість готових комунікаційних рішень, як комерційних, так і відкритих, що пропонують різноманітний функціонал для корпоративних комунікацій. До популярних комерційних рішень належать Slack [14], Zoom та Microsoft Teams [15]. Серед відкритих альтернатив можна виділити Mattermost [16], Rocket.Chat та Zulip [17].

Аналіз архітектур існуючих рішень показує, що більшість сучасних платформ використовують мікро сервісний підхід, де різні функції такі як обробка повідомлень, управління користувачами, файлове сховище та відеоконференції реалізовані, як окремі сервіси, що взаємодіють між собою. Для забезпечення роботи в реальному часі широко застосовується технологія WebSocket. Для зберігання даних використовуються як реляційні БД, наприклад PostgreSQL [18], MySQL [19], так і NoSQL[20] бази даних як MongoDB та Redis [21] для кешування та тимчасових даних. Для масштабування та відмовостійкості застосовуються брокери повідомлень такі які RabbitMQ та Kafka [22], або системи оркестрації контейнерів Kubernetes [23].

Хоча готові рішення пропонують багатий функціонал, їх використання може бути обмежене з точки зору кастомізації, інтеграції зі специфічними внутрішніми системами та відповідності корпоративним політикам безпеки. Розробка власного рішення дозволяє повністю контролювати ці аспекти.

1.4 Обґрунтування вибору стеку технологій для розробки

Вибір стеку технологій для розробки програмного інтерфейсу комунікаційного сервісу базується на вимогах до системи, необхідності забезпечення роботи в реальному часі, масштабованості, швидкості розробки та наявності відповідних знань у команди.

Laravel 11 буде використано як основний фреймворк для розробки бекенду обрано Laravel 11. Laravel є популярним PHP-фреймворком, який надає широкий набір інструментів для швидкої розробки веб-додатків, включаючи ORM [24] (Eloquent), систему маршрутизації, механізми аутентифікації та авторизації, а також підтримку черг повідомлень. Це дозволяє зосередитись на бізнес-логіці, а не на реалізації базових функцій.

Для забезпечення роботи в режимі реального часу та обробки WebSocket-з'єднань обрано Swoole. Інтеграційний пакет hhxsv5/laravel-s, який дозволяє запускати Laravel-додаток на асинхронному сервері Swoole, використовуючи переваги обох технологій. Це забезпечує високу продуктивність та ефективне управління довготривалими з'єднаннями [7], необхідними для чат-системи.

Використання протоколу WebSocket є ключовим для реалізації функціональності обміну повідомленнями в реальному часі без затримок.

Для контейнеризації та спрощення процесів розробки, розгортання та управління інфраструктурою обрано Docker. Docker Compose [25] використовується для визначення та запуску багато контейнерних Docker-додатків.

В якості фронтенд-веб-сервера для обробки HTTP-запитів та проксування WebSocket-з'єднань до Swoole-сервера використовується Nginx. Він відомий своєю високою продуктивністю та ефективністю.

Як основну реляційну базу даних для зберігання структурованих даних, таких як інформація про користувачів, чати та повідомлення вибрано MySQL, яка є надійною та широко використовуваною СУБД.

Як кеш та брокер повідомлень використовується Redis який є високопродуктивним сховищем даних типу "ключ-значення" в пам'яті, що ідеально підходить для зберігання тимчасових даних [21], сесій, а також для реалізації механізмів публікації/підписки для обміну повідомленнями між різними компонентами системи або воркерами Swoole.

Обраний стек технологій є сучасним, добре документованим та має активну спільноту підтримки, що сприяє швидкій та ефективній розробці.

1.5 Постановка задачі та визначення вимог до системи

На основі аналізу предметної області та вимог до корпоративних комунікаційних систем, було сформульовано задачу розробки програмного інтерфейсу для комунікаційного сервісу компанії. Система повинна забезпечити базову функціональність обміну текстовими повідомленнями в режимі реального часу між зареєстрованими користувачами.

1.5.1 Функціональні вимоги

Програмний інтерфейс комунікаційного сервісу передбачає реалізацію низки основних функціональних можливостей, необхідних для повноцінної взаємодії користувачів у рамках чат-системи. Однією з базових функцій є реєстрація користувачів, що дозволяє створювати нові облікові записи у системі. Після цього забезпечується можливість автентифікації шляхом входу з логіном та паролем.

Система також підтримує базове управління обліковими записами, зокрема перегляд інформації про користувачів із певними обмеженнями доступу. Важливими функціональними компонентами є створення індивідуальних чатів між двома користувачами. Також додаткові функції зміни статусу та назви чатів.

Комунікація в чатах здійснюється через надсилання текстових повідомлень, які доставляються в реальному часі за допомогою WebSocket-з’єднань або REST API. Користувачі можуть не лише отримувати нові повідомлення миттєво, але й

переглядати історію листування у кожному чаті. Додаткові функції прикріплень та відповіді на повідомлення. Для зручності навігації передбачено також можливість перегляду списку всіх активних чатів, у яких бере участь конкретний користувач.

1.5.2 Нефункціональні вимоги

Нефункціональні характеристики системи орієнтовані на забезпечення її надійності, ефективності та зручності в експлуатації. Одним із ключових параметрів є продуктивність, тобто система повинна гарантувати мінімальну затримку при доставці повідомлень, що не перевищує 100 мілісекунд. Архітектура розроблена з урахуванням потреби у горизонтальному масштабуванні, що дозволяє адаптувати її до збільшення навантаження у разі зростання кількості користувачів та обсягу переданих повідомлень.

Система повинна бути стійкою до часткових відмов і збоїв, забезпечуючи збереження даних і відновлення роботи без втрати інформації. Особливу увагу приділено питанням безпеки — передбачається реалізація механізмів захисту від несанкціонованого доступу, збереження конфіденційності персональних даних і шифрування переданої інформації.

Крім того, важливою характеристикою є доступність сервісу: він має функціонувати цілодобово, з мінімальними перервами у наданні послуг. Завдяки використанню контейнеризації з Docker, розгортання системи спрощується і прискорюється, що особливо актуально для забезпечення оперативного масштабування або перенесення в інше середовище.

Для досягнення поставлених вимог необхідно вирішити наступні завдання:

– провести аналіз предметної області, описати особливості корпоративних комунікаційних систем та проаналізувати сучасні технології для розробки веб-додатків реального часу.

– розробити архітектуру програмного інтерфейсу та серверної частини, описати компоненти системи та їх взаємодію в Docker-середовищі;

– провести проектування бази даних для зберігання інформації про користувачів, чати, повідомлення та з'єднання;

– розробити алгоритмічне забезпечення ключових алгоритмів обробки HTTP-запитів, встановлення та управління WebSocket-з'єднаннями, надсилання та отримання повідомлень, оновлення статусів;

– провести вибір середовища розробки та розгортання;

– розробити програмний інтерфейс та основні сервіси, конфігурації проєкту;

– провести тестування розробленого програмного інтерфейсу за допомогою методики тестування API, тестування WebSocket-з'єднань, функціональне, та навантажувальне.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ІНТЕРФЕЙСУ ТА СИСТЕМИ

2.1 Архітектура системи

Архітектура розробленого комунікаційного сервісу базується на принципах мікро сервісів та використовує контейнеризацію за допомогою Docker Compose для організації та взаємодії компонентів. Система складається з декількох ключових сервісів, кожен з яких виконує свою специфічну роль.

2.1.1 Структура системи

На рисунку 2.1 зображено структуру, що відображає взаємодію між основними компонентами, такими як клієнтський застосунок (веб-інтерфейс або мобільний додаток), фронтенд-сервер Nginx, бекенд-сервіс на базі Laravel/Swoole, база даних MySQL та сховище тимчасових даних/брокер повідомлень Redis. Повну архітектуру комунікаційного сервісу надано в додатку А.

Клієнт взаємодіє з системою через Nginx, який виступає єдиною точкою входу, маршрутизуючи стандартні HTTP-запити до Laravel-додатку та проксуючи WebSocket-з'єднання безпосередньо до Swoole-сервера. Для аутентифікації використовується стандартна для галузі реалізація OAuth2 — Laravel Passport. Це дозволяє безпечно взаємодіяти з сервером за допомогою токенів доступу. Після успішної аутентифікації клієнт отримує не лише доступ до захищених API-ендпоінтів, але й право на встановлення WebSocket-з'єднання. Для цього на стороні клієнта токен передається в заголовках запиту або безпосередньо в запиті на встановлення WebSocket-з'єднання.

Наступним кроком є встановлення цього з'єднання зі Swoole-сервером для обміну повідомленнями в реальному часі. Бекенд-сервіс працює з даними, що зберігаються в MySQL, та активно використовує Redis для кешування, управління активними WebSocket-з'єднаннями та, потенційно, для реалізації механізмів публікації/підписки для масштабування. Такий підхід значно розвантажує основну базу даних та прискорює ідентифікацію користувачів при обміні повідомленнями, що є критично важливим для забезпечення високої продуктивності системи.

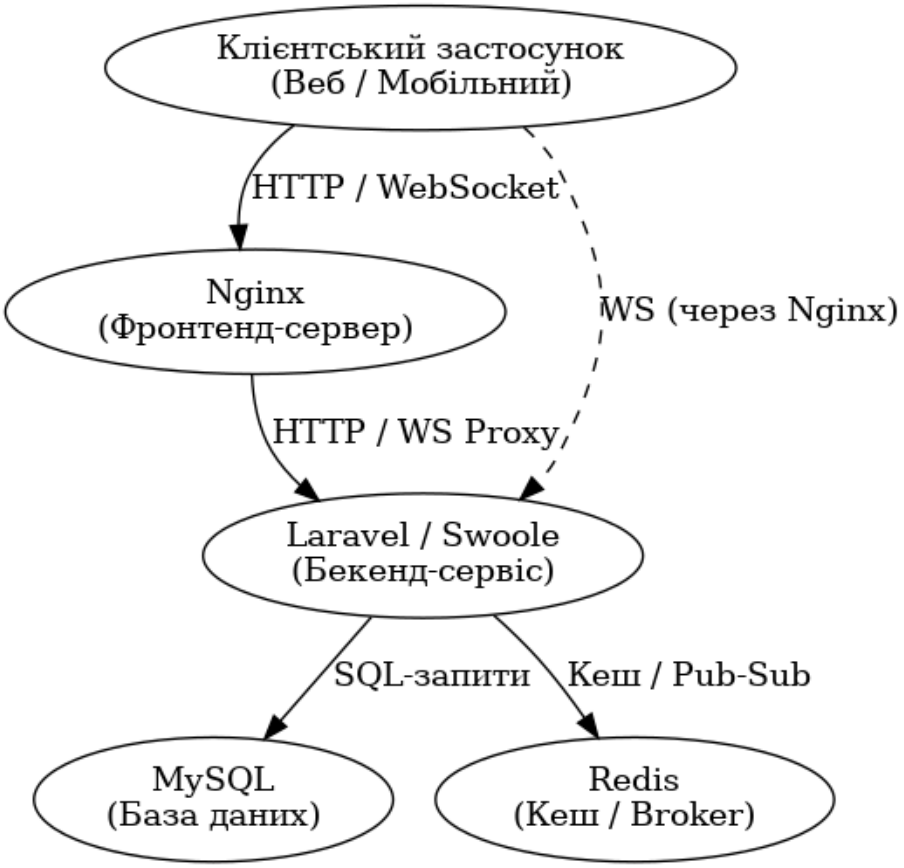


Рисунок 2.1 – Структура комунікаційного сервісу

2.1.2 Опис взаємодії компонентів на базі Docker Compose

Файл docker-compose.yml визначає конфігурацію сервісів, їх образи, порти, томи та залежності.

Сервіс nginx веб-сервера, що використовує образ nginx:stable-alpine налаштований для прослуховування HTTP-трафіку на стандартному порту 80 (або іншому, прописаному в конфігурації) та проксування запитів до бекенд-сервісу. Важливою частиною конфігурації Nginx є налаштування проксування WebSocket-з'єднань, що дозволяє клієнтам встановлювати прямий зв'язок зі Swoole-сервером через Nginx. Приклад конфігурації може включати директиви proxy_pass для HTTP та proxy_http_version, proxy_set_header Upgrade, proxy_set_header Connection для WebSocket.

Основний бекенд-сервіс, що працює на базі Laravel 11 та Swoole за допомогою інтеграції hhxsv5/laravel-s використовує власний Dockerfile для збирання образу, який включає PHP з розширенням Swoole та встановлені

залежності проєкту. Цей сервіс прослуховує HTTP та WebSocket запити, що проксуються Nginx. Swoole-сервер обробляє запити асинхронно, дозволяючи ефективно управляти великою кількістю одночасних з'єднань.

Сервіс бази даних MySQL використовує образ mysql:8.0. Дані бази даних зберігаються у томі, що забезпечує їх збереження між перезапусками контейнера. Налаштовуються змінні середовища для імені бази даних, користувача та пароля.

Сервіс Redis використовує образ redis:alpine використовується для кешування даних, зберігання інформації про активні WebSocket-з'єднання та як брокер повідомлень для взаємодії між воркерами Swoole або іншими потенційними сервісами.

Взаємодія між сервісами відбувається через внутрішню мережу Docker. Nginx звертається до backend за його іменем сервісу, бекенд-сервіс звертається до database та redis також за їх іменами. Така архітектура забезпечує гнучкість, ізоляцію компонентів та спрощує процеси розробки та розгортання.

2.2 Проектування бази даних

Для зберігання інформації про користувачів, чати та повідомлення розроблено реляційну модель даних. База даних MySQL була обрана через її надійність, широку підтримку та відповідність вимогам проєкту.

2.2.1 Модель даних

Модель даних включає наступні основні сутності:

1. Users: Зберігає інформацію про користувачів системи. Основні поля можуть включати унікальний ідентифікатор, ім'я користувача, час останнього підключення, адресу електронної пошти, хешований пароль, час створення та оновлення запису. Таблиця users.

2. Chats: Зберігає інформацію про чати (розмови). Поля можуть включати унікальний ідентифікатор, тип чату (індивідуальний, груповий), назву чату, час створення та оновлення. Таблиця chats.

3. ChatMessages: Зберігає інформацію про повідомлення в чатах. Поля можуть включати унікальний ідентифікатор, ідентифікатор чату, ідентифікатор автора повідомлення, текст повідомлення, час створення. Таблиця messages.

4. ChatUser: Допоміжна таблиця для реалізації зв'язку "багато-до-багатьох" між користувачами та чатами. Зберігає інформацію про те, які користувачі є учасниками яких чатів. Поля можуть включати ідентифікатор зв'язку, ідентифікатор чату, ідентифікатор користувача, час приєднання. Таблиця chat_users.

5. Connections: Ця таблиця використовується для тимчасового зберігання інформації про активні WebSocket-з'єднання. Вона пов'язує ідентифікатор користувача з файловим дескриптором його активного WebSocket-з'єднання на Swoole-сервері. Це дозволяє швидко знаходити активні з'єднання користувачів для надсилання повідомлень. Поля можуть включати ідентифікатор з'єднання, ідентифікатор користувача, час встановлення з'єднання. Таблиця connections.

Зв'язки між сутностями:

- Користувач може мати багато повідомлень “один-до-багатьох”.
- Чат може мати багато повідомлень “один-до-багатьох”.
- Користувач може брати участь у багатьох чатах, і чат може мати багато користувачів, реалізується через таблицю chat_users.
- Користувач може мати одне або декілька активних WebSocket-з'єднань “один-до-багатьох”.

2.2.2 Діаграма сутність-зв'язок (ER-діаграма)

Рисунок 2.2 представляє діаграму сутність-зв'язок (ER-діаграму) розробленої моделі даних [26]:

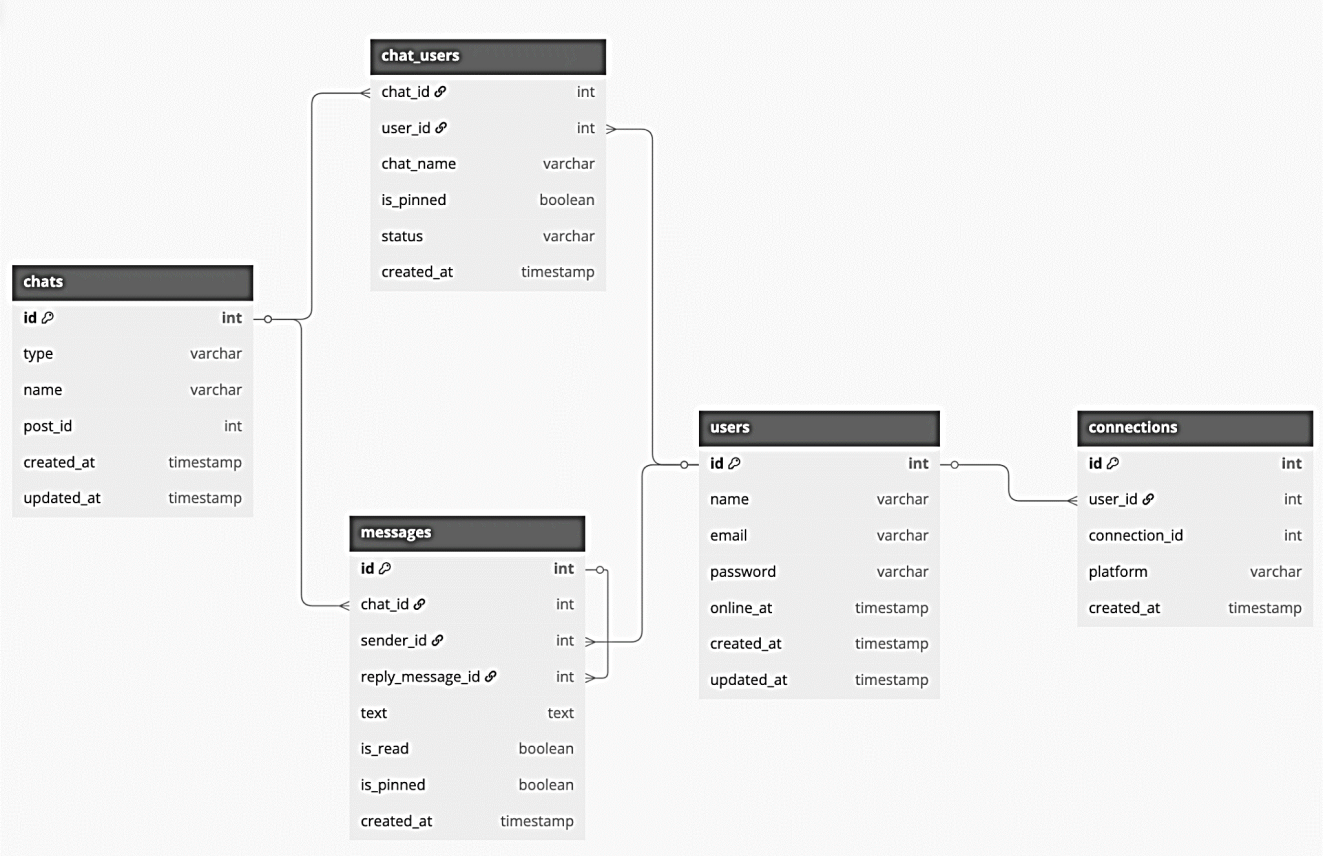


Рисунок 2.2 – Діаграма сутність-зв'язок (ER-діаграма) бази даних

Діаграма візуалізує сутності (таблиці) та зв'язки між ними, демонструючи структуру бази даних, необхідну для підтримки функціональності чат-системи.

2.3 Проектування основних алгоритмів

Проектування алгоритмів охоплює ключові операції системи, забезпечуючи логіку їх виконання. Далі в розділі представлено алгоритми для реєстрації, управління чатами, надсилання/отримання повідомлень та управління WebSocket-з'єднаннями.

2.3.1 Алгоритм реєстрації та аутентифікації користувача

Алгоритм реєстрації нового користувача передбачає отримання даних (ім'я користувача, електронна пошта, пароль), їх валідацію, хешування пароля та збереження нової сутності користувача в базі даних. У разі успішної реєстрації

користувачу надається токен для подальшої аутентифікації. Схему алгоритму зображено на рисунку 2.3.

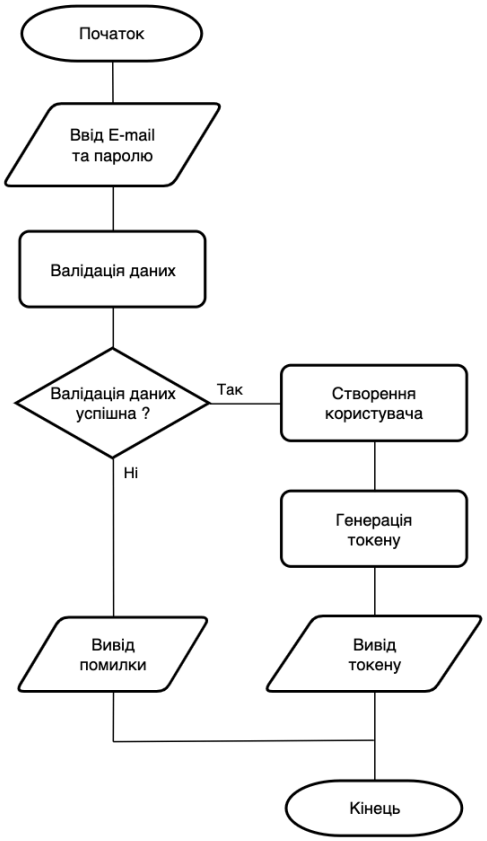


Рисунок 2.3 – Схема алгоритму реєстрації користувача

Алгоритм аутентифікації користувача включає отримання облікових даних (електронна пошта/ім'я користувача та пароль), пошук користувача в базі даних за логіном, порівняння наданого пароля з хешованим паролем, що зберігається. У разі успішної перевірки, користувачу видається аутентифікаційний токен що використовується для ідентифікації користувача при подальших запитах.

2.3.2 Алгоритм створення та управління чатами

Алгоритм створення індивідуального чату передбачає перевірку існування чату між двома вказаними користувачами. Якщо такий чат вже існує, повертається інформація про нього. Якщо ні, створюється нова сутність чату типу "індивідуальний" та додаються записи до таблиці ChatUser для обох учасників.

Алгоритм оновлення чату, а саме: зміна назви, зміна статусу (активні та архівні), закріплення чату.

2.3.3 Алгоритм надсилання та отримання повідомлень через WebSocket

Алгоритм надсилання повідомлення починається з отримання повідомлення від клієнта через встановлене WebSocket-з'єднання. Сервер ідентифікує користувача за його з'єднанням та отримує ідентифікатор чату, в який надсилається повідомлення. Здійснюється перевірка, чи є користувач учасником даного чату. Якщо так, створюється нова сутність повідомлення в базі даних. Після успішного збереження повідомлення, сервер отримує список усіх активних WebSocket-з'єднань користувачів, які є учасниками цього чату. Використовуючи Swoole, сервер асинхронно надсилає нове повідомлення (або сповіщення про нього) всім підключеним клієнтам-учасникам чату. Схема надсилання повідомлення через WebSocket зображена на рисунку 2.4.

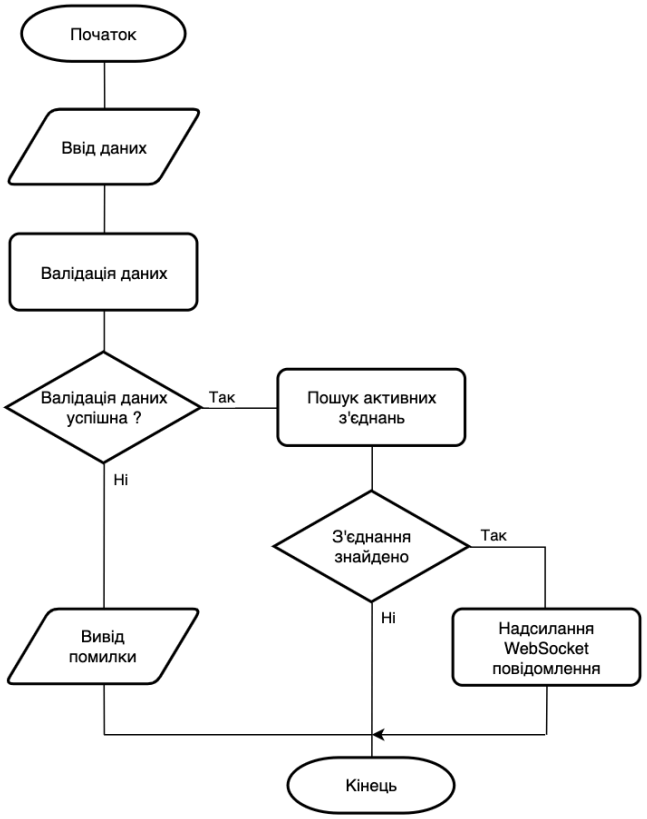


Рисунок 2.4 – Схема алгоритму надсилання повідомлення через WebSocket

Алгоритм отримання повідомлень клієнтом реалізується через обробку подій onMessage на стороні клієнтського WebSocket-з'єднання. При отриманні нового повідомлення від сервера, клієнт оновлює інтерфейс користувача, відображаючи отримане повідомлення в відповідному чаті.

2.3.4 Алгоритм управління WebSocket-з'єднаннями

Алгоритм управління WebSocket-з'єднаннями реалізується на стороні Swoole-сервера через обробники подій onOpen, onMessage, onClose.

При встановленні нового WebSocket-з'єднання (onOpen), сервер отримує файловий дескриптор з'єднання. На цьому етапі відбувається аутентифікація користувача через передачу токена. Після успішної аутентифікації, сервер зберігає відповідність між з'єднання та ідентифікатором користувача у тимчасовому сховищі (таблиця Connections). Це дозволяє швидко ідентифікувати користувача за його з'єднанням. Схема обробки WebSocket-з'єднання зображено на рисунку 2.5.

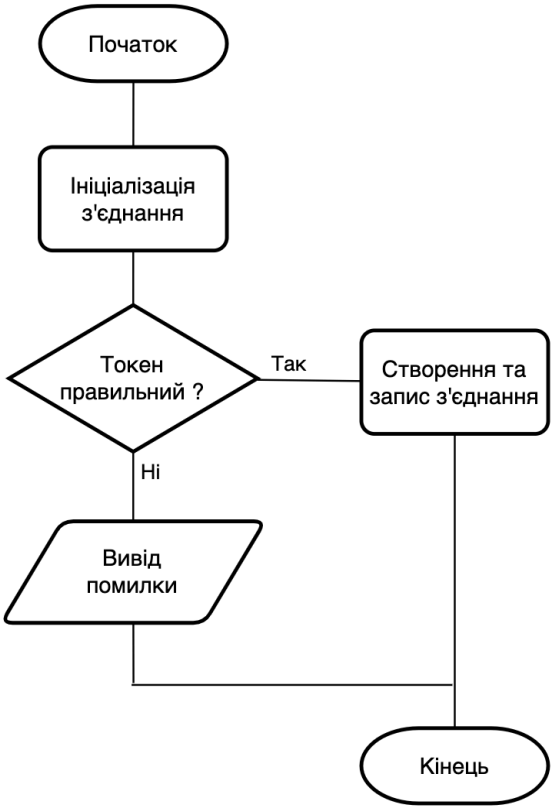


Рисунок 2.5 – Схема алгоритму обробки нового WebSocket-з'єднання

При отриманні повідомлення від клієнта (onMessage), сервер обробляє його згідно з алгоритмом надсилання/отримання повідомлень, описаним вище.

При закритті WebSocket-з'єднання (onClose), сервер видаляє відповідний запис про з'єднання з тимчасового сховища. Це важливо для коректного управління активними користувачами та уникнення надсилання повідомлень на неіснуючі з'єднання.

2.4 Специфікація програмного інтерфейсу API та WebSocket

Програмний інтерфейс системи складається з двох основних частин: RESTful API для управління користувачами та чатами, та протоколу обміну даними через WebSocket для функціональності реального часу.

2.4.1 Специфікація API інтерфейсу

RESTful API використовується для виконання операцій, які не вимагають обміну даними в реальному часі, таких як реєстрація, аутентифікація, отримання списку чатів користувача, створення нових чатів, додавання/видалення учасників чату, отримання історії повідомлень чату. На основі наданої структури маршрутів, визначено наступні кінцеві точки API:

- POST /api/register : Реєстрація нового користувача. Ця кінцева точка використовується для створення нового облікового запису в системі, приймаючи дані користувача, такі як електронна пошта та пароль.
- POST /api/login : Аутентифікація користувача. При успішній аутентифікації за допомогою логіна та пароля, сервер повертає аутентифікаційний токен, який використовується для доступу до захищених ресурсів API.
- POST /api/logout : Вихід користувача із системи. Ця дія анулює поточний аутентифікаційний токен користувача.
- POST /api/message/send : Надсилання текстового повідомлення. Використовується для відправки повідомлення в існуючий чат.

- POST /api/message/read : Позначення повідомлення як прочитаного. Дозволяє клієнту повідомити сервер про прочитання конкретного повідомлення.
- GET /api/message/new : Перевірка наявності нових повідомлень та отримання інформації про непрочитані повідомлення.
- GET /api/chat/all: Отримання списку всіх чатів, в яких бере участь поточний аутентифікований користувач.
- GET /api/chat/messages: Отримання історії повідомлень для конкретного чату. Вимагає ідентифікатор чату та підтримує пагінацію.
- GET /api/chat/users: Отримання списку користувачів, які є учасниками конкретного чату.
- PUT /api/chat/update: Оновлення інформації про чат.
- POST /api/chat/start: Створення нового чату.
- GET /api/users: Отримання списку користувачів системи.

Всі запити до захищених кінцевих точок API (крім /register та /login) повинні супроводжуватися дійсним аутентифікаційним токеном, що передається у заголовку Authorization у форматі Bearer Token. Відповіді від API повертаються у форматі JSON, містять дані або інформацію про статус виконання операції.

2.4.2 Специфікація WebSocket подій

Протокол WebSocket використовується для забезпечення обміну даними в режимі реального часу між сервером та клієнтами. Взаємодія через WebSocket відбувається за допомогою JSON-об'єктів. Кожен такий об'єкт обов'язково містить поле event, яке ідентифікує тип події, та може містити додаткові поля з даними, специфічними для конкретної події. Після встановлення WebSocket-з'єднання, клієнт повинен пройти аутентифікацію, надіславши аутентифікаційний токен у полі token.

Реалізовано наступні події, що надсилаються клієнтом до сервера:

- Подія send_message використовується для надсилання нового текстового повідомлення в конкретний чат. При надсиланні цієї події клієнт

повинен передати поля chat_id, що вказує на ідентифікатор цільового чату, та text, що містить власне текст повідомлення.

- Подія read_message призначена для позначення певного повідомлення як прочитаного поточним користувачем. Для цього клієнт надсилає ідентифікатор повідомлення у полі message_id.
- Подія ping використовується клієнтською стороною з певною періодичністю для підтримки активності WebSocket-з'єднання та запобігання його закриття через тайм-аут бездіяльності.
- Подія new_message надсилається всім відповідним клієнтам, коли в чаті, учасниками якого вони є, з'являється нове повідомлення. Ця подія містить поле message, значенням якого є JSON-об'єкт, що представляє повний ресурс повідомлення, включаючи його унікальний ідентифікатор (id), ідентифікатор чату (chat_id), ідентифікатор автора (author_id), текст повідомлення (text) та час створення (created_at).
- Подія message_read сповіщає учасників чату про те, що певне повідомлення було прочитано іншим користувачем. Вона містить поля message_id (ідентифікатор прочитаного повідомлення) та user_id (ідентифікатор користувача, який здійснив читання).
- Подія user_online надсилається користувачам, які мають спільні чати з користувачем, щойно встановив активне WebSocket-з'єднання. Вона містить поле user_id з ідентифікатором користувача, який став онлайн.
- Подія user_offline надсилається аналогічним чином, коли користувач закриває своє останнє активне WebSocket-з'єднання і стає офлайн. Вона також містить поле user_id користувача, який перейшов у статус оффлайн.
- Подія pong є відповіддю сервера на отриману від клієнта подію ping. Вона слугує підтвердженням активності сервера та підтримки з'єднання.

Ці специфікації API та WebSocket подій визначають формат взаємодії між клієнтською та серверною частинами системи, забезпечуючи основу для розробки фронтенд-застосунків, здатних повноцінно використовувати функціональність комунікаційного сервісу.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

3.1 Вибір та налаштування програмного забезпечення

Реалізація програмного інтерфейсу комунікаційного сервісу здійснювалась з використанням обраного стеку технологій: Laravel 11, Swoole (через hhxsv5/laravel-s), Docker, Nginx, MySQL та Redis.

3.1.1 Налаштування середовища розробки Docker, Nginx, PHP та MySQL

Середовище розробки було налаштовано за допомогою Docker Compose. Файл docker-compose.yml визначає всі необхідні сервіси та їх конфігурацію.

Сервіс database використовує стандартний образ MySQL 8.0 з налаштуваннями доступу та збереження даних у локальний том.

Сервіс redis використовує стандартний образ Redis Alpine для легкості та швидкості.

Сервіс backend базується на власному Dockerfile, який встановлює PHP разом з необхідними розширеннями, включаючи Swoole. У цьому ж контейнері розгортається код Laravel-додатку та встановлюються його залежності за допомогою Composer. Swoole-сервер запускається як основний процес контейнера.

Сервіс nginx використовує стандартний образ Nginx та налаштовується для прослуховування зовнішніх запитів. Конфігураційний файл Nginx (nginx.conf прописується таким чином, щоб обслуговувати статичні файли, проксувати HTTP-запити до бекенд-сервісу (контейнера backend) за його внутрішньою адресою та портом, проксувати WebSocket-з'єднання до бекенд-сервісу, використовуючи відповідні заголовки для коректної роботи протоколу WebSocket. Приклад конфігурації Nginx для проксування до Swoole-сервера зображено на додатку А.

Ця конфігурація Nginx спрямовує всі запити до бекенд-сервісу, використовуючи proxy_pass. Для WebSocket-з'єднань додаються спеціальні заголовки (Upgrade, Connection) для перемикання протоколу.

Запуск середовища здійснюється командою docker-compose up –build -d, що створює та запускає всі визначені сервіси в ізольованій мережі.

3.1.2 Інтеграція Laravel та Swoole (hhxsv5/laravel-s)

Інтеграція фреймворку Laravel з асинхронним сервером Swoole була реалізована за допомогою пакету hhxsv5/laravel-s. Цей пакет дозволяє запускати Laravel-додаток як довгоживучий процес під управлінням Swoole, що значно підвищує продуктивність порівняно з традиційним PHP-FPM за рахунок відсутності необхідності перезавантажувати фреймворк на кожен запит.

Після встановлення пакету через Composer, необхідно налаштувати параметри Swoole-сервера (порти, кількість воркерів, налаштування WebSocket тощо) у файлі конфігурації config/laravels.php [7].

Для реалізації функціональності WebSocket, пакет hhxsv5/laravel-s надає можливість визначення обробників подій WebSocket (onOpen, onMessage, onClose). Ці обробники є частиною Laravel-додатку і мають доступ до всіх його компонентів (сервісів, моделей, бази даних). Параметри конфігурацій Swoole-сервера [4] зображено на рисунку 3.1.

```
'swoole' => [
    'daemonize'          => env( key: 'LARAVELS_DAEMONIZE', default: false),
    'dispatch_mode'      => env( key: 'LARAVELS_DISPATCH_MODE', default: 2), # previous: 3
    'worker_num'         => env( key: 'LARAVELS_WORKER_NUM', default: 60), # previous: 30
    'task_worker_num'    => env( key: 'LARAVELS_TASK_WORKER_NUM', default: 10),
    'task_ipc_mode'      => 1,
    'task_max_request'   => env( key: 'LARAVELS_TASK_MAX_REQUEST', default: 100000),
    'task_tmpdir'        => @is_writable( filename: '/dev/shm/') ? '/dev/shm' : '/tmp',
    'max_request'        => env( key: 'LARAVELS_MAX_REQUEST', default: 100000),
    'pid_file'           => storage_path( path: 'laravels.pid'),
    'log_level'          => env( key: 'LARAVELS_LOG_LEVEL', default: 4),
    'log_file'           => storage_path(sprintf( format: 'logs/swoole-%s.log', date( format: 'Y-m'))),
    'document_root'      => base_path( path: 'public'),
    'package_max_length' => 4 * 1024 * 1024,
    'reload_async'       => true,
    'max_wait_time'      => 180,
    'enable_reuse_port'  => true,
    'enable_coroutine'   => false,
    'upload_tmp_dir'     => @is_writable( filename: '/dev/shm/') ? '/dev/shm' : '/tmp',
    'http_compression'   => env( key: 'LARAVELS_HTTP_COMPRESSION', default: false),
```

Рисунок 3.1 – Конфігурацій Swoole-сервера

3.2 Реалізація програмного інтерфейсу

Реалізація програмного інтерфейсу включала створення моделей даних, розробку сервісів для бізнес-логіки, написання обробників WebSocket-подій та реалізацію HTTP контролерів.

3.2.1 Реалізація моделей даних

На основі спроектованої моделі даних були створені відповідні моделі Eloquent в Laravel. Кожна модель представляє таблицю в базі даних та визначає її поля, зв'язки з іншими моделями. Наприклад, модель User, Chat, ChatMessage, ChatUser. Модель Connection що зберігає в собі всі активні з'єднання до сокетів. Приклад структури моделі ChatMessage зображено на рисунку 3.2

```php
class ChatMessage extends Model
{
    use HasFactory;

    protected $table = 'messages';

    no usages
    protected $fillable = [
        'id',
        'chat_id',
        'sender_id',
        'reply_message_id',
        'text',
        'is_read',
        'is_pinned',
        'created_at'
    ];

    protected $casts = [
        'is_read' => 'boolean',
        'is_pinned' => 'boolean'
    ];

    public function chat()
    {
        return $this->hasOne( related: Chat::class, foreignKey: 'id', localKey: 'chat_id');
    }

    public function sender()
    {
        return $this->hasOne( related: User::class, foreignKey: 'id', localKey: 'sender_id');
    }
```

Рисунок 3.2 – Модель ChatMessage.php

Це модель повідомлення чату, має в собі поля chat_id, user_id, text та зв'язки з моделями Chat та User.

3.2.2 Реалізація сервісів для роботи з чатами та повідомленнями

Бізнес-логіка системи була реалізована у вигляді сервісних класів. Такий підхід дозволяє відокремити логіку від контролерів та обробників подій, роблячи код більш організованим та легким для тестування. Наприклад, були створені класи ChatService та ChatMessageService.

Клас ChatMessageService містить методи для збереження нових повідомлень в базі даних та отримання історії повідомлень для чату.

Клас ChatService містить методи для створення чатів, додавання/видалення користувачів, отримання списку чатів користувача. Приклад методу з ChatService для створення та отримання чату зображено на рисунку 3.3.

```php
class ChatService
    1 usage
    public static function getChatWithConnections($chatId)
    {
        return Chat::query()
            ->with(['users', 'users.connections'])
            ->find($chatId);
    }

    1 usage
    public function createUserChat(User $user, User $targetUser, string|null $name = null)
    {
        $userName = $this->generateUserChatName($user);
        $targetUserName = $this->generateUserChatName($targetUser);

        $chat = Chat::create([
            'type' => Chat::TYPE_USER,
            'name' => $name ?? substr( string: "$userName / $targetUserName", offset: 0, length: 120)
        ]);

        $chat->users()->sync([
            $user->id => [
                'chat_name' => $targetUserName,
            ],
            $targetUser->id => [
                'chat_name' => $userName,
            ]
        ]);

        return $chat;
```

Рисунок 3.3 – Функції з ChatService.php

Цей приклад демонструє використання моделей Eloquent та транзакцій бази даних для створення та отримання чату з учасниками.

3.2.3 Реалізація обробників WebSocket-подій

Обробники WebSocket-подій (onOpen, onMessage, onClose) були реалізовані в класі, який конфігурується в пакеті hhxsv5/laravel-s як обробник WebSocket. Цей клас може використовувати сервісні класи Laravel для виконання необхідної логіки. Весь сервіс можна переглянути на додатку Г. Опрацювання самого повідомлення можна переглянути на додатку Д.

Метод onOpen відповідає за обробку нового з'єднання, аутентифікацію та збереження інформації про з'єднання.

Метод onMessage обробляє вхідні повідомлення від клієнтів, парсить JSON-дані, визначає тип події та викликає відповідні методи сервісних класів та логіку опрацювання повідомлень через Swoole-сервер. Схему обробки та надсилання повідомлень можна переглянути у додатках Б та В відповідно.

Метод onClose видаляє інформацію про закрите з'єднання.

3.2.4 Реалізація HTTP контролерів

HTTP контролери були реалізовані в Laravel для обробки RESTful API запитів. Контролери використовують сервісні класи для виконання бізнес-логіки та повертають відповіді у форматі JSON. Приклад методу контроллера для створення нового чату зображено на рисунку 3.4.

```php
class StartUserChatController extends Controller
    public function __construct(ChatService $chatService){...}

    public function __invoke(Request $request)
    {
        /* @var User $targetUser */
        $targetUser = User::query()
            ->where( column: 'id', $request->get( key: 'user_id'))
            ->first();

        if (!$targetUser) {
            return new JsonResponse(['message' => 'User not found'], status: Response::HTTP_NOT_FOUND);
        }

        /* @var User $user */
        $user = auth()->user();

        if ($user->id == $targetUser->id) {
            return new JsonResponse(['message' => 'Cant create a chat with yourself'], status: Response::HTTP_FORBIDDEN);
        }

        $chatUser = $this->chatService->chatUserExists([$user->id, $targetUser->id], type: Chat::TYPE_USER);

        if ($chatUser) {
            return new JsonResponse(['data' => ['chat_id' => $chatUser->chat_id], 'message' => 'Chat already exist']);
        }

        $chat = $this->chatService->createUserChat($user, $targetUser);
```

Рисунок 3.4 – Контроллер створення нового чату

Цей приклад ілюструє використання валідації запитів, отримання автентифікованого користувача та виклик методів сервісного класу для виконання бізнес-логіки.

3.3 Тестування програмного інтерфейсу

Тестування розробленої системи є важливим етапом для забезпечення її коректної роботи та відповідності вимогам. Було проведено функціональне тестування для перевірки реалізації всіх заявлених функцій, а також концептуально розглянуто підходи до тестування продуктивності.

3.3.1 Методика тестування

Функціональне тестування проводилось шляхом виконання тестових сценаріїв та запитів, що охоплюють основні функції програмного інтерфейсу. Кожен з яких відтворює можливі дії користувача та очікуваний результат. Тестування включає перевірку: процесу реєстрації та аутентифікації користувачів, створення чатів, надсилання та отримання повідомлень в режимі реального часу через WebSocket, перегляду історії повідомлень, обробки помилок та некоректних запитів. Тестування проводилось вручну з використанням інструментів Postman [27] та K6 [28].

Інструмент K6 використовувався для тестування продуктивності, передбачення здатності системи обробляти велику кількість одночасних користувачів та повідомлень. Навантажувальне тестування, яке симулює роботу багатьох клієнтів, що одночасно надсилають та отримують повідомлення через WebSocket.

Інструмент Postman, що показує ключові метрики для оцінки продуктивності включають затримку доставки повідомлень та затримка підключення до серверу. Також через нього проводилось тестування всіх алгоритмів роботи з аутентифікацією, створенням чатів та опрацювання повідомлень на архітектурі RESTful API.

3.3.2 Результати функціонального тестування

За результатами функціонального тестування було підтверджено коректну роботу всіх реалізованих функцій згідно з поставленими вимогами. Процеси реєстрації, аутентифікації, створення чатів (зображено на рисунку 3.5) та обміну повідомленнями працюють належним чином. Результат тестування WebSocket з'єднання та передачі повідомлень зображено на рисунку 3.6.

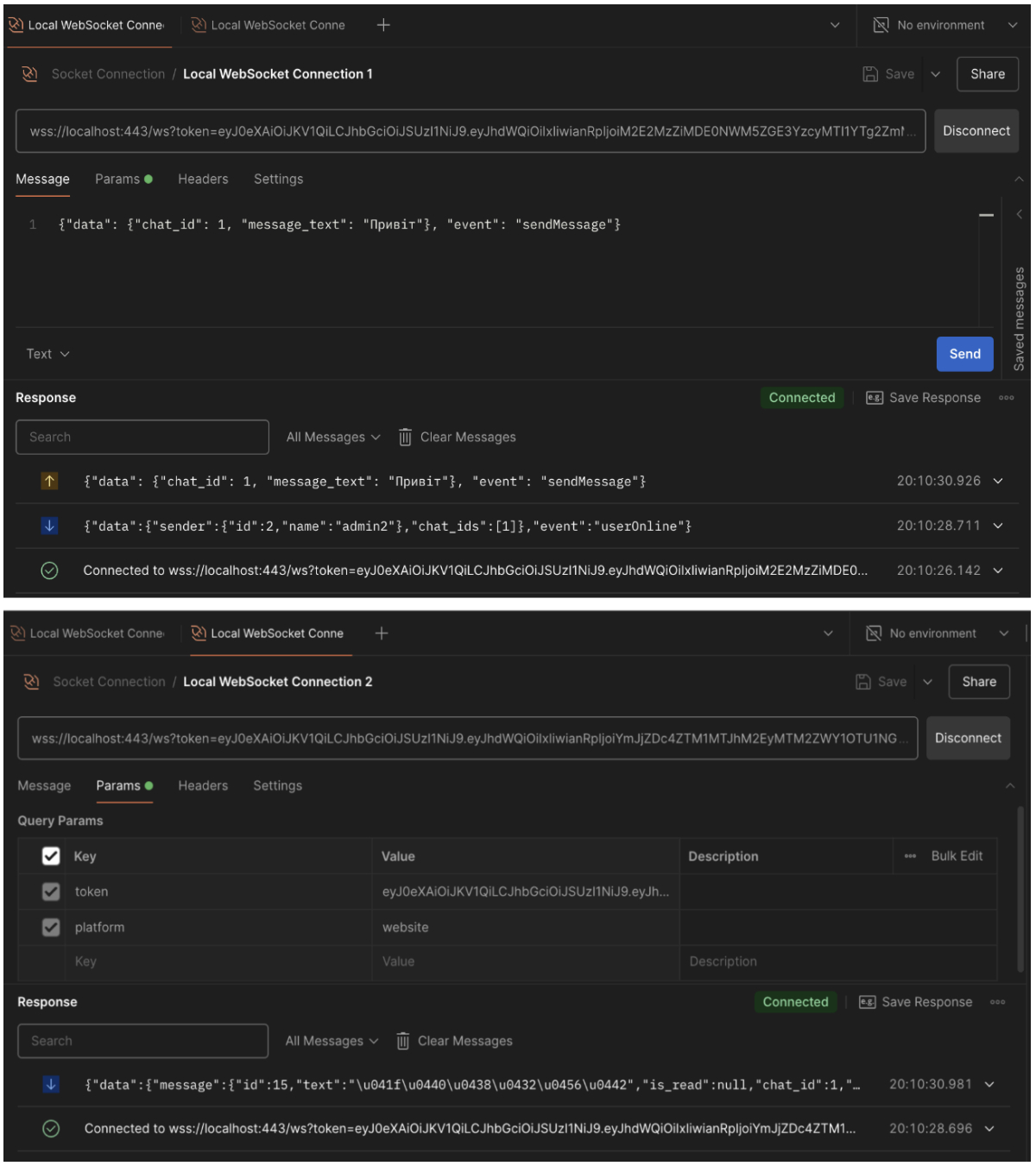


Рисунок 3.5 – Результати функціонального тестування WebSocket сервера

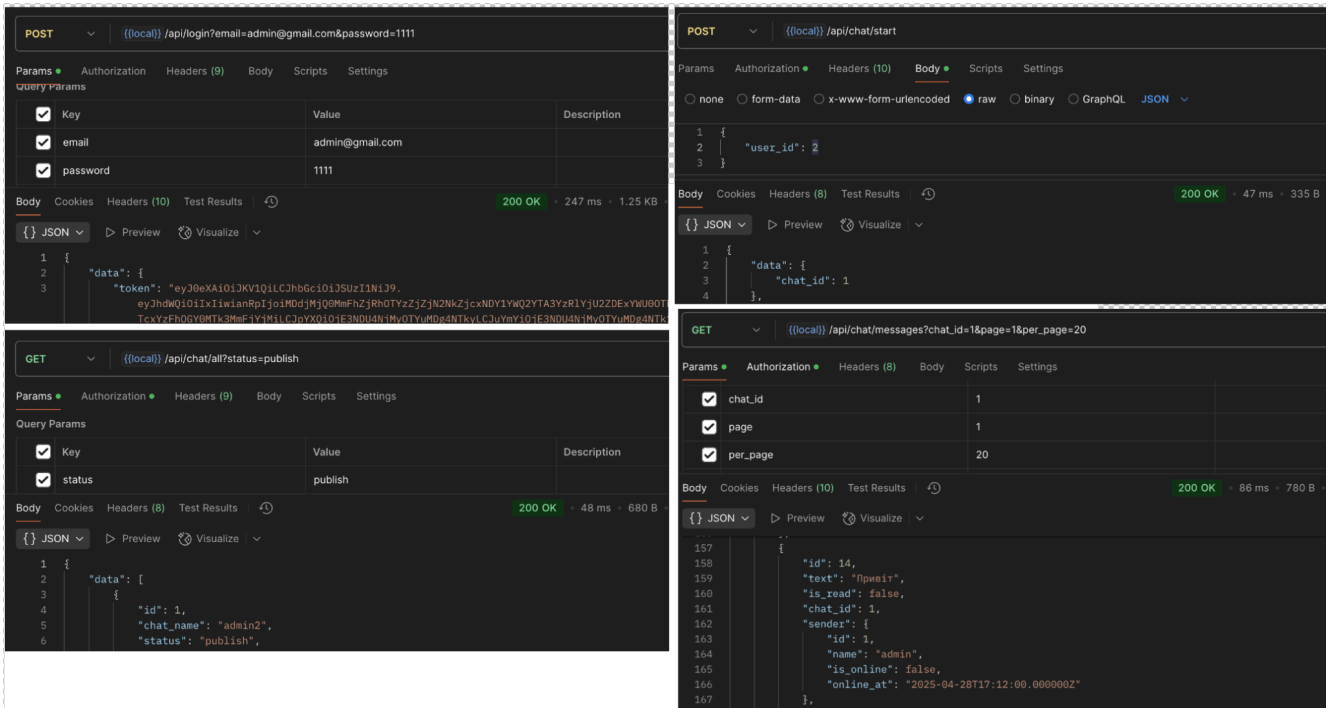


Рисунок 3.6 – Результати функціонального тестування RESTful APIs запитів

3.3.3 Результати тестування продуктивності

Хоча повноцінне навантажувальне тестування виходить за рамки даної роботи, концептуальний аналіз архітектури та використаних технологій дозволяє зробити висновки про потенційну продуктивність. Використання Swoole забезпечує асинхронну обробку та ефективне управління WebSocket-з'єднаннями, що є критично важливим для високонавантажених чат-систем. Архітектура на базі Docker Compose дозволяє легко масштабувати бекенд-сервіс шляхом збільшення кількості його екземплярів, що розподілить навантаження та підвищить загальну пропускну здатність системи.

За допомогою інструменту K6 було проведено навантажувальне тестування WebSocket серверу. Система опрацювала 80719 з’єднань з піковим навантаженням до 15000 віртуальних користувачів із гарантованим успішним встановленням і середнім часом життєвого циклу сеансу 11,9 с. Середня швидкість створення сесій сягала 897 сеансів/с. При цьому спостерігалися прийнятні мережеві витрати (≈ 202 кБ/с на вихідному та 7,8 кБ/с на вхідному трафіку) та стало підтримувалася функціональність обміну повідомленнями (1,8 повід./с), що підтверджує

працездатність обраної архітектури під високими навантаженням.. Результат тестування зображено на рисунку 3.7.

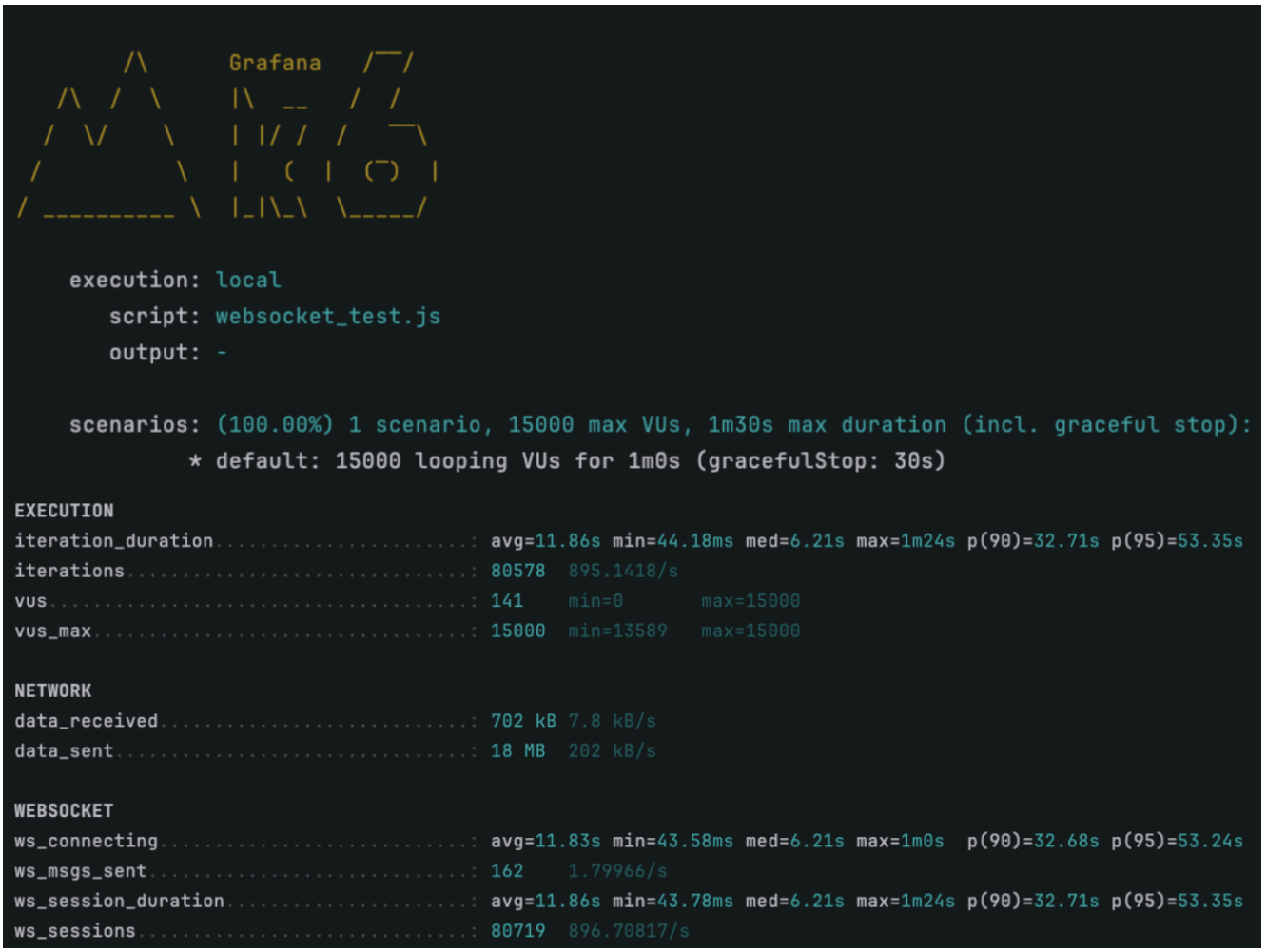


Рисунок 3.7 – Результатів навантажувального тестування WebSocket-з'єднань

ВИСНОВКИ

В результаті виконання роботи отримані наступні результати:

1. Проведено аналіз предметної області, порівняння традиційної моделі PHP-FPM з асинхронними серверами на базі Swoole, переваги WebSocket та роль контейнеризації за допомогою Docker. Наведено порівняння готових рішень, таких як Slack, Zoom, Mattermost.

2. Проведено огляд існуючих рішень і архітектур, представлено сучасні архітектурні підходи (мікросервісна архітектура, брокери повідомлень, кешування, масштабування), включено приклади використання RabbitMQ, Redis, MongoDB.

3. Обґрунтовано вибір стеку технологій таких, як Laravel 11 + Swoole + Redis + MySQL + Docker + WebSocket.

4. Сформовано постановку задачі та сформульовано функціональні, так і нефункціональні вимоги до системи (реєстрація, чати, повідомлення, WebSocket, масштабованість, безпека, відмовостійкість, час доставки < 100 мс).вимог до системи

5. Розроблено архітектуру програмного інтерфейсу у Docker-середовищі, описано компоненти системи та їхню взаємодію через docker-compose.

6. Спроектовано базу даних, представлено модель з п’ятьма таблицями (users, chats, messages, chat_users, connections) з поясненням логіки зв’язків і надано ER-діаграму.

7. Розроблено алгоритмічне забезпечення, описані алгоритми для реєстрації, аутентифікації, створення чатів, обробки WebSocket-повідомлень та керування з’єднаннями.

8. Вибрано середовище розробки та розгортання в якості Docker, Nginx, конфігурацію Laravel-Swoole та запуск через docker-compose.

9. Розроблено програмний інтерфейсу, сервісів, конфігурації, описано структуру ChatService, ChatMessageService, моделі Eloquent.

10. Проведено функціональне тестування з Postman, навантажувальне з K6, отримано результати щодо стабільності й продуктивності.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Hercog D. Communication service, communication protocol. Communication protocols. Cham, 2020. P. 45–65.
2. Dougherty W. PHP programming language for server-side web development. Independently Published, 2022.
3. Lasocha W., Badurowicz M. Comparison of WebSocket and HTTP protocol performance. Journal of computer sciences institute. 2021. Vol. 19. P. 67–74.
4. Swoole documentation. URL: https://wiki.swoole.com/en/#/ (дата звернення: 12.02.2025).
5. John S. C, C ++ and C# programming. Lulu Press, Inc., 2015.
6. Sinha S. Introduction to laravel. Beginning laravel. Berkeley, CA, 2019. P. 1–10.
7. Hhxsv5/laravel-s | Larablocks. URL: https://www.larablocks.com/package/hhxsv5/laravel-s (дата звернення: 03.03.2025).
8. Dilip P. J. An optimized virtualization using docker container. International journal for research in applied science and engineering technology. 2017. Vol. V, no. XI. P. 1074–1078.
9. Kumari S. REST based API. International journal of trend in scientific research and development. 2017. Volume-1, Issue-4. P. 571–575.
10. Kaul V. Strategic corporate communications. International journal of trend in scientific research and development. 2017. Volume-1, Issue-5. P. 803–820.
11. Tomic Z., Jovanovic M. ERP and CRM data integration. Management - Journal for theory and practice of management. 2016. Vol. 21, no. 78. P. 63–72.
12. Słodziak W., Nowak Z. Performance analysis of web systems based on xmlhttprequest, server-sent events and websocket. Information systems architecture and technology: proceedings of 36th international conference on information systems architecture and technology. 2nd ed. Cham, 2016. P. 71–83.

13. Byte P. Benchmarking of PHP application with php-fpm vs swoole/openswoole. URL: https://bytepursuits.com/benchmarking-of-php-application-with-php-fpm-vs-swoole-openswoole (дата звернення: 14.03.2025).

14. Meddaugh J. An introduction to slack, A popular chat app for teams and workplaces. URL: https://www.afb.org/aw/20/8/16736 (дата звернення: 27.03.2025).

15. Jaya P. H., Septiani Y. Zoom vs. microsoft teams: students' preference. Ahmad dahlan journal of english studies. 2023. Vol. 10, no. 1.

16. Mattermost – корпоративний месенджер для обміну повідомленнями. URL: https://bravard.space/mattermost-oglyad-korporatyvnogo-mesendzhera-dlya-obminu-povidomlennyamy (дата звернення: 02.04.2025).

17. Gralio. Zulip vs Rocket.Chat: which is best for your team?. URL: https://gralio.ai/compare/zulip-vs-rocket-chat (дата звернення: 28.03.2025).

18. The PostgreSQL Global Development Group. The postgresql reference manual volume 1: SQL language reference. Network Theory Ltd., 2007. 716 p.

19. Ab M. MySQL language reference. Upper Saddle River : Pearson Education, 2005.

20. Li Z. NoSQL databases. Geographic information science technology body of knowledge. 2018. Vol. 2018, Q2.

21. Alraddadi S., Almotairi S. Performance evaluation for mongodb and redis: a comparative study. Journal of engineering and applied sciences. 2019. Vol. 14, no. 19. P. 7218–7222.

22. Dobbelaere P., Esmaili K. S. Kafka versus RabbitMQ. DEBS '17: the 11th ACM international conference on distributed and event-based systems, Barcelona Spain. New York, NY, USA, 2017.

23. Martin P. Discovery and load balancing. Kubernetes. Berkeley, CA, 2020. P. 101–114.

24. Halpin T., Bloesch A. Data modeling in UML and ORM. Journal of database management. 1999. Vol. 10, no. 4. P. 4–13.

25. Jangla K. Docker compose. Accelerating development velocity using docker. Berkeley, CA, 2018. P. 77–98.

26. Ordonez C., Tahsin Al-Amin S., Bellatreche L. An er-flow diagram for big data. 2020 IEEE international conference on big data (big data), Atlanta, GA, USA, 10–13 December 2020. 2020.

27. API testing using postman tool / P. P. Kore et al. International journal for research in applied science and engineering technology. 2022. Vol. 10, no. 12. P. 841–843.

28. Patel R. Step-by-Step guide to load testing with k6. Medium. URL: https://medium.com/@ravipatel.it/step-by-step-guide-to-load-testing-with-k6-5afb625e231a (дата звернення: 02.04.2025).

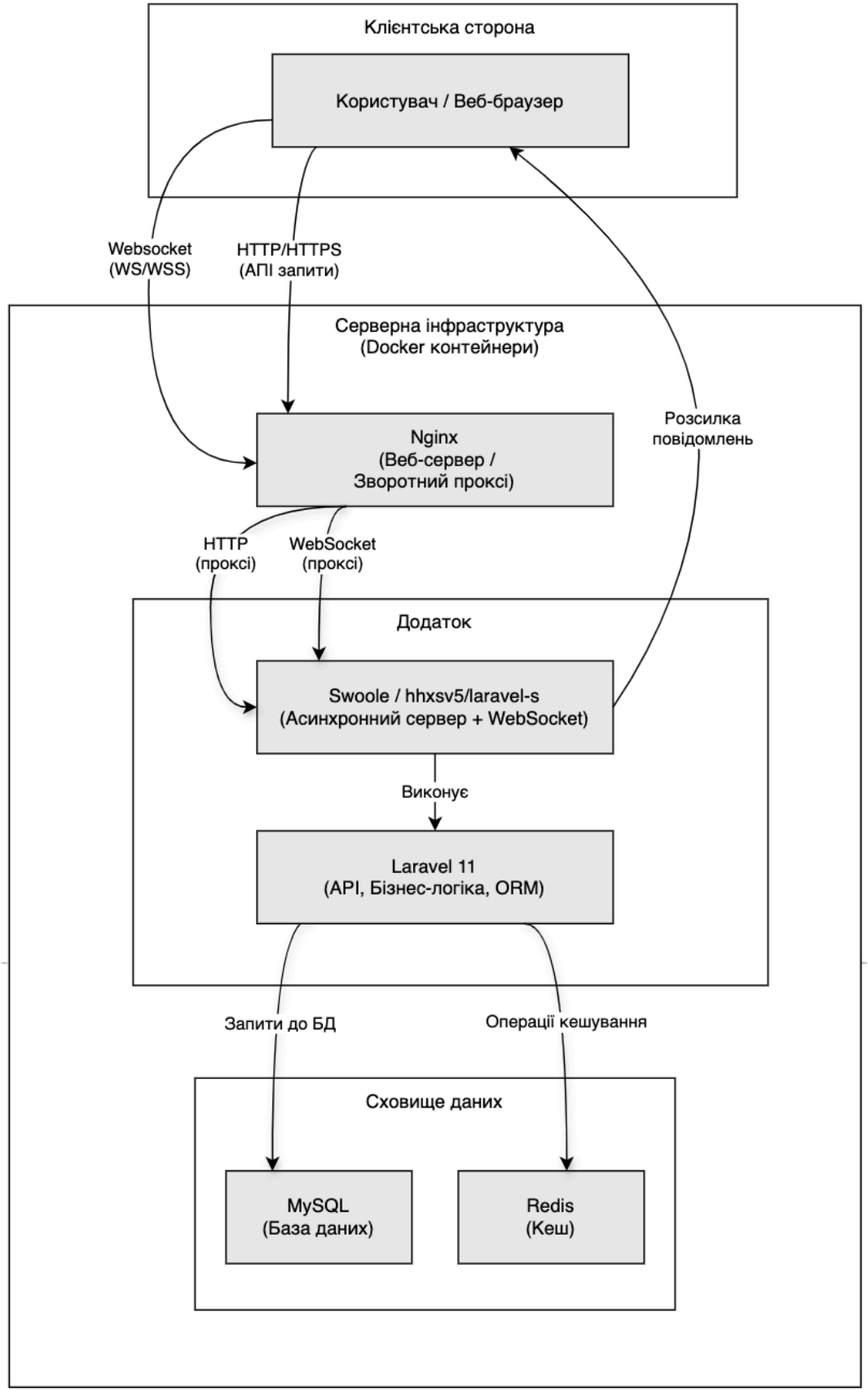
29. Макар М.О. Підвищення продуктивності та масштабованості Laravel-застосунків за допомогою Swoole та Docker. *Стан, досягнення і перспективи інформаційних систем і технологій* : матеріали XXV Всеукраїнської науково-технічної конференції молодих вчених, аспірантів та студентів. Одеса, 17-18 квітня 2025 р. Одеса, Видавництво ОНТУ, 2025. С. 128-130.

30. Комар М.П., Саченко А.О., Васильків Н.М., Гладій Г.М., Коваль В.С., Ліп'яніна-Гончаренко Х.В. Методичні рекомендації до виконання кваліфікаційної роботи з освітньо-професійної програми «Комп'ютерні науки» спеціальності 122 «Комп'ютерні науки» за першим (бакалаврським) рівнем вищої освіти. Тернопіль: ЗУНУ, 2024. 52 с.

31. В. М. Островерхов, Л. І. Біловус, К. З. Возьний, О. О. Луцишин, Г. Л. Монастирський, С. А. Надвиничний, С. В. Питель,С. К. Шандрук., Загальні методичні рекомендації з підготовки, оформлення, захисту та оцінювання кваліфікаційних робіт здобувачів вищої освіти першого (бакалаврського) і другого (магістерського) рівнів. Тернопіль: ЗУНУ, 2024. 83 с.

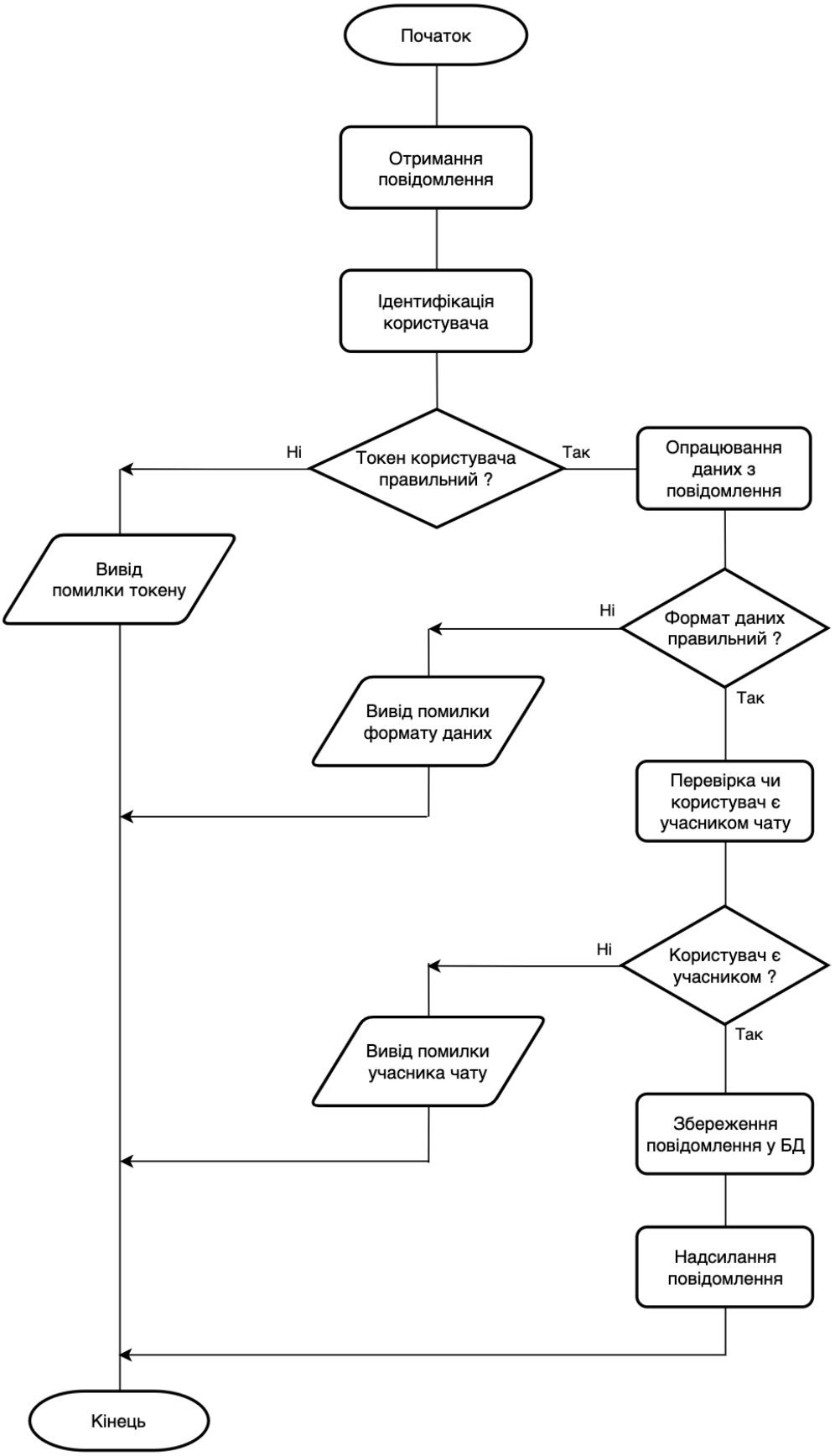
Додаток А

Архітектура комунікаційного сервісу компанії



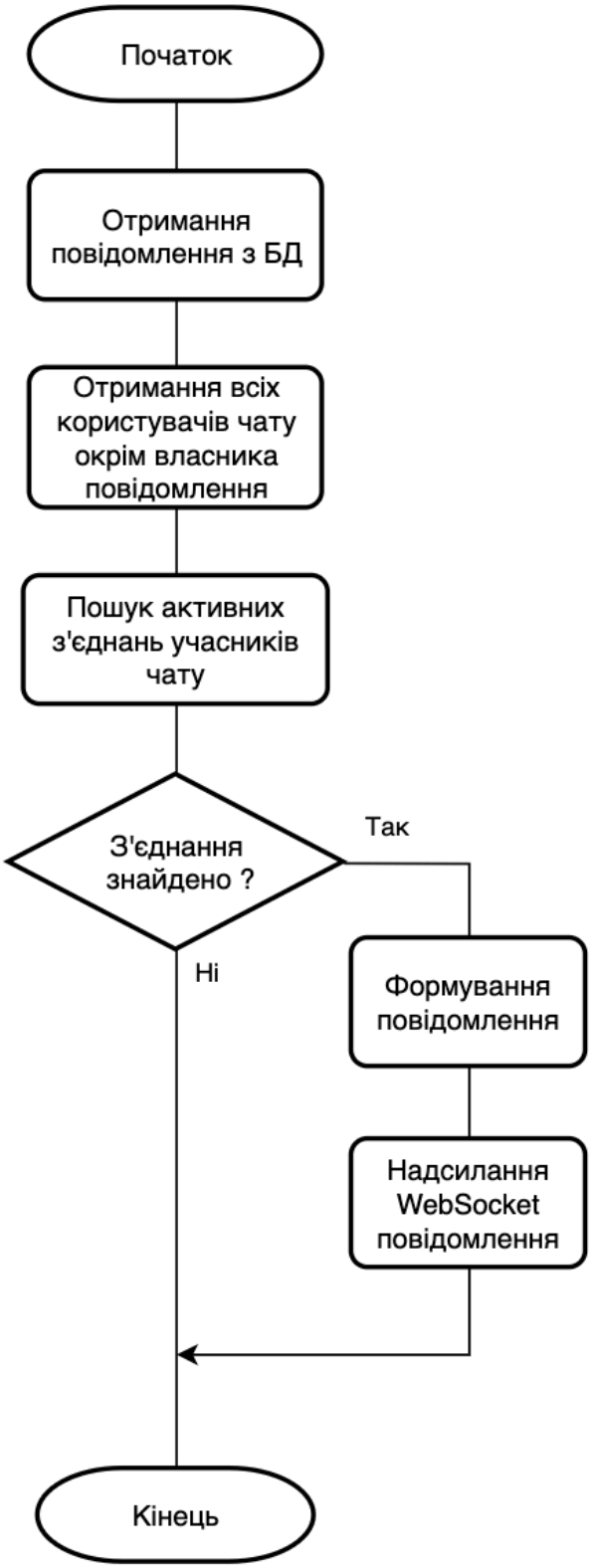
Додаток Б

Опрацювання повідомлень в серверній частині додатку



Додаток В

Надсилання повідомлень в серверній частині додатку



Додаток Г

Основний сервіс обробки WebSocket-повідомлень

```
<?php

...

class WebSocketService implements WebSocketHandlerInterface
{
    public function onOpen(Server $server, Request $request)
    {
        if (!WebSocketAuthService::makeAuth()) {
            $server->disconnect($request->fd, 4001, 'Unauthorized');
            return;
        }

        $user = auth()->user();
        if (!$user) {
            $server->disconnect($request->fd, 4004, 'Not found');
            return;
        }

        $event = new OnOpenEvent($request, $user);
        Event::fire($event);
    }

    public function onMessage(Server $server, Frame $frame)
    {
        if ($frame->data == 'ping') {
             $server->push($frame->fd, 'pong');
        }

            if (empty($frame->data) || !json_validate($frame->data))
return;
        $event = new OnMessageEvent($frame);
        Event::fire($event);
    }

    public function onClose(Server $server, $fd, $reactorId)
    {
        $event = new OnCloseEvent($fd);
        Event::fire($event);
    }
}
```

Додаток Д

Опрацювання WebSocket-повідомлення

```
<?php
class OnMessageListener extends Listener
{
    public function handle(Event $event): void
    {
        /* @var OnMessageEvent $event  */
        $wms = new WebSocketIncomingMessageService();
        $wms->setMessage($event->getMessage())
            ->processMessage();
    }
}

<?php

...

class WebSocketIncomingMessageService implements
WebSocketMessageInterface
{

public function processMessage(): void
    {
        $type = $this->message->getEventType();

        switch ($type) {
            case self::EVENT_SEND_CHAT_MESSAGE:
                $this->sendMessageAction();
                break;
            case self::EVENT_READ_CHAT_MESSAGE:
                $this->readMessageAction();
                break;
        }
    }

private function sendMessageAction(): void
    {
        $data = $this->message->getData();
        $userId = $this->message->getUserId();

        if (!$userId) return;

        $chatMessageService = new ChatMessageService();
        $chatMessageService->createMessage(
            $data['chat_id'],
            $userId,
            $data['message_text'],
            $data['reply_id'] ?? null
        );
    }
```

Додаток Е

Апробація отриманих результатів

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Одеський національний технологічний університет
ННІ Комп`ютерної інженерії, автоматизації робототехніки та програмування ім. П.М.Платонова
Національний технічний університет України
«Київський політехнічний інститут»
Університет Інформатики і прикладних знань м. Лодзь, Польща

XXV Всеукраїнська науково-технічна конференція молодих вчених, аспірантів та студентів

«СТАН, ДОСЯГНЕННЯ І ПЕРСПЕКТИВИ ІНФОРМАЦІЙНИХ СИСТЕМ І ТЕХНОЛОГІЙ»

Матеріали конференції

Одеса

17-18 квітня 2025 р.

Стан, досягнення і перспективи інформаційних систем і технологій / Матеріали XXV Всеукраїнської науково-технічної конференції молодих вчених, аспірантів та студентів. Одеса, 17-18 квітня 2025 р. - Одеса, Видавництво ОНТУ, 2025 р. – 287 с.

Збірник включає матеріали доповідей учасників конференції, які об'єднані за тематичними напрямками конференції.

Збірник буде корисним як для фахівців і працівників фірм, зайнятих в області IT, так і для викладачів, магістрів і студентів вищих навчальних закладів, які навчаються за напрямами і спеціальностями програмного забезпечення, обчислювальної техніки і автоматизованих систем, прикладної математики та обробки інформації, буде корисним професіоналам з комп'ютерного моделювання та розробки комп'ютерних ігор.

Результати досліджень у збірнику представляють собою своєрідний зріз сучасного стану справ в перерахованих галузях знань, який може допомогти як фахівцям, так і студентам університетів скласти загальну картину розвитку інформаційних технологій та пов'язаних з ними питань.

Наукові праці згруповані за напрямками роботи конференції та наведені в алфавітному порядку прізвищ авторів.

Матеріали (тези доповідей) друкуються в авторській редакції.

Відповідальність за якість та зміст публікацій несе автор.

Матеріали подано українською та англійською мовами.

Науковий редактор збірника Зосімов В.В.

комп'ютери [2]. Це значно розширює функціональні можливості та забезпечує постійний доступ до необхідних ресурсів незалежно від місця перебування користувача.

Важливо також приділяти увагу дизайну користувацького інтерфейсу та досвіду (UI/UX), що забезпечує інтуїтивність, естетичність та зручність взаємодії з системою [4]. Добре розроблений інтерфейс зменшує час навчання та підвищує ефективність роботи клініки.

На завершальному етапі проектування важливо провести ретельне тестування системи, щоб виявити і усунути можливі недоліки до її впровадження [3]. Використання автоматизованого та ручного тестування забезпечує стабільність та ефективність системи в реальних умовах. Документування процесу дозволяє забезпечити подальшу підтримку та розширення системи.

Таким чином, інтеграція сучасних веб-технологій у проектування систем управління ветеринарною клінікою є перспективним напрямком, що спрощує рутинні процеси, забезпечує високий рівень обслуговування і комунікації з власниками тварин, підвищуючи загальну ефективність роботи ветеринарних закладів [1].

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. I. Sommerville, Software Engineering: Pearson New International Edition. Pearson Educ., Limited, 2013.

2. Learning Web Design: A Beginner's Guide to HTML, CSS, JavaScript, and Web Graphics. O'Reilly Media, 2018.

3. M. Kleppmann, Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. O'Reilly Media, Inc., 2017.

4. D. A. Norman, The Design of Everyday Things: Revised and Expanded Edition. Basic Books, 2013.

УДК 004.41

ПІДВИЩЕННЯ ПРОДУКТИВНОСТІ ТА МАСШТАБОВАНОСТІ LARAVEL-ЗАСТОСУНКІВ ЗА ДОПОМОГОЮ SWOOLE ТА DOCKER

МАКАР М.О.
(77markiyan77@gmail.com)
Західноукраїнський національний університет

У доповіді розглядається підхід до підвищення продуктивності та масштабованості Laravel-застосунків шляхом використання асинхронного фреймворку Swoole у середовищі Docker. Наведено порівняльну характеристику PHP-FPM та Swoole, представлено архітектуру системи з використанням Redis, MySQL та Docker Compose, а також реалізацію асинхронних задач через hhxsv5/laravel-s. Окреслено типові проблеми та запропоновано шляхи їх вирішення.

Постановка проблеми. В умовах стрімкого зростання кількості веб-користувачів та підвищення вимог до швидкодії застосунків розробники стикаються з проблемами масштабування та продуктивності. Традиційна модель PHP-FPM створює новий процес на кожен запит, що супроводжується накладними витратами на ініціалізацію фреймворку [1]. Це ускладнює обробку великої кількості одночасних запитів у Laravel-застосунках. Особливо актуальним це є для сервісів реального часу та API, які потребують мінімальних затримок і високої пропускної здатності.

Серед альтернатив розглядається використання Swoole – розширення для PHP, що забезпечує асинхронну, подієво-орієнтовану модель виконання з корутинами, дозволяючи зберігати стан і повторно використовувати ресурси між запитами [2]. Однак інтеграція Swoole в Laravel потребує адаптації середовища, врахування можливих витоків пам'яті, конфліктів з пакетами, особливостей керування сесіями та сумісності з Docker-інфраструктурою [5].

Запропоновані рішення. Оптимізація продуктивності Laravel-застосунків досягнута шляхом інтеграції Swoole через пакет hhxsv5/laravel-s у контейнеризованому середовищі Docker. Такий стек забезпечує запуск довготривалого сервера Laravel, який ініціалізується один раз і обробляє вхідні запити без повторного завантаження, що значно знижує накладні витрати [3]. Архітектура реалізована як набір взаємодіючих контейнерів (рис.1):

- Laravel + Swoole. Головний сервер додатку;
- Nginx. Зворотний проксі-сервер;
- Redis. Кешування сесій і обробка черг;
- MySQL. Робота з базою даних;
- Docker Compose. Координація контейнерів і внутрішня мережа socket_custom_network.

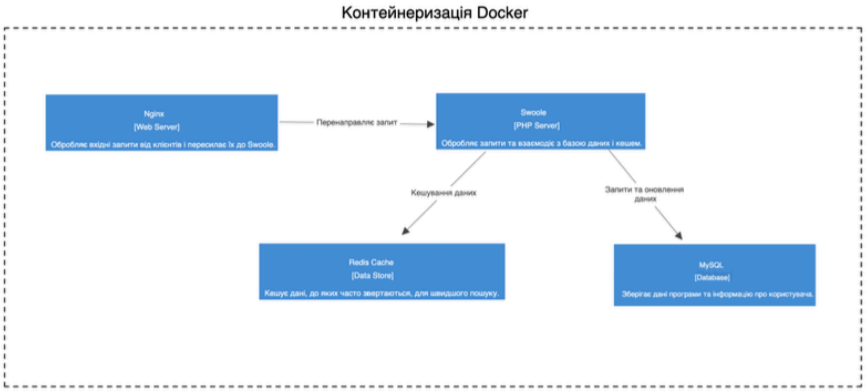


Рис. 1. Архітектура контейнеризованого Laravel-застосунку з Swoole та Docker

У межах реалізації використано можливості асинхронної обробки запитів, фонових задач та WebSocket-з'єднань. Для прикладу, створено Artisan-команду, яка запускає задачу обробки даних користувача у фоновому режимі, не блокуючи основний процес HTTP-запитів.

Для запобігання витокам пам'яті встановлено обмеження max_request у конфігурації сервера Swoole. Очищення глобального стану реалізовано за допомогою middleware. Підбір сумісних пакетів здійснено з урахуванням специфіки довготривалих процесів [5].

Висновок. Використання Laravel 11.x у поєднанні з Swoole та Docker дозволяє суттєво підвищити продуктивність, зменшити час відповіді та покращити масштабованість веб-застосунків. Основною перевагою є уникнення повторної ініціалізації фреймворку для кожного запиту, а також підтримка асинхронної архітектури.

Водночас, успішне впровадження вимагає розуміння принципів роботи з довготривалими процесами, контролю стану, обробки помилок та адаптації до сумісних пакетів. Запропоноване рішення є перспективним для високонавантажених систем, API-сервісів та застосунків реального часу з використанням WebSocket-з'єднань.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. PHP-FPM vs Swoole Execution Model. YouTube, 2025.
2. Blackfire Blog. Unlocking the Power of Asynchronous PHP. 2025.
3. LaravelS. GitHub. URL: https://github.com/hhxsv5/laravel-s
4. Victor Gazotti. How We Increased Our PHP App Performance by 80% with Laravel and Swoole. Medium, 2025.
5. Laminas Docs. Considerations When Using Swoole, 2025.
6. Laravel Framework Issues. GitHub Issue #52774, 2025.

УДК 004.65

ІНФОРМАЦІЙНА СИСТЕМА ЗБОРУ ТА ОБРОБКИ СТАТИСТИЧНИХ ДАНИХ ЩОДО ВІДКЛЮЧЕНЬ ЕЛЕКТРОЕНЕРГІЇ ПОБУТОВИХ КЛІЄНТІВ

МАЛЬЦЕВ А.С., СНІГУР Т.С.
(annaaudia5@gmail.com)
Одеський національний технологічний університет

Метою роботи є проєктування та розробка інформаційної системи, яка дозволяє здійснювати моніторинг та ведення статистики щодо наявності або відсутності електроенергії у побутових клієнтів. Система автоматично фіксує моменти ввімкнення та вимкнення живлення, зберігає ці дані та дозволяє користувачам переглядати статистику у локальній мережі або дистанційно за наявності придбаного віддаленого доступу.

У сучасному світі стабільне електропостачання є критично важливим фактором для забезпечення нормального функціонування побутових, промислових і державних систем. Водночас через різноманітні причини — від аварій на лініях електропередач до запланованих відключень — періодичні вимкнення світла залишаються актуальною проблемою в багатьох регіонах. Особливо це стосується кризових ситуацій або періодів енергетичної нестабільності.